Bayesian Reasoning Over Models

Sebastian J. I. Herzig and Christiaan J. J. Paredis

Model-Based Systems Engineering Center (MBSEC),
G.W. Woodruff School of Mechanical Engineering,
Georgia Institute of Technology, Atlanta, GA, USA
`sebastian.herzig@gatech.edu,chris.paredis@me.gatech.edu`
`http://www.mbsec.gatech.edu`

Abstract. A crucial part of verifying and validating models is the identification of inconsistencies. Inconsistencies can exist whenever models overlap semantically. Such overlaps are predominant in model-driven engineering, where the use of multiple viewpoints leads to a variety of incomplete representations of one or more aspects of a system. While the commonly employed rule-based approaches to identifying inconsistencies can be effective, state of the art methods for inferring or determining semantic overlaps are not. Techniques relying on unification algorithms or a unifying ontology make strong assumptions, are error prone and can be costly to maintain. In this paper, an alternative approach based on Bayesian reasoning is proposed. We show how Bayesian inference combined with pattern matching can be used to infer likely semantic overlaps in models. The approach is illustrated and evaluated using the inference of semantic equivalences as an example of inferring one type of semantic overlap.

Keywords: inconsistency management, Bayesian reasoning, verification and validation, model composition

1 Introduction

When designing and developing complex systems, one common practice in model-driven engineering is for stakeholders to study the system from a variety of different viewpoints. Such viewpoints are defined by a number of factors, including the context, level of abstraction and concerns of interest [1]. Concerns of interest addressed from different viewpoints may overlap, leading to semantic relations and, hence, semantic overlaps among the corresponding views and models. Redundant definitions, for instance, imply semantic equivalence. Knowledge of such relations is required when verifying and validating models, particularly because violations of their intended semantics can lead to inconsistencies. While semantic overlaps can certainly be minimized by separating concerns as much as possible, a complete separation of concerns is rarely (if ever) possible.

Several methods for identifying semantic overlaps are proposed in the related literature. However, the associated cost, the strong assumptions made, and the fact that many of the techniques are error prone, renders them impractical for most scenarios. For example, unification algorithms are typically based on name or predicate matching and can fail whenever homonyms or synonyms are encountered. Rule- or logical inference based approaches rely on rule antecedents to be matched completely. However, particularly when reasoning over incomplete information and knowledge, antecedents may not always (fully) match. This can lead to results and conclusions that are unintuitive to a human but logically correct [15]. In fact, there may be cases in which a human would have considered a partial match to provide sufficient evidence to suggest that the consequent of the rule should apply. Such behavior suggests that it is useful to account for uncertainty in automated reasoning processes. In this paper, we use this as a motivation for introducing a novel, Bayesian inference - based inexact reasoning approach for constructing probabilistic arguments about model-based information and knowledge, and apply the developed concepts to the problem of identifying semantic overlaps.

The remainder of the paper is organized as follows: section 2 briefly introduces the running example. Our conceptual approach is developed in section 3. A corresponding algorithm is introduced and evaluated in sections 3.3 and 3.4. Section 4 reviews similar approaches and compares them to our work. The paper closes with a short discussion and conclusions in section 5.

2 Running Example: Inferring Semantic Equivalence

To illustrate our conceptual approach, we apply it to the problem of inferring semantic overlaps. A semantic overlap implies the existence of a semantic relationship between two or more utterances of one or more languages. There are numerous kinds and types of semantic relationships. Some well-known relationships from object-oriented modeling are *meronomous* (part-whole, has a), *hyponymous* ("is a", i.e., type-of), *causal* and *instance-of* relations [12]. Particularly in Model-Driven Engineering (MDE), Model-Based Systems Engineering (MBSE), and generally multi-view and multi-paradigm software engineering, the additional category of *synonymy*-related relationships – which includes *semantic equivalence* – is of interest in identifying model correspondences and for the purpose of defining model transformations. We say that two or more expressions are semantically equivalent if they share a common semantic mapping (meaning).

