Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 12 Issue 11 Version 1.0 Year 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

A Naive Based Approach for Mapping Two ADL Models

By Sai Bharath Kadati, K. K. Baseer, A. Rama Mohan Reddy & CH. Gowthami

JNIAS/JNTUA, University, Hyderabad. SVEC, Tirupati, and A.P., India

Abstract - In software engineering, we have identified and described the model correspondence problem. To Describe system architecture and artifacts uses models and diagrams. Models contains series of versions. To understand how versions correspondence are difficult. So, we designed a framework based on Search and Ammolite algorithms, which can cardinally finds the correspondence software models. Models are represented as graphs whose nodes have attributes (name, edge, label, connections). For a given diagram pair, it performs different individual matches such as pair-wise match, Split-Merge Match and Drop match and then combine all matches together to design a ADL model. Every ADL Model has its correspondence score for rating quality candidates. To find best Correspondence among the given ADL models uses Search and Ammolite Algorithms.

Keywords : *Decision, Design Artifacts, Elements, Reasoning Principles, Semantic Information, Syntactic Information, Visual Information.*

GJCST-C Classification : *D.2.11*

Strictly as per the compliance and regulations of:

A Naive Based Approach for Mapping Two ADL Models

Sai Bharath Kadati [α], K. K. Baseer [σ], A. Rama Mohan Reddy [ρ] & CH. Gowthami [ω]

Abstract - In software engineering, we have identified and described the model correspondence problem. To Describe system architecture and artifacts uses models and diagrams. Models contains series of versions. To understand how versions correspondence are difficult. So, we designed a framework based on Search and Ammolite algorithms, which can cardinally finds the correspondence software models. Models are represented as graphs whose nodes have attributes (name, edge, label, connections). For a given diagram pair, it performs different individual matches such as pair-wise match, Split-Merge Match and Drop match and then combine all matches together to design a ADL model. Every ADL Model has its correspondence score for rating quality candidates. To find best Correspondence among the given ADL models uses Search and Ammolite Algorithms.

Keywords : Decision, Design Artifacts, Elements, Reasoning Principles, Semantic Information, Syntactic Information, Visual Information.

I. Introduction

An Architecture is defined as building for humans, and being an architect is having the spirit to build for humans. A framework is a collection of classes and applications, libraries of SDKs and APIs to help the different components all work together. In engineering discipline an essential part of quality is control of change. That dictates the need to review and understand changes prior to accept them. Models and Diagrams are a primary design artifacts in this environment, this means being able to compare diagrams to identify correspondence and discrepancies between them. In large-scale IT system development techniques have long existed for comparing textual artifacts, somewhat less work has been reported concerning comparisons of the diagrams and model that are common. The main problem of this paper is to correspondence between **a pair of diagrams** (a mapping between elements of one diagram and elements of the other) and **introduce a Bayesian approach to solve the problem.** The application which are in the central to modern IT systems development process includes structured representation of requirements, business process workflows, system overviews, architectural specifications of systems, network topologies, object designs, state transition diagrams, and control and data flow representation of code.

a) Scenarios

The system development life cycle has several application to find correspondence between models. A series of successive **revisions** of a model from design activity. There is a need to review and understand the nature of revisions as part of accepting them, rejecting them or merging them with other concurrent revisions and to identify correspondences and discrepancies is central to such activities. Model **variants** correspond is crucial for integration. Different collaboration may experiment with different paths of evolution of a model, resulting in a number of transient variants, with the intent that those branches deemed successful will be integrated back into a main stream. The use of multiple **views** of the architecture of the system by using many development approaches and methodologies[6]. The model we propose is made up of five main views [7].

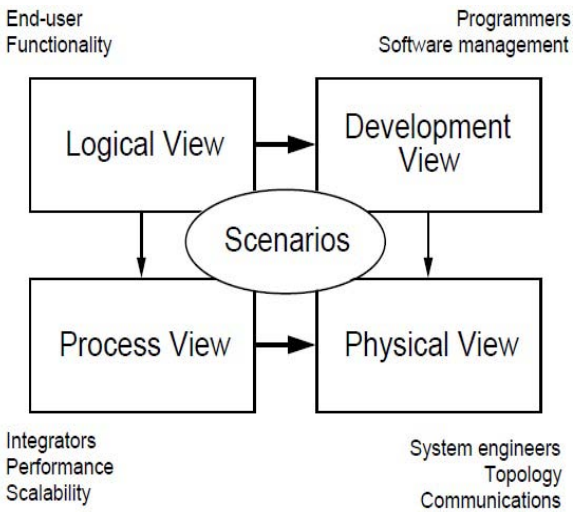

Fig. 1: The 4+1 View Model

- The ***logical*** **view**, which is the object model of the design (when an object-oriented design method is used),
- The ***process*** **view**, which captures the concurrency and synchronization aspects of the design,

Author α : Bachelor of Technology in Computer Science and Engineering and Department of Information Technology, SVEC, Tirupati, and A.P., India.
Author σ : Assistant Professor in department of Information Technology, JNIAS/JNTUA, University, Hyderabad. SVEC, Tirupati, and A.P., India.
Author ρ : Professor & HOD of Computer Science and Engineering, Sri Venkateswara University, Tirupati, A.P., India.
Author ω : B. Tech Computer Science and Engineering from JNTUK, Kakinada, Anantapur, India.

- The *physical* view, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
- The *development* view, which describes the static organization of the software in its development environment.

Traceability is the another important requirement for maintaining quality [10], [5]. Traceability between software artifact, such as requirements, design elements, code, test cases and defect reports. At finer level of granularity, traceability provides the ability to navigate between the elements of different artifacts such as individual software components, hardware nodes, requirements, non-functional requirements, and architectural decisions that reflects the design rationale for the system. The larger asset of reuse is the incorporation of reference architecture from a repository into a solution design.

b) Contribution of the Paper

In Present days, determining correspondences between models is a tedious, error-prone, time-consuming, manual process. The main goal is to achieve an automated means of determining the correspondences, similar to techniques for automated comparison of textual artifacts. This requires us to answer several questions:

- How do we represent models?
- Which features of models must be represented?
- What algorithms should be used to find correspondences?

In this paper, provide answers to these questions.

II. Diagram Features

We focus mainly on the problem of finding correspondences in the domain of IT architecture operational models [2], although the paper techniques have proven effective for other kinds of IT architecture models as well. Operational models are used by IBM Global Services architects as part of a development methodology for customized IT solutions. An operational model also includes model elements reflecting the key decisions constituting the rationale for the solution design.

The main features of an operational model diagram can be abstracted to elements found in many other kinds of diagrams:

- Labeled nodes. System components can be represented as textual or pictorial in a diagram. For example, an attribute may indicate whether the node is internal or external to the solution in an operational model diagram.
- Edges. A edge represents a relationship or association and it can indicate communication paths connection between nodes. Bandwidth, Technology, Security etc., are the attributes of the edge.
- Containers. A node that which contains other nodes is simply called Container. For example, In operational model diagram, a server may contain multiple software components or a region may contain multiple servers. Containers may be nested, current prototype only considers the nesting of servers within regions when correspondences.
- Groups.[8] Nodes are grouped together semantically. For instance, in operational models, servers located in the same building may be grouped within a common region. Like nodes, groups have labels and relationship. For example, regions have an adjacency relationship that indicates a connection.

Regions are discussed in greater detail below.

The information represented by system diagrams can be broadly classified into three types: 1) syntactic information (e.g., nodes, labels, containment, and edges), 2) semantic information (e.g., types, defined semantic attributes), and 3) visual information (e.g., position, shape, and color of diagram elements). Leveraging all of these kinds of information is one of the major challenges of diagram matching.

III. Model Correspondence Problem

The model correspondence problem is the problem of finding the "best" correspondence between the elements of two diagrams.

a) Semantics and Domain-Specific Knowledge as a Basis

The first issue is how to define "best." It may seem appealing to define "best" as the correspondence that preserves a specific semantic relationship between the two diagrams, but this definition would be difficult to apply in practice, for several reasons.

First, there are many possible semantic relationships between diagrams and it is hard to decide which applies. For example, in one case, we may have a diagram pair(E, E'), where E' is a revision of E, with the semantic relation "is a revision of." In another case, E may be a conceptual description of a system and D' a physical description, with the semantic relation "realizes."

Second, even if the semantic relationship is known, defining it in precise detail would be difficult, and even a precise definition may not have sufficient information to find the best correspondence.

Third, many diagrams found in practice have no formal semantics: They use informal notions of "boxes" and "lines" to convey context-specific architectural notions.

Either way, we conjecture that generic matching techniques can go a long way in finding

correspondences between diagrams without having to incorporate knowledge of these kinds of semantic relationships or even knowledge of any of the deeper semantics of the various types of diagram.

b) Reasoning Principles for Recovering Traceability

Human experts can often identify good correspondences after careful examination of a pair of diagrams. Human experts did this by manually finding the best correspondences for some diagram pairs, and recording the reasoning principles used to find the correspondences

The following principles of reasoning about diagram pairs correspondences:

- Most decisions are made using ***evidence*** about which nodes from one diagram match which nodes from the other diagram.
- Every feature of the nodes in the diagrams can be important evidence, including text, connection and containment relationships, and geometric and pictorial attributes.

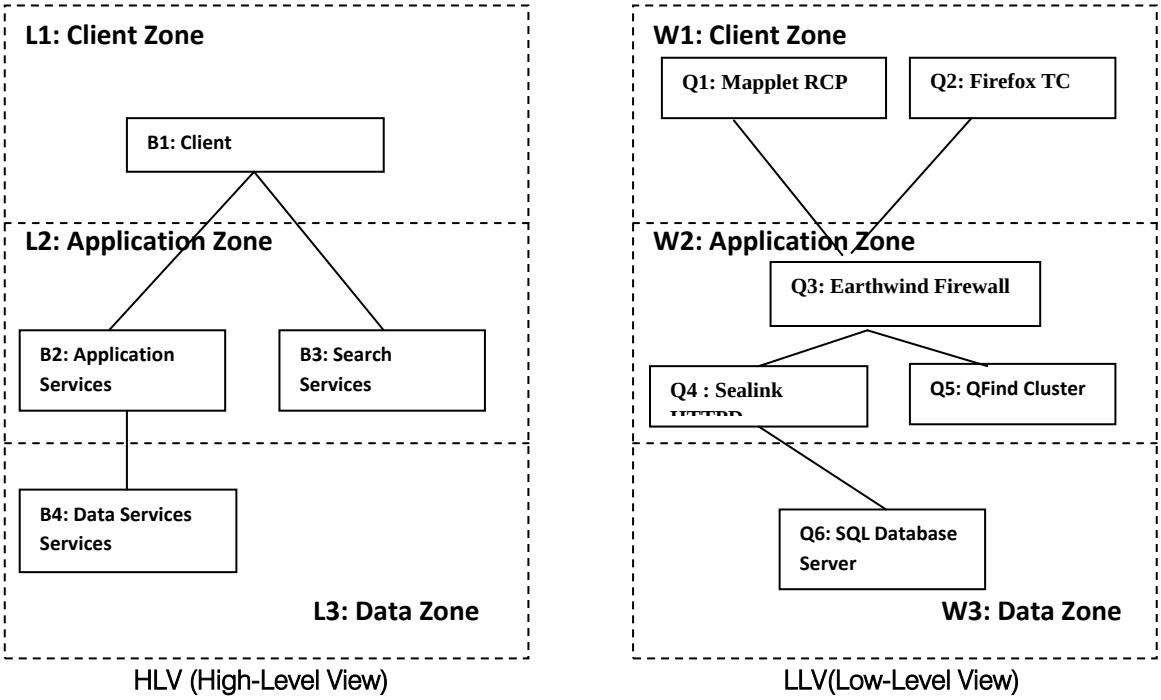

Fig. 2: Simple Example Diagram Pair

- Evidence takes the form of having similar or dissimilar features. For example, if two nodes have the same label, this is strong evidence that they match. If two nodes are at totally different positions in their respective diagrams, that is evidence that they do not match.
- For a node pair (n, n') sometimes there is some evidence that n and n' match and other evidence that n and n' do not match. Practitioners will use their experience to weigh the relative significance of the different pieces of evidence and decide whether or not n and n' match.
- The correspondence can be filled in by identifying one-to-one matches using evidence about node pairs. Other kinds of evidence help suggest non-one-to-one matches when necessary. For example, if diagram D has a node n labeled "Firewall and Access Control" and D' has node n'1 labeled "Firewall" and n'2 labeled "Access Control," the labels suggest that n matches to both n'1 and n'2. If n'1 and n'2 are both within the same container, this is further evidence that they may match to the same node in D.

IV. Solution Overview

An overview of our solution, and it serves as a road map to Bayesian correspondence, which gives the mathematical and gives a mathematical description an algorithm.

Our algorithm as Automated Matching of Models (AMMO). We explain the main ideas of the AMMO algorithm by tracing its behavior on a simple example diagram pair, HLV and LLV, as shown in Fig. 2. This diagram pair is highly simplified for presentation purposes, but it does exhibit some of the difficulties found in production models, such as non-obvious node matches and matches that are not one-to-one

The tags B1; B2; . . . ; Q1; Q2; . . .; L1; L2; . . . ; W1; W2; . . . are only for ease of reference in this discussion and are not part of the actual node labels. Also, note that regions, such as "L2: Application Zone," contain nodes, such as "B2: Application Services" and

"B3: Search Services." Further note that the label of a node is not qualified by the label of the region that contains it.

a) Feature Similarity

Our algorithm begins by computing a number of similarity values for each possible node pair consisting of a node from one diagram and a node from the other diagram, i.e., $(x, x') \in HLV \times LLV$. A similarity value is computed for each feature from a predetermined set of features. For example, nodes with similar labels often match, so one of the features we work with is the textual label of a node, and one of the similarities we compute for a node pair is its label similarity—a value between 0 and 1 reflecting the string similarity between the node labels. A similarity value can be regarded as a "raw similarity score" for a particular feature for a node pair.

Table 1: Pair-wise Label Similarity for Fig. 2

	Q1	Q2	Q3	Q4	Q5	Q6
B1	0.118	0.125	0.083	0.211	0.316	0.095
B2	0.385	0.240	0.061	0.143	0.143	0.200
B3	0.190	0.200	0.286	0.348	0.174	0.240
B4	0.316	0.111	0.231	0.095	0.095	0.435

b) Match Probability from Feature Similarity

A similarity value in itself does not indicate whether pair of nodes match; that is, it is unclear whether a particular similarity is low or high with respect to the population. To transform a raw score consisting of a feature similarity value into a probability that a pair of nodes match. Given a probability distribution of the similarity values, based on similarities observed for matching and non-matching pairs in training data, Bayesian inference will convert the similarity of (x, x') into the probability that (x, x') match. From Table 1 to Table 2 the probabilities resulting from Bayesian inference given the similarities. One can see that the probability of node B4 matching to Q6 is much higher than the probability of B4 matching to any other node. One can also see that B2 is approximately twice as likely to match to Q1 as it is to match to any other node. Finally, one can see that the probabilities of B1 and B3 matching to any of the nodes in the second diagram are approximately equal, indicating that the label feature is inadequate in determining matches for these nodes.

Table 2 : Pairwise Match probabilities based on Label Similarity

	Q1	Q2	Q3	Q4	Q5	Q6
B1	0.100	0.100	0.104	0.108	0.155	0.102
B2	**0.225**	0.116	0.108	0.108	0.100	0.105
B3	0.104	0.105	0.135	0.182	0.102	0.116
B4	0.155	0.101	0.113	0.102	0.102	**0.308**

c) Multiple Evidencer

For some nodes, such as B1, label similarity does not help much in finding a match. In general, one evidencer is not usually enough to find the best match for a node. Thus, AMMO algorithm employs several evidencers. For example, it is noted previously that B2 appeared to correspond to Q1 based on label probabilities. However, a human expert would know intuitively that B2 should correspond to Q4, because both appear to be in similar positions in the two diagrams. For multiple evidencers need a mechanism for combining one kind of evidence with another. AMMO combines evidence using Bayesian inference on a joint probability distribution over all of the kinds of evidence. The results of combining the label and position evidence. Note that B2 now matches to Q4 with probability five times greater than any other node. Note as well that the possibilities concerning matches for other nodes have been narrowed down considerably.

Table 3 : Probabilities based on Position similarity

	Q1	Q2	Q3	Q4	Q5	Q6
B1	**0.728**	**0.844**	**0.255**	0.003	0.009	0.000
B2	0.010	0.001	**0.313**	0.913	0.022	**0.407**
B3	0.002	0.015	**0.275**	0.022	0.917	**0.238**
B4	0.000	0.000	0.087	**0.659**	0.033	**0.741**

d) Simple Evidencer and Complex Evidencer

The evidencers combining obtained by both the label and position evidencers yielded a very probable match for B2. Beyond this, there is additional evidence that makes this match even more probable. B4, which is a neighbor of B2, matches Q6, which is a neighbor of Q4—having matching neighbors is additional evidence that B2 matches Q4. Our implementation includes a "connection evidencer" that provides such evidence. Evidencers such as the label or position evidencers simple evidencers because they use only information about the given pair of nodes. In contrast, call evidencers like the connection evidencer complex evidencers because they use more than just information about a given pair of nodes to compute the similarity for that pair of nodes—they also use information about other pairs of nodes (in this case: neighboring nodes) that have already been determined to match.

Table 4 : Probabilities based on Both Label and Position Similarity

	Q1	Q2	Q3	Q4	Q5	Q6
B1	**0.230**	**0.375**	0.038	0.000	0.002	0.000
B2	0.003	0.000	0.053	**0.561**	0.003	0.075
B3	0.000	0.002	0.056	0.005	**0.555**	0.039
B4	0.000	0.000	0.012	0.181	0.00	**0.560**

e) Splits and Merges

HLV has four nodes and LLV has six, clearly not every node of LLV can participate in a one-to-one match. It is possible that a node from one diagram

matches no nodes from the other diagram. Another possibility is that a node from one diagram matches a combination of nodes from the other diagram. Splits (one-to-many matches) and merges (many-to-one matches) are common in practice. Experts identify splits and merges by combining several pieces of evidence.

For example, an expert might note the following characteristics of HLV and LLV:

- C1 is close in position to each of P1, P2, and P3.
- P1, P2, and P3 are interconnected.
- The combination of P1, P2, and P3 taken together has connections to P4 and to P5, and these connections appear to match the connections from C1 to C2 and to C3.

These characteristics, when taken together, indicate that C1 is likely to have split into P1, P2, and P3, i.e., that C1 matches P1, P2, and P3.

f) Drops

It is also possible that a node in one diagram does not match any node in the other diagram. The probability that a node is dropped as the drop probability, denoted as P_DROP. This probability is determined empirically based on training data.

g) Correspondence Score

The entire correspondence between the two diagrams from individuals matches between nodes. A Naïve approach to find "best" correspondence between two diagrams would be to include the node pairs with the highest pair probabilities. Table 5 below shows the results of combining all evidence about pairs for above example.

Table 3.6 : Pair-wise Match Probabilities based on all evidence

	P1	P2	P3	P4	P5	P6
C1	**104.4**	**189.6**	5.371	0.000	0.001	0.000
C2	0.415	0.019	30.82	**787.0**	2.787	0.037
C3	0.082	0.642	79.93	5.572	**783.0**	0.019
C4	0.000	0.000	1.021	0.100	0.002	**786.3**

If only to consider the pair probabilities shown in Table 5 determine the "best" correspondence to be Corr1 = {(B1, Q2), (B2, Q4), (B3, Q5), (B4, Q6)}. However, this approach fails to yield the optimal correspondence for several reasons. First, although it might result in dropped nodes (when none of the chosen pairs involve a given node), it does not take into consideration the probability of those drops. For example, correspondence Corr1 does not include a match for Q1, and thus, Q1 is a dropped node (as we have defined that above). However, if the probability of a node being dropped is extremely low, it might have been better for Corr1 to include a split (as that was defined above) involving Q1, resulting in a correspondence which is more likely overall. Second, although it might result in splits and merges (when more than one of the chosen pairs involves a given node), this approach does not take into account the probability of these splits and merges. Third, greedily choosing the best pairs, one after the other, does not take into account the fact that choosing a particular pair match can raise or lower the probability of other pair matches, due to complex evidencers such as the connection evidencer.

h) Complexity

A correspondence using only simple node pair evidencers such as label and position, and restrict ourselves to correspondences in which all node matches are one-to-one, then need to find the maximum score correspondence using a polynomial-time algorithm based on maximum-weight bipartite matching. Using complex evidencers and allowing correspondences that are not one-to-one, the problem of identifying the maximum score correspondence is NP-hard.

V. Bayesian Correspondence Model

a) Correspondences and Matches

Let E and E' be diagrams whose nodes are sets N and N', respectively. Our core notion is the diagram correspondence, which equates sets of nodes in N with sets of nodes in N', but also allows nodes to be left out. Formally, Q is a $partial\ partition$ of a set U iff $P = \{a1, a2, ..\}$, where each $a_i \subseteq U$ and $a_i \cap a_j = \emptyset$; for all $i \neq j$. A diagram correspondence for nodes N and N' of two diagrams is a tuple $C = (S, S', f)$, where

S is a partial partition of N;
S' is a partial partition of N';
$f : S \rightarrow S'$ is one to one:

b) Evidencers

Evidencers provide the basis for determining the probability that a pair of nodes match, based on one kind of evidence. Informally, an evidencer consists of three parts: 1) a definition of a node feature (e.g., a node's label), 2) a function that measures the similarity of two nodes based on that feature, and 3) a probability distribution of node pair similarity values in cases where the two nodes match, and a probability distribution of node pair similarity values in cases where the two nodes do not match.

Formally, an evidencer consists of a similarity function e_i and probability functions a_i and b_i.

The similarity function is a function$e_i(x, x')$, where (x, x') is a node pair from (E, E'), where E' is a diagram derived from E by an unspecified procedure $\mathcal{D}$. We model $\mathcal{D}$ by asserting that $e_i(x, x')$ is a random variable. The range of e_i is arbitrary: The set of values used to measure similarity can be chosen to suit the evidencer. For example, the label evidence similarity function $e_l(x, x') = textsim(label(x), label(x))$ returns a real number in the interval $[0,1]$ ($textsim$ is a function

that returns a similarity value for two strings: our prototype used a function implemented in the Python standard libraries).

c) Correspondence Probability

In order to use the evidence to find the best correspondence, model the best correspondence as a random variable c that can take any diagram correspondence as its value. Estