Structural Equivalence in Model-Based Reuse: Overcoming Differences in Abstraction Level

Pnina Soffer

Department of Management Information Systems, Haifa University

Carmel Mountain, Haifa 31905, Israel

Phone: +972-48288506, Fax: +972-48288522

Email: spnina@is.haifa.ac.il

Category: research paper

Structural Equivalence in Model-Based Reuse: Overcoming Differences in Abstraction Level

Abstract

Reuse of models assists in constructing a new model on the basis of existing knowledge, by retrieving a model that matches a preliminary partial input model or some facts about the domain and adapting it to the current needs. It often employs similarity measures to identify reusable models that are structurally and semantically similar to the input model. However, in many cases an input model, being a preliminary one, is of a higher level of abstraction than the detailed models to be retrieved, and structural similarity cannot be detected. This paper proposes the concept of structural equivalence, which means that a detailed model is a refinement of an abstract input model. Measuring structural equivalence rather than structural similarity enables retrieving an appropriate model despite differences in the abstraction level between the models. The paper discusses the structural characteristics of refinement operations in Object-Process Methodology (OPM) models, and presents an algorithm that detects structural equivalence.

1. Introduction

The benefits of applying reuse at various stages of design and implementation have been widely recognized. Reuse of software components has been addressed for over forty years, and the idea has been extended to other and more abstract design tasks and artifacts, such as design specifications (Eckstein et. al, 2001; Kim, 2001; Reinharz-Berger et. al, 2002; Zhang and Lyytinen, 2001), requirements engineering (Lai et. al, 1999; Massonet & Lamsweerde, 1997; Sutcliffe & Maiden, 1998), conceptual models (Pernici et. al., 2000), enterprise modeling (Chen-Burger et. al, 2000), method engineering (Ralyte & Rolland, 2001), and others.

Reuse usually employs a repository of reusable artifacts, a retrieval mechanism that retrieves artifacts that meet criteria posed by the user, and a mechanism that enables the user to adapt the artifact and use it in the current design task. Retrieval can be index-based, according to indices that characterize the artifacts; formal specification-based, by matching formal specifications of the artifact (e.g. signature); or model-based, by matching an input model (query) given by the user with a model stored in the repository (Mili et. al, 1995). The model may be the reusable artifact itself, or its representation. While index-based retrieval is relatively simple and quick, formal specification or model-based retrieval is more accurate, relying on higher volume of information rather than on classification represented by indices.

When the reusable artifact is a model, the purpose of reuse is to assist in constructing a new model, either of the same domain, or of another domain by analogical reasoning. Two types of reusable models can be used for this purpose. One is a generic high-level domain model that has to be specialized in adaptation to the current needs. Retrieval in this case can be index-based, since descriptive indices are sufficient for such generic models. The other type of reusable model is a complete and detailed model that matches partial information available about the domain. 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), found similar to the input model. When model-based retrieval is aimed at retrieving other artifacts represented by the model, the input model can be a detailed or a partial one.

Model-based retrieval may entail different types of similarity measures for matching between the input model and the repository models. Two common similarity measures are entity similarity and structural similarity.

Entity similarity assessment (also called semantic similarity) aims at identifying entities in the reusable models that are semantically similar to entities in the query model. It may employ mechanisms of various accuracy and complexity levels, ranging from identification of identical entity name and type (Soffer, 2002), through thesaurus-based affinity measurement (Castano et. al, 1998; Ralyte & Rolland, 2001), to concept hierarchy-based distance measurement (Chen-Burger et. al, 2000; Lai et. al, 1999).

Structural similarity measurement typically follows the links among the entities in the query model and searches for parallels in the reusable model (Chen-Burger et. al, 2000; Massonet & Lamsweerde, 1997; Ralyte & Rolland, 2001; Sutcliffe & Maiden, 1998). This is sometimes termed neighboring entities search.

Summarizing this, a model is to be retrieved if it includes the same entities and the same links as the input model to some extent. However, if the input model is a preliminary and partial model, and the aim of the retrieval is to obtain an appropriate complete and detailed model, than one cannot expect the input model and the output model to have the same structure and set of links. Rather, the input model would be at a higher level of abstraction, specifying an incomplete set of entities and relationships among them. The same entities would appear in the detailed model along with other entities, and therefore the link structure might be different, including all the other entities that exist in the detailed model.

This paper deals with the assessment of structural similarity between two models. Semantic similarity assessment has been widely addressed, both in the context of reuse (e.g., Ralyte &

Rolland, 2001) and in other contexts, such as schema analysis and integration (e.g., Palopoli et. al., 2003), and the solution approaches suggested seem satisfactory. Structural similarity assessment is particularly problematic when the models being matched are of different abstraction levels. Here we seek for *structural equivalence* rather than similarity, meaning that a detailed model to be retrieved can be perceived as a refinement of an input model, which is of a higher abstraction level. The paper discusses several types of refinement and indicates their structural characteristics, applying 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 paper is organized as follows: Section 2 briefly introduces the Object-Process modeling language; Section 3 discusses different refinement operations and illustrates their outcome in an OPM model; Section 4 describes a rule-based algorithm for identifying structural equivalence of OPM models; Section 5 reviews related work, and concluding discussion is given in Section 6.

2. Object-Process Methodology (OPM)

This section provides a brief introduction to OPM, whose details are provided in Dori (2002). OPM has been applied for various purposes at different design phases and tasks, such as conceptual requirements model (Soffer et. al., 2001), ERP system modeling (Soffer et. al., 2003), web application design (Reinhartz-Berger et. al., 2002), real-time systems specification (Peleg and Dori, 1999), algorithm specification (Wenyin and 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 model of all the system aspects, both structural and dynamic. Structure is expressed by objects related by structural relations such as characterization, aggregation, specialization, and general tagged structural relation. The behavior of a system is represented by a set of procedural links, which can be classified to three classes of links: enabling links, transformation links, and triggering links. Enabling links (e.g. Instrument link) 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 state of the object. Transformation links (e.g. Effect link) relate an object to a process if its state is transformed by the process. Triggering links (e.g. Event link) relate a transformation of an object (reflected in its state) to a process it triggers.

OPM allows 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.

*** Insert Figure 1 about here ***

3. Structural Equivalence

This section discusses different refinement operations and provides observations that characterize their structural impact in an OPM model. We view an OPD as a directed and labeled graph, whose nodes are entities (objects and processes) and edges are the 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 as structurally equivalent to a given link.
Definition 1: Let A, B be entities and let P be a path between A and B. P is *structurally equivalent* to a link of type *l* iff a link *l* between A and B can be replaced by P through a refinement operation.
Notation: $P \cong l$.

3.1 Refinement of structure

The paths established when structure is refined can replace both structural and procedural links that originally exist 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: The direct structural link in the abstract model is replaced by a path including structural links and entities. This is demonstrated in the example shown in Figure 2, in which a Characterization link in the abstract model (a) appears as a path including both Specialization and Characterization links in the refined model (b).

*** Insert Figure 2 about here ***

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 iff $P \cong l$ 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.

Observation 1: Let A, 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 $P \cong l$.

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.

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 3 (a) shows an abstract model including an Effect link between *Engineering Change Processing* and *Item Technical Data*. A refined model (b) shows that the *Item Technical Data* is composed of *Bill of Material* and *Routing*, which are affected by *Engineering Change Processing*, and *Technical Specification*, which remains intact.

*** Insert Figure 3 about here ***

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.

Observation 2: Let A, B, 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) OR (Aggregation) OR (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 3(b) the path between *Engineering Change Processing* and *Technical Specifications* is not structurally equivalent to the Effect link included in it.

3.2 Refinement of behavior

Behavior of a system or a domain is captured by processes. A process can be refined into a sequence of activities (sub-processes) 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 sub-processes 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.

The difficulty in identifying a refined process lies in the fact that unlike 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 a number of 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 structural equivalence. Consider a process of a high level of abstraction, having an initial state and a final state. This process can be refined into many different processes, all having the same initial and final states and subset of interactions as the abstract one. Yet, while being all equivalent to the abstract model, these refined processes are not equivalent to one another. As an example, consider the abstract process of *Supplying Customer Order* in Figure 4(a), which can be refined to the two different processes in Figure 4(b) and (c). These two refined processes have identical initial and final states, as does the abstract process. However, while both processes can be retrieved for a query specifying the abstract model, none of them should be retrieved if the query specifies the other (e.g., if (b) is the query model (c) should not be retrieved). It is therefore easier to formulate a necessary condition rather than a necessary and sufficient condition for structural equivalence of processes.

*** Insert Figure 4 about here ***

Observation 3: Let m_1 be a model portion, in which process A transforms an initial state s_1 into a final state s_2. Let E_1 be the set of entities directly linked to A in m_1. Let m_2 be a model portion that refines m_1. Then m_2 consists of a path P, and a set E_2 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 $E_1 \subseteq E_2$.

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, which might not be sufficient for the identification of equivalence. Notice, that it is likely that at least one of the sub-processes 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 affinity detection techniques (which are assumed to be used, although they are not the focus of this paper). 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. Castano et. al. (1998), suggesting a semi-automatic procedure for abstracting a database schema, 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 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. If such paths exist in a detailed model, and if they are structurally 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 structurally 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 backwards, 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 4(b) 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 (a).

Observation 4 provides a sufficient condition for identifying structural 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, etc. In such cases the path between the initial and final state 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.

4. Tracking Structural Equivalence

Section 3 identified conditions that enable the detection of structural equivalence. When models are retrieved from a repository, we seek for a measure that expresses the level of structural equivalence rather than a Boolean indicator. Such a measure should indicate the likelihood that one model is a refinement of another one.

Consider a query OPM model (OPD) being matched against a set of detailed OPM models in a repository of reusable models. The matching includes a semantic affinity measurement, which is out of the scope of this paper, and a structural equivalence measurement, in which the links among the entities in the input model are searched in the reusable model. The commonly applied structural similarity measures typically provide the proportion of matching links with respect to the total number of links in the query model (e.g., Ralyte and Rolland, 2001). Another possibility is Dice's function (Castano et. al, 1998), computed as twice the number of matches over the total number of links in the two models. This metric, clearly, regards a symmetric matching of models, unlike our problem, in which the matching is single sided (the retrieved models have to match an input model). For our structural equivalence measure, if the reusable model includes a link that matches a link of the input model, then it is counted (as in structural similarity measurement). Otherwise, an equivalent path between the source entity and the destination entity of the link is searched and counted if found. The measure is the proportion of links that are successfully matched, either by a link or a path with respect to the total number of links in the query model.

This section describes a rule-based algorithm that identifies equivalent paths for computing a structural equivalence measure. The rules follow the discussion and observations of Section 3.

4.1 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. An Equivalence Condition defines conditions for a path to be equivalent to a link of a certain type. Conditions may specify link types that must be included in a path and their required position: 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*}

Where *Link Type* is the type of link to which the path is to be equivalent, and *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 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 *Required Position* in the path.

Where *Mandatory Type* is a link type that is necessarily included in the path in order to preserve the nature of the interaction, and *Required Position* is the exact position where it should appear.

Required Position: enum (at Source Position, at Destination Position, 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 link) require the Mandatory Type at the source of the path, while links that lead from the process to the object (e.g., Result link, which is a uni-directional Effect link) require the Mandatory Type at the destination of 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 search algorithm. The details of this algorithm are given in the following section.

4.2 Searching an Equivalent Path

Consider a pair of OPDs <Q, R>, where Q is the query model and R is a reusable model being matched. Assume R is searched for a path between two entities that are directly related in Q. The search is performed in steps, whose maximal number is n_R-1, where n_R is the number of entities in R. It applies the following notation:

s: the source entity of the link in Q whose equivalent path is being searched in R.

d: the destination entity of the link in Q whose equivalent path is being searched in R.

$L_M(a,b)$: Let *a* and *b* be entities, then $L_M(a,b)$ is a Boolean variable whose TRUE value indicates the existence of a direct link from *a* to *b* in a model M.

$\text{Link}_M(A,B)$: Let A and B be non-overlapping sets of entities in model M, then $\text{Link}_M(A,B)$ is an indicator expressing the existence of a direct link from an entity in A to an entity in B.

$$\text{Link}_M(A,B) = \begin{cases} 1 & \text{if } \exists\, a \in A, b \in B, \text{such that } L_M(a,b) = \text{TRUE} \\ 0 & \text{otherwise} \end{cases}$$

S_M: The set of entities in model M.

C_i: The set of entities in R to which a path from *s* has been found until the *i*th step of the search.

U_i: The set of entities in R whose relationship with *s* has not yet been investigated by the *i*th step of the search.

C_i and U_i partition S_M so that at each step *i* of the search $S_R = C_i + U_i + \{d\}$. Each entity in R 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 R 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 Max $[\text{Link}_R(C_i,\{d\}), \text{Link}_R(C_i,U_i)*\text{Link}_R(U_i,\{d\})] = 1$.

Proof: Assume a path exists. It can lead from C_i directly to *d*, then $\text{Link}_R(C_i,\{d\}) = 1$. Otherwise it leads from C_i to some entity $a \in U_i$ and from *a* to *d*. Then $\text{Link}_R(C_i,U_i) = 1$ AND $\text{Link}_R(U_i,\{d\}) = 1$. Assume a path does not exist. Then $\text{Link}_R(C_i,\{d\}) = 0$ AND

(1) If $\text{Link}_R(C_i,U_i) = 1$ then $\text{Link}_R(U_i,\{d\}) = 0$.

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

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

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

The equivalence rules ensure that the found path is equivalent to the link being searched. The algorithm, specified in Figure 5, 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 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, 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. One reason is that they cannot be a part of the path according to the Equivalence Conditions, in which case they are excluded at the beginning of the search. The second reason is that their direction is opposite to the search direction. At every step of the search the uni-directional links from the entities of U_i to the entities of C_i are excluded from the search. The last reason applies to the Is-A link, 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 to the entities of U_i are arranged according to the order defined by the relevant Link Selection Rule. The first link according to this order is selected and the entity it relates to is moved to C_i and becomes the Current entity.

Verify_Equivalence: Equivalence Conditions specify for a given link the link type that must be included in the path, and its required position. If the required position is at the source or destination of the path, then all the links from s or to d, which are not of the mandatory type, 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 position is *Anywhere*, the condition is verified by a set of indicators EC_e.

Let e be an entity in C_i, then $EC_e = 1$ iff a link of the mandatory type is in the path from s to e.

Starting at $EC_s = 0$, and let e be moved from U_i to C_i through a link of type t from an entity $a \in C_i$, then

$$\mathrm{EC}_e = \begin{cases} 1 & \text{if } (\mathrm{EC}_a = 1) \text{ OR } (t = \text{Mandatory Type}) \\ 0 & \text{otherwise} \end{cases}$$

When a path is found, then the Equivalence Condition is met only if EC_d =1, in which case Condition = TRUE.

Compute_Cardinality: This operation is performed only when structural relations are searched. 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, 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 *e* be moved to C_i through a link whose cardinality is <SL, SU, DL, DU> from entity $a \in C_i$, then $SL_e = SL_a * SL$, $SU_e = SU_a * SU$, $DL_e = DL_a * DL$, $DU_e = DU_a * DU$.

For example, assume an item is supplied by 0 to 3 suppliers and a supplier has 1 to 2 contact persons (a supplier can supply 1..m items). The aggregated cardinality of the path between an item and a purchasing contact person is <1, m, 0, 6>.

*** Insert Figure 5 about here ***

4.3 An Example

The algorithm steps are illustrated by the example given in Figures 6 and 7, in which a query model (Figure 6(a)) is matched against a detailed model (Figure 6(b)).

*** Insert Figure 6 about here ***

*** Insert Figure 7 about here ***

None of the procedural links specified in Figure 6(a) appears as a direct link in Figure 6(b). Nevertheless, they are all matched by equivalent paths in the detailed model. We shall follow the steps of the search algorithm applied for tracking an equivalent path that matches the Instrument Link from *Production Order BOM* to *Issuing to Production* (Figure 6(a)) in the detailed model (Figure 6(b)), as illustrated in Figure 7.

Step 1: C_1 includes the source entity, *Production Order BOM* (highlighted). The source entity is the Current, 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_1 includes all the other entities in the model, except the destination entity, *Issuing to Production* (highlighted). C_1 is linked to U_1, which is linked to the destination entity, thus the condition of the lemma is satisfied.

Step 2: Following the Instrument link, C_2 includes *Releasing Production Order* as well. 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, since their directionality is opposite to the path direction. C_2 is still linked to U_2, which is linked to the destination entity.

Step 3: Following the Effect link, *Order Documents* is included in C_3. Note, that this is a random choice from the three Effect links that lead from *Issuing to Production*. C_3 is still linked to U_3, which is linked to the destination entity.

Step 4: Following the next Effect link from C_3, *Kitting List* is now added to C_3. C_3 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 input 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 set, the algorithm is able to simply look one step ahead at a time and avoid a recursive back-tracking.

The complexity of the search algorithm is $O(n_R)$, where n_R is the number of entities in R. The search is performed for each link in Q when the structural equivalence measure is computed. Since the maximal number of links in Q is $n_Q(n_Q-1)/2$, the complexity of measuring structural equivalence is $O(n_Q^2 n_R)$. Note, that n_Q is expected to be significantly smaller than n_R.

5. 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 a query model and a model to be retrieved has not, to the best of our knowledge, been explicitly addressed in the reuse literature. Kim (2001) presents an OO model reuse application in which an initial model, including classes and non-specific 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 non-specific links of the input model, and how structurally different from each other are the models that are retrieved.

Structural similarity plays an important role in the works that deal with analogical reasoning (Sutcliffe & Maiden, 1998; Massonet & Lamsweerde, 1997), where models defined 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 & Maiden (1998) the models to be retrieved include a number of layers, each dealing with different information types, going from abstract to detailed. 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 & Lamsweerde (1997), 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, 2001) or on generalization hierarchy (Chen-Burger et. al., 2000). The model used by Ralyte & Rolland (2001) 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.

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 affinity of terms with structural considerations. Schema matching maps elements of one schema to elements of another schema rather than computes 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 et. al., 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 other ones, 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 (Alagic & Bernstein, 2001), which relates to a consistent mapping of schema elements from one schema to another and backwards in the lowest abstraction level. It is defined as structural as opposed to semantic equivalence, which relates to integrity constraints as well.

Similarity assessment of entities, presented by Rodriguez & Egenhofer (1999), relates to parts, functions, and attributes of two entity classes. Their similarity measure applies 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. Yet, 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 with 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.

In summary, the main contribution of this paper as compared to related earlier work is in explicitly addressing models of different abstraction levels, representing both structure and behavior of a domain.

6. Concluding Discussion

Reuse of models and model-based retrieval of artifacts employ in many cases a structural similarity assessment, aimed at retrieving models that are structurally similar to an input model. In this paper we stressed that differences in the abstraction level are likely to exist between an input model and a detailed model to be reused, and therefore structural equivalence is a better measure than structural similarity. Structural equivalence is identified when the detailed model is a refinement of the abstract input model.

The discussion of refinement operations and the observations that characterize their impact on model structure, as well as the search algorithm, address OPM models. Other modeling languages are different mainly in the separation of structural and behavioral aspects of the modeled domain. However, the notion of structural equivalence is of relevance to models independently of the modeling language. Some of the observations made in this paper can easily be generalized and become applicable to other modeling languages. For example, Observation 1, which deals with dominance of structural relations in a path, is not specific to OPM only. Regarding the behavioral aspects, generalization is less straightforward. In multiview modeling languages (e.g., 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 for other modeling languages, while the unique features of the language might be affected mainly through the equivalence rules.

The search algorithm that enables structural equivalence measurement has been implemented in a reuse application that supports business process alignment and gap analysis in the implementation of ERP systems (Soffer, 2002). 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 structural equivalence concept and apply it to other modeling languages that serve in reuse applications, such as UML and ERD.

References

Algaic, S. & Bernstein, P. A. (2001). A Model Theory for Generic Schema Management. *Proceedings of DBPL, Lecture Notes in Computer Science Vol. 2397*, Springer-Verlag, Berlin. 228-246.

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

Chen-Burger Y. H., Robertson D., & Stader J. (2000). A Case-Based Reasoning Framework for Enterprise Model Building, Sharing and Reusing. *Proceedings of the ECAI Knowledge Management and Organization Memories Workshop*, Berlin (2000)

Dori, D., (2002). *Object Process Methodology – A Holistic Systems Paradigm*. Springer Verlag, Heidelberg, New York .

Eckstein S., Ahlbrecht P., & Neumann K. (2001). Increasing Reusability in Information Systems Development by Applying Generic Methods. *Advanced Information Systems*

Engineering. Lecture Notes in Computer Science, Vol. 2068. Springer-Verlag Berlin. 251-266.

Kim Y. J. (2001). An Implementation and Design of COMOR System for OOM Reuse. *Active Media Technology, 6th International Computer Science Conference. Lecture Notes in Computer Science, Vol. 2252*. Springer-Verlag Berlin 314-320.

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, 331-357.

Madhavan, J., Bernstein, P. A. & Rahm, E. (2001). Generic Schema Marching with Cupid. *Proceedings of the VLDB Conference*, Roma, Italy.

Massonet P. & Lamsweerde A.V. (1997), Analogical Reuse of Requirements Frameworks. In: RE'97, *Proceedings of the Third IEEE Symposium on Requirements Engineering*, IEEE Press Los Alamitos CA. 26-37.

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

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, 53-58.

Pernici B., Mecella M., & Batini C. (2000). Conceptual Modeling and Software Components Reuse: Towards the Unification. *Information Systems Engineering: State of the Art and Research Themes*. Springer-Verlag London Berlin Heidelberg , 209-220.

Ralyte J. & Rolland C. (2001). An Assembly Process Model for Method Engineering. *Advanced Information Systems Engineering. Lecture Notes in Computer Science, Vol. 2068*. Springer-Verlag Berlin, 267-283.

Rahm, E. & Bernstein, P. A. (2001). A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10, 334-350.

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

Rodriguez, M. A. & Egenhofer, M. J. (1999). Putting Similarity Assessments into Context: Matching Functions with the User's Intended Operations. *Proceedings of CONTEXT'99, Lecture Notes in Artificial Intelligence Vol. 1688*. Springer-Verlag, Berlin. 310-323.

Soffer, P., Golany, B., Dori B., & Wand, Y. (2001), Modelling Off-the-Shelf Information Systems Requirements: An Ontological Approach. *Requirements Engineering,*. 6, 183-198.

Soffer, P., Golany, B., & Dori D. (2003). ERP Modeling: a Comprehensive Approach. *Information Systems*, 28, 673-690.

Soffer P. (2002). A Methodology for Adapting an ERP System to the Needs on an Enterprise. PhD Thesis, Technion – Israel Institute of Technology.

Sutcliffe A. & Maiden N. A. (1998). The Domain Theory for Requirements Engineering. *IEEE Transactions on Software Engineering*, 24, 174-196.

Wenyin L. & Dori D. (1998). Object-Process Diagrams as an Explicit Algorithm Specification Tool. *Journal of Object-Oriented Programming*, 12, 52–59.

Zhang Z. & Lyytinen K. (2001). A Framework for Component Reuse in a Meta-modelling-Based Software development. *Requirements Engineering*, 6, 116-131.

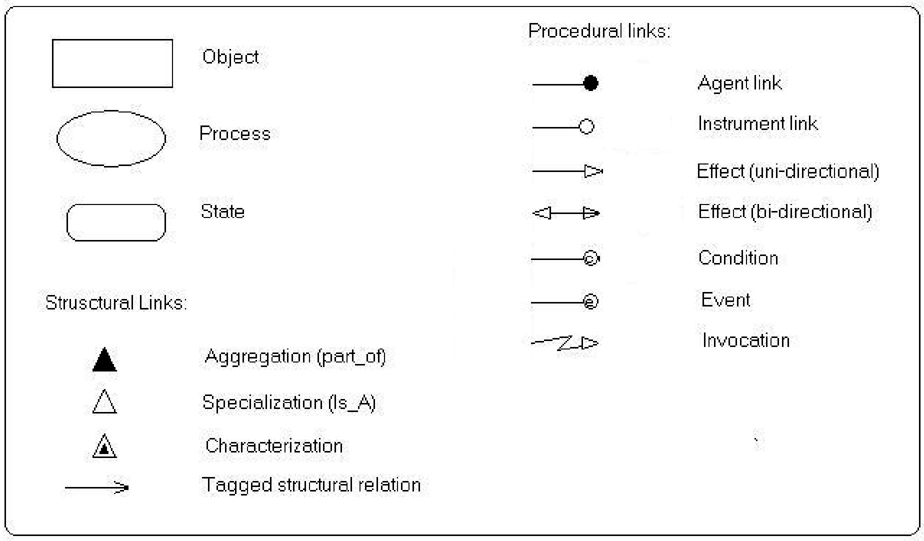

Figure 1: OPM notation

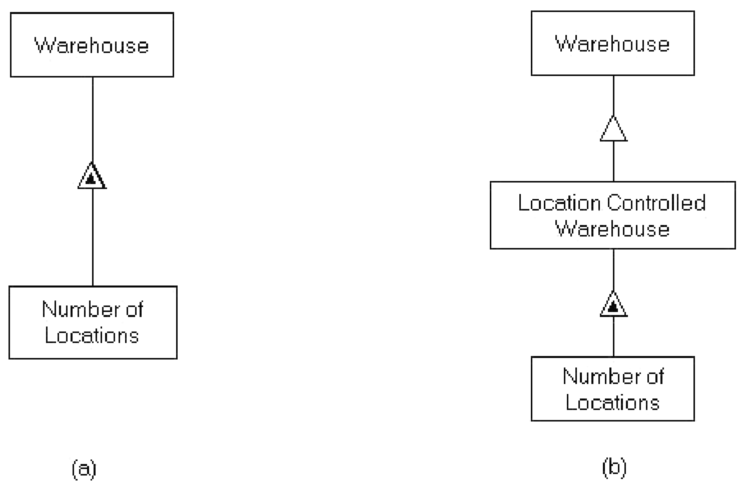

Figure 2: Example of refinement involving a structural link

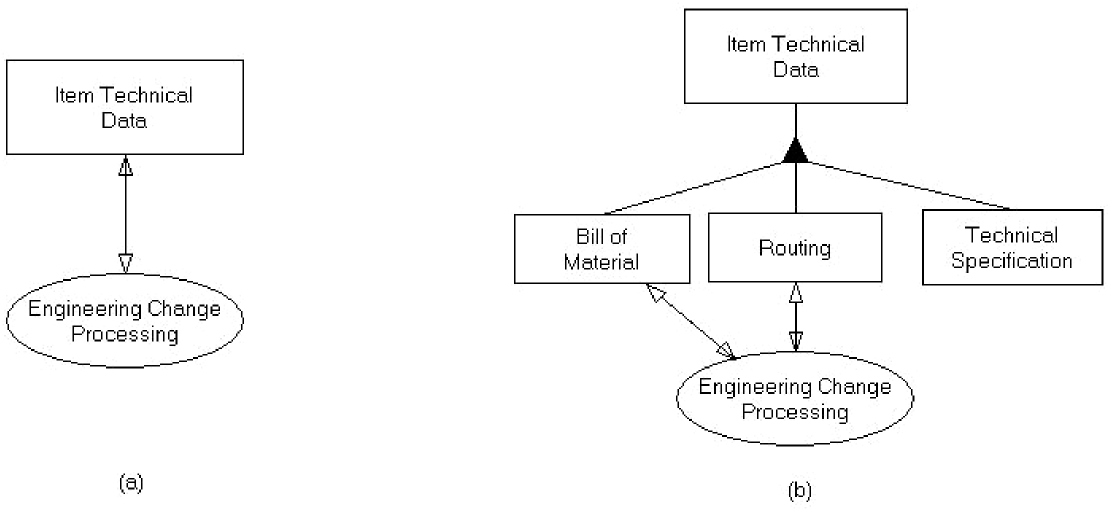

Figure 3: Example of a procedural link in structure refinement

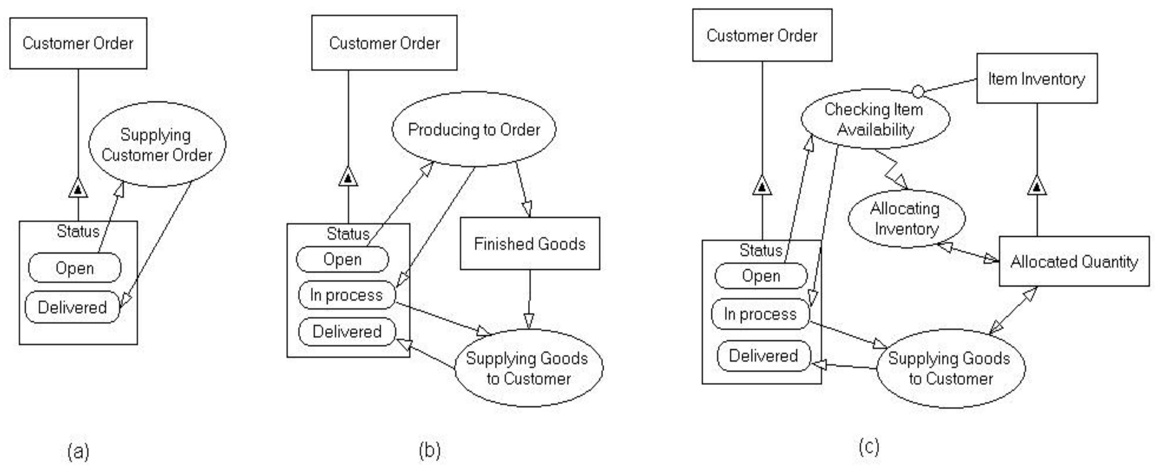

Figure 4: An abstract model and two possible refinements

```
Current = $s$
Fold ($d$)
Exclude_Links
Do while ($Link_R(C_i,U_i)*Link_R(U_i,\{d\})$ = 1) AND ($Link_R(C_i,\{d\})$ <> 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 ($Link_R(C_i,\{d\})$ = 1) AND (Path_Cardinality = Link_Cardinality)
      AND (Condition) then Path_Found
Else Path_Found = FALSE
```

Figure 5: Equivalent path search algorithm

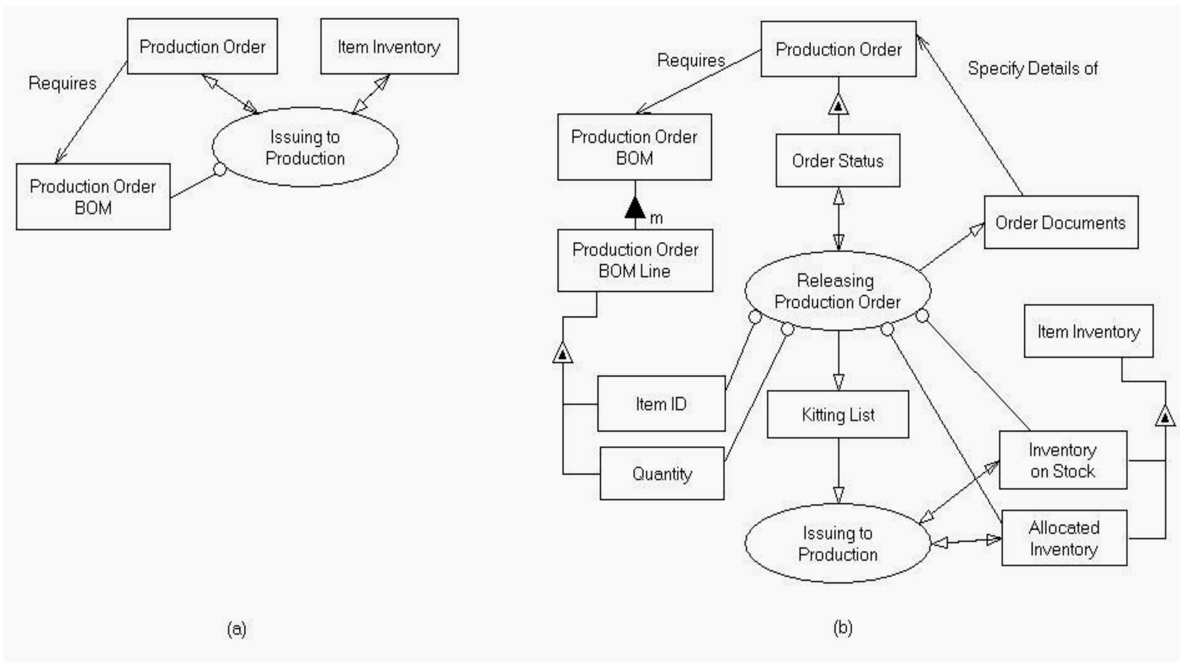

Figure 6: Structural Equivalence Example

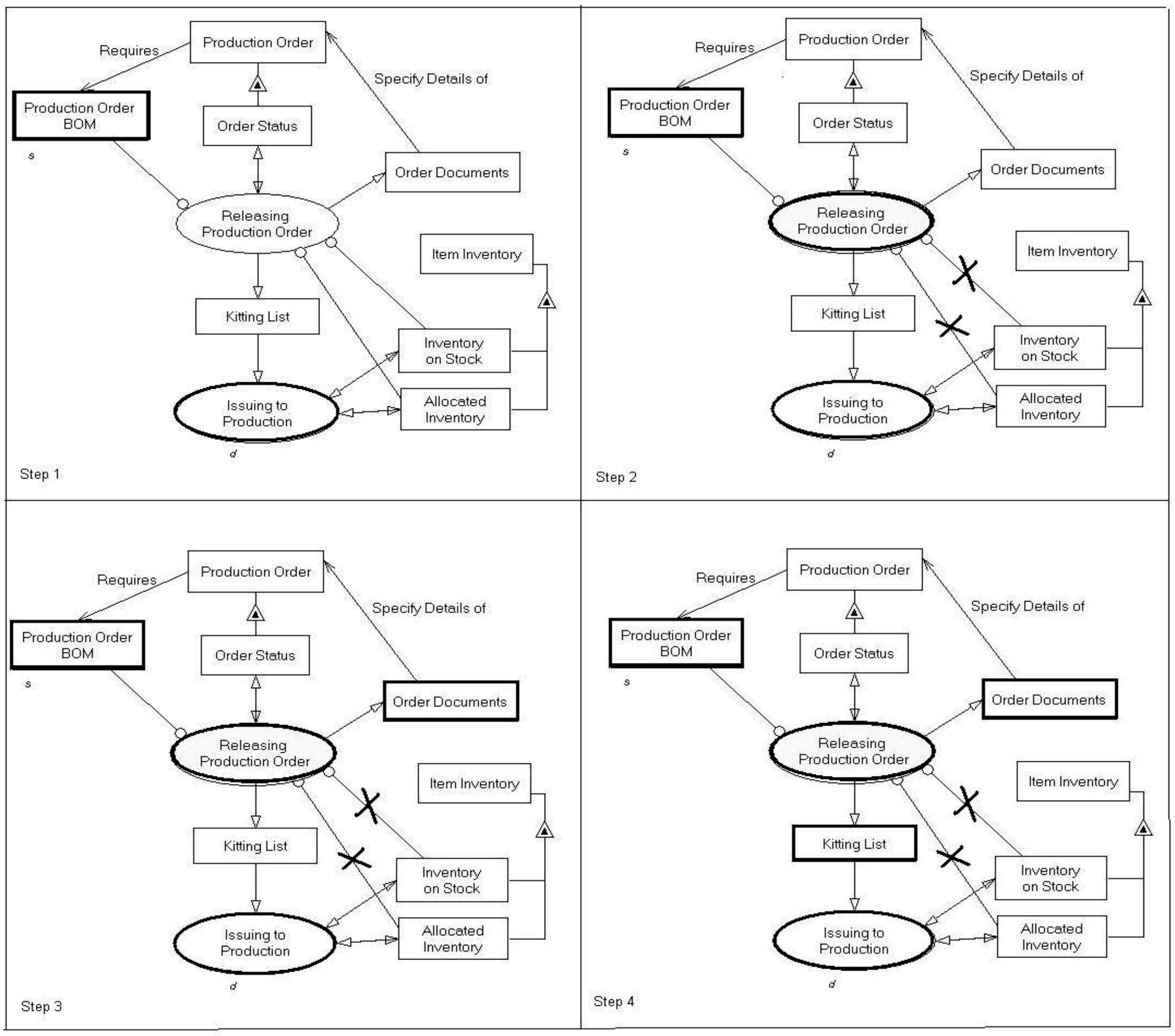

Figure 7: Search Algorithm Steps