In our running example, we assume that a number of UML models are given. As illustrated in figure 1a, some of these models contain classes with properties that have default values assigned. It is assumed that, across the different models, some of these properties may be semantically equivalent, but knowledge of their equivalence is not explicitly captured. The task is to find those pairs of properties that are likely to be semantically equivalent. Because models describing engineering systems (software and physical systems) are often heterogeneous in nature, and a great number of very different formalisms is typically employed, we also assume the existence of a (bi-directional) mapping to some common representational formalism for all models. For purposes of illustration and mathematical elegance, and to build on previous work, directed, attributed

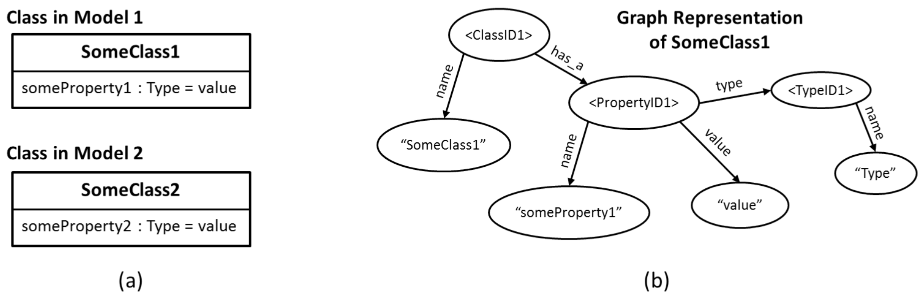


Fig. 1. Running example: (a) UML classes with properties that have default values assigned; (b) automatically generated graph representation of *SomeClass1* (extract).

(and typed) multi-graphs are used as a common representational formalism [9]. This is illustrated in figure 1b.

3 Bayesian Reasoning in Models Represented by Graphs

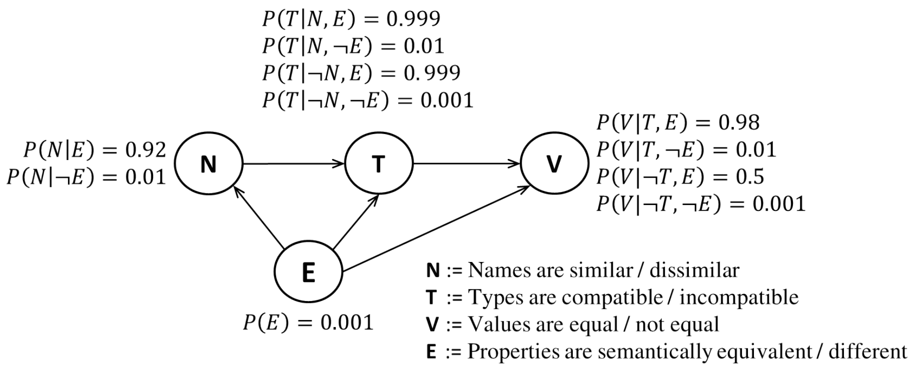
Bayesian reasoning is often considered similar to human reasoning [13]. Illustrative of this are situations in which humans are asked to classify objects. Without any information about the object (other than its existence), a human's belief about the class that the object belongs to is his or her *subjective belief*, which is typically formed on the basis of past experience. Once exposed to the object, a human tends to look for certain *features*, each of which provides further clues towards which class the object is likely to belong to. These features can be identified by *observing* the object – a process that can be interpreted as the collection of additional information (or *evidence*) that can be used to update a belief to form a *posterior belief*. We argue that the same principles can be applied to reasoning over model-based information and knowledge.

3.1 Bayesian Inference & Belief Networks

In Bayesian probability theory, beliefs are updated with new information by applying *Bayes' theorem* [2]. The belief to be updated is then also referred to as the *prior* belief. Assuming that A, B and C are observed events, and given a prior belief about A, the posterior belief about A can be updated with observations B and C using Bayes' theorem:

$$P(A \mid B, C) = \frac{P(A, B, C)}{P(B, C)} = \frac{P(A)\ P(B, C \mid A)}{P(B, C)}\ . \tag{1}$$

Determining the joint probabilities required to compute equation 1 is non-trivial. In part, this is due to values of joint probability distributions rarely being readily accessible. Also, when determining the joint probabilities by means of



$P(T|N,E) = 0.999$
$P(T|N,\neg E) = 0.01$
$P(T|\neg N,E) = 0.999$
$P(T|\neg N,\neg E) = 0.001$
$P(N|E) = 0.92$
$P(N|\neg E) = 0.01$
N
T
V
$P(V|T,E) = 0.98$
$P(V|T,\neg E) = 0.01$
$P(V|\neg T,E) = 0.5$
$P(V|\neg T,\neg E) = 0.001$
E
$P(E) = 0.001$
N := Names are similar / dissimilar
T := Types are compatible / incompatible
V := Values are equal / not equal
E := Properties are semantically equivalent / different

Fig. 2. Bayesian belief network for the running example – nodes denote random variables, arrows denote influences.

factorization, the number of terms required is (in general) very large, even for a small number of random variables [11].

Bayesian belief networks address both the problem of representing the joint probability distribution over a set of random variables and performing inference with these. Formally, a Bayesian belief network is a tuple $(\boldsymbol{G}, \mathcal{P})$, where $\mathcal{P}$ is a joint probability distribution over a set of random variables $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{G}$ is a directed acyclic graph whose nodes are random variables in $\boldsymbol{\mathcal{V}}$. Directed edges are used to indicate *influence* or *causal* relationships, which express local dependence among random variables [11].

In addition to the DAG property, $(\boldsymbol{G}, \mathcal{P})$ must also satisfy the *Markov condition.* The Markov condition states that each variable is (locally) conditionally independent of its non-descendants given its parent variables. This condition, along with information about the conditional dependence significantly reduces the number of terms required to fully define the joint probability distribution $\mathcal{P}$ represented by a Bayesian belief network. Therefore, to determine the joint probability through simple enumeration, the product of the conditional probabilities of all random variables $\mathbf{X}_i \in \boldsymbol{\mathcal{V}}$ given values of their parents $pa\left(\mathbf{X}_i\right)$ (whenever these conditional distributions exist) must be determined [11] (see equation 2):

$$\mathcal{P} = P(\mathbf{X}_1 = x_1, \mathbf{X}_2 = x_2, ...) = \prod_i P(\mathbf{X}_i = x_i \mid pa(\mathbf{X}_i)) \ . \tag{2}$$

This reduces the set of unknowns to only the conditional distributions of the random variables in $\boldsymbol{\mathcal{V}}$ given values of their parents in the Bayesian network. These distributions are known as the *parameters* of a Bayesian network.

Figure 2 illustrates both the structure and the parameters of the Bayesian network used for the running example. Note the use of the short hand notation $P(\mathbf{X}_i = 0) = P(X_i)$ and $P(\mathbf{X}_i = 1) = P(\neg X_i)$ for binary random variables $\mathbf{X}_i$. The network is assumed to be constructed by a human and encodes the assumption that knowing whether or not a particular pair of properties has similar (or dissimilar) names (**N**), compatible (or incompatible) types (**T**), and equal (or unequal) values (**V**) is influenced by whether or not the pair of properties is semantically equivalent (or different) (**E**). In addition, it is assumed that whether or not names are similar influences the probability of types being compatible which, in turn, influences the probability of values being equal. The parameter values shown reflect the subjective beliefs of the same human. For example, at the time of specifying the network parameters, the human believes that the probability of any pair of properties being semantically equivalent is 0.1%.

3.2 Using Pattern Matching to Measure Random Variables

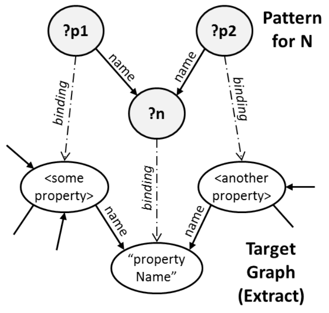
Using the information provided in the Bayesian network illustrated in figure 2, as well as equations 1 and 2, a number of interesting *diagnostic* inferences can be performed. For instance, consider an experiment where the random outcome $\omega \in \Omega$ is a pair of properties from the space of all pairs of properties Ω. Say one can determine (by observing the object) that the pair of properties (ω) has similar names, unequal values and compatible types. Furthermore, say that, due to a lack of available information and knowledge, we cannot be certain about the semantic equivalence of the two properties. Taking all of this new information into account, the probability of semantic equivalence for this particular pair of properties (which, without the observations is only the belief of any pair of properties being semantically equivalent: i.e., $P(E)$) can be updated. Mathematically, this equates to determining $P(E \mid N, \neg V, T)$. Note that $P(E \mid N, \neg V, T)$ is a meaningful statement about the probability of semantic equivalence of *any* pair of properties for which it can be determined with certainty that their names are similar, types compatible and values not equal.

By the earlier assumption that all models are represented by a graph, the definition of the UML class properties considered in the running example must also be represented by a graph (at least at some level of abstraction – see figure 1b). Determining whether or not any two properties represented by a graph fulfill a certain condition (e.g., such as both properties having similar names) can be done computationally by means of graph pattern matching. Therefore, we argue that the process of collecting more information about, e.g., a particular pair of UML class properties can be mapped to querying a graph pattern.

To illustrate this result more formally, let $(\Omega, \mathcal{F}, P)$ represent the probability space over which all random variables $\mathbf{X}_i \in \boldsymbol{\mathcal{V}}$ in the Bayesian network are defined. We define the sample space Ω as the set of all pairs of property definitions $N_{\mathcal{G},prop}$ in the graph $\mathcal{G}$ representing models: $\Omega = N_{\mathcal{G},prop} \times N_{\mathcal{G},prop}$. Furthermore, we define the σ-algebra as $\mathcal{F} = 2^{\Omega}$ (where 2^{Ω} denotes the power set). By definition, a random variable is a mapping $\mathbf{X}_i : \Omega \to E$ from the sample space to some measurable space E. Therefore, an $e \in E$ must be measurable for any $\omega \in \Omega$. By definition of the measurable preimage $\mathbf{X}_i^{-1}(e) \in \mathcal{F}$, an $e \in E$ is measured whenever an event $f \in \mathcal{F}$ is observed. To fully define the mapping, it is sufficient to determine which pairs of properties are elements of the preimages of a random variable. Per our definition of the random variable **N**, the preimage of $\mathbf{N}^{-1}(0)$ is the set of all pairs of properties with similar names, i.e., all $\omega \in \Omega$

Event		Associated Pattern
N	:	`(?p1 name ?n)   (?p2 name ?n)`
¬N	:	`(?p1 name ?n1)  (?p2 name ?n2) notEqual(?n1, ?n2)`
T	:	`(?p1 type ?t)   (?p2 type ?t)`
¬T	:	`(?p1 type ?t1)  (?p2 type ?t2) notEqual(?t1, ?t2)`
V	:	`(?p1 value ?v1) (?p2 value ?v2) equal(?v1, ?v2)`
¬V	:	`(?p1 value ?v1) (?p2 value ?v2) notEqual(?v1, ?v2)`
E	:	`(?p1 equivalentTo ?p2)`
¬E	:	`(?p1 differentFrom ?p2)`

(a)



(b)

Fig. 3. (a) Patterns used in running example; (b) graph representation of the pattern associated with event N with example node variable bindings for a single match.

which have similar names. We argue that determining the pairs of properties (i.e., the ω_is) for which this is the case, is computationally possible by encoding the necessary knowledge in an appropriate pattern.

Figure 3 illustrates the patterns used for the running example in a datalog-like syntax. Variables are unique by name and are indicated by a `?` as prefix. *Graph triples* – i.e., two nodes (a *subject* and an *object*) connected by a directed edge (a *predicate*) – are separated by brackets and are written in the form `(subject predicate object)`. Note that `notEqual(x, y)` and `equal(x, y)` are functors that perform semantic equality checks on their arguments (for instance, 1 and 1.0 are considered semantically equal).

Note carefully that, per the definition of the probability space, every property can and, in a state of perfect information and knowledge, must have a name, type and value. This means that all pairs of properties must be a part of one of the preimages $\mathbf{X}_i^{-1}(e)$. In demonstrating our approach, only binary random variables were used. However, we recognize that multi-valued random variables may be more appropriate in other cases. Also note that, in a state of incomplete or inconsistent information, only a subset of the members of each of the preimages of the random variables can be determined using only the knowledge encoded in a pattern. Additional resources need to be committed to determine set membership for the other pairs of properties. Committing additional resources may require human intervention and the adding of additional information to the models.

3.3 Algorithm

By exploiting the structure of the Bayesian network, inference of probability distributions can be performed quite efficiently, e.g., using the *junction tree* algorithm [11]. Less trivial is the process of collecting information about a particular pair of properties – i.e., computationally determining which information and knowledge should be taken into account when determining the probability of semantic equivalence for a particular pair of properties. For instance, for a pair of

Algorithm 1: Infer propositions and associated probability distributions given a set of changes to a graph and a Bayesian network.

```
Algorithm doInference(Graph $\mathcal{G}$, Triples $T$, BayesianNetwork $B$)
    for $t \in T$ do
        InfContext $\longleftarrow$ observe($\mathcal{G}$, $t$, events($B$), true) ;
        for observations $\in$ InfContext do
            ObservedRVs $\longleftarrow$ observations.getRandomVariables() ;
            for $rv \in$ ($B$.getRandomVariables() $\setminus$ ObservedRVs) do
                $D \longleftarrow D \cup B$.inferDistribution($rv$, ObservedRVs) ;
    return $D$

Procedure observe(Graph $\mathcal{G}$, Triple $t$, Events $E$, Boolean expand)
    for $e \in E$ do
        $p \longleftarrow$ pattern(e) ;
        $T_L \longleftarrow \emptyset$ ;
        if tripleMatchesPartOfPattern($p$, $t$) then
            while ($m \longleftarrow \mathcal{G}$.nextMatch($t$, $p$)) $\neq \emptyset$ do
                $B_s \longleftarrow$ m.getBindingsToSharedVariables() ;
                $Obs[B_s] \longleftarrow Obs[B_s] \cup (e$, variableBindings($m$)) ;
                if expand then
                    for $b \in B_s$ do
                        $T_L \longleftarrow T_L \cup \mathcal{G}$.findTriplesAbout($b$) ;
        for $t_l \in T_L$ do
            $Obs \longleftarrow Obs \cup$ observe($\mathcal{G}$, $t_l$, $E \setminus e$, false) ;
    return $Obs$
```

properties that has no values assigned, but similar types and names, $P(E \mid N, T)$ constitutes a meaningful, and, using the Bayesian network, inferable probability of semantic equivalence, since it takes all of computationally determinable information into account – that is, all of information that can be extracted from the graph solely using graph pattern matching. To determine the probability of semantic equivalence for each individual pair of properties, a naive algorithm would have to iterate over all pairs of properties and, for each pair, a number of graph searches would need to be performed to determine matches to all patterns associated with the random variables. In addition, a potentially large number of inferences in the Bayesian network need to be performed.

Given the complexity of these operations, we propose an incremental algorithm (see algorithm 1) which considers only the changes made to an input graph. The changes are provided in the form of a set of graph triples. The incremental behavior of the algorithm is valid for as long as the structure of the Bayesian network does not change (note that a change in the parameters would only require a re-computation of the posterior beliefs). Verbally, algorithm 1 attempts to measure all random variables within the context of a single triple by matching the associated patterns, followed by performing inference in the Bayesian network. This procedure is called iteratively over the set of added triples: first, the current triple is compared to all patterns. If the triple can be matched against any part of a pattern, a full pattern match in the graph is performed. For each match to the pattern, the value of the random variable and a *context* defined by the bindings to the common, shared pattern variables (in the running example: `?p1` and `?p2`) is stored in a map. To find matches to the other patterns within one variable binding context, a list of triples directly related to the nodes and edges bound to the variables shared across patterns is compiled (i.e., triples with one of the bound elements as a subject, predicate or object). The pattern matching procedure is then repeated over this list.

3.4 Proof-of-Concept Implementation & Algorithm Evaluation

In previous work, we have developed a model-based reasoning framework called ConSystent [9]. ConSystent uses the Resource Description Framework (RDF) as an underlying formalism for representing models by graphs. RDF representations of models are automatically generated. This generated RDF data is collected and stored in a central RDF store (Apache Fuseki).

For the purpose of demonstrating the technical feasibility of our approach, and to test our algorithm, we have extended this existing infrastructure with two additional components: firstly a simple Bayesian network library supporting inference with discrete random variables, and secondly a custom reasoning engine which implements Apache Jena's *Reasoner* interface. The expressiveness for defining patterns using the Apache Jena datalog-like rule language is preserved by internally rewriting the patterns used for measuring random variables as rules with empty rule headers.

A preliminary evaluation of the algorithm and its implementation was performed using three sets of models. In each scenario, different quantities, kinds (UML, Simulink) and sizes of models were used. Samples of the inference results were drawn at random and inspected manually. Two important reflections have been made: firstly, common inferences (such as pairs of properties, for which the only observation is type inequality) can lead to a very large number of inferences, even for small models. This indicates that patterns need to not only be designed carefully, but heuristics may need to be employed to further define which inferences are considered *valuable*. Such heuristics can then form a basis for a classifier that decides which inferences to present to a modeler for further consideration. Secondly, due to the algorithm iterating over the set of all triples, some inferences are performed multiple times. However, this is not considered an issue – rather, it is a likely indicator of a non-optimality of the algorithm.

4 Related Work

In the related literature, most approaches to reasoning over (model-based) information and knowledge are based on logical inference. Inference of semantic overlaps typically makes use of unification algorithms, which exploit representation conventions. This includes predicate matching, of which the work by Finkelstein et al. is an example [6]. Additionally, Triple Graph Grammars (TGG) and, more generally, correspondence models have been used for similar purposes [7]. The approaches are similar in that syntactic matching is performed. Rules are used to define correspondences and transformations, which are typically based on structural semantics. In some instances, models are enhanced with stereotypes [14] or elements from a common, shared ontology [8]. However, all of these methods make a number of strong assumptions: for example, in name- or predicate-based approaches, spelling mistakes or the use of synonyms can produce false negatives. False positives may be produced as a result of homonyms. Common, shared ontologies can be criticized based on the argument that necessitating tagging of models with elements of the ontology is highly labor intensive and agreement of the ontology among stakeholders is difficult to achieve [15]. Secondly, unifying ontologies can quickly grow unmanageably large at which point they become expensive to maintain.

Approaches to inferring semantic overlaps that are based on similarity analysis can be considered complementary to our work. In [3, 5] probabilistic approaches to mediating database schemas are introduced. Probabilistic inference has also been investigated in semantic web applications. For instance, in [10], an extension to RDF was proposed to support uncertain inferences by associating probabilities with both implications and statements. PR-OWL, a proposed probabilistic extension to the *Web Ontology Language* (OWL) to define Bayesian networks, is introduced in [4]. The disadvantage to most of these approaches is that they are either not Bayesian – which, by definition, does not provide a suitable basis for admissible decision rules [2] – or are incomplete, or provide no working implementation.

5 Discussion & Conclusions

In this paper, the use of Bayesian inference in combination with pattern matching is demonstrated and applied to the problem of inferring (likely) semantically equivalences. Identifying semantic overlaps – and generally reasoning over models – is an essential part of identifying inconsistencies and, hence, indispensable to verification and validation of models.

Logical inference can fail in scenarios where incomplete, underspecified and inconsistent views of models are consolidated. Bayesian inference, on the other hand, can always draw useful conclusions. Combining Bayesian inference and pattern matching as described in this paper can be viewed as an extension to the more commonly applied approach of using implications (or, generally, rules) to perform logical inference, and model- and graph-transformations: any outcome with probability 1 or 0 can be said to have been logically entailed by the evidence considered. For these reasons, applications such as *spam filtering* employ a similar combination of Bayesian inference and pattern matching to improve the effectiveness of the reasoning task at hand.

To the best of knowledge of the authors, the combination of Bayesian inference and graph pattern matching has, to the date of writing this paper, not been used within the context of reasoning about properties of engineering models. However, we strongly believe that such an approach is promising, particularly for large-scale model-driven development applications. This is supported by the fact that Bayesian inference allows for rational assessment of important properties, e.g., related to the state of consistency and validity, of incomplete, underspecified and inconsistent models.

Acknowledgments. This work was supported by Boeing Research & Technology. The authors would like to thank Michael Christian (The Boeing Company), Dr. Vinod Cheriyan (MBSEC) and the anonymous reviewers for their feedback.

References

1. ISO / IEC 42010 Systems and Software Eng.: Architectural Description (2007)
2. Berger, J.O.: Statistical Decision Theory and Bayesian Analysis. Springer (1985)
3. Berlin, J., Motro, A.: Database Schema Matching using Machine Learning with Feature Selection. In: Advanced information systems engineering. pp. 452–466. Springer (2002)
4. Costa, P.C., Laskey, K.B.: PR-OWL: A Framework for Probabilistic Ontologies. Frontiers in Artificial Intelligence and Applications 150, 237 (2006)
5. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling Schemas of Disparate Data Sources: A Machine-learning Approach. SIGMOD Rec. 30(2), 509–520 (May 2001), `http://doi.acm.org/10.1145/376284.375731`
6. Finkelstein, A.C., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency Handling in Multiperspective Specifications. IEEE Transactions on Software Engineering 20(8) (1994)
7. Giese, H., Wagner, R.: From Model Transformation to Incremental Bidirectional Model Synchronization. Software & Systems Modeling 8(1) (2009)
8. Hehenberger, P., Egyed, A., Zeman, K.: Consistency Checking of Mechatronic Design Models. In: Proceedings of IDETC/CIE (2010)
9. Herzig, S.J., Qamar, A., Paredis, C.J.: An Approach to Identifying Inconsistencies in Model-based Systems Engineering. Procedia Computer Science (2014)
10. Meiser, T., Dylla, M., Theobald, M.: Interactive Reasoning in Uncertain RDF Knowledge Bases. In: Proceedings of the 20th ACM international conference on Information and knowledge management. pp. 2557–2560. ACM (2011)
11. Neapolitan, R.E.: Probabilistic Reasoning in Expert Systems: Theory and Algorithms. CreateSpace Independent Publishing Platform (2012)
12. OMG: OMG Unified Modeling Language (OMG UML). Tech. rep. (2011), `http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF`
13. Pearl, J.: Fusion, Propagation, and Structuring in Belief Networks. Artificial intelligence 29(3), 241–288 (1986)
14. Shah, A.A., Kerzhner, A.A., Schaefer, D., Paredis, C.J.: Multi-View Modeling to Support Embedded Systems Engineering in SysML. In: Graph Transformations and Model-Driven Engineering, pp. 580–601. Springer (2010)
15. Spanoudakis, G., Zisman, A.: Inconsistency Management in Software Engineering: Survey and Open Research Issues. Handbook of Software Engineering and Knowledge Engineering 1 (2001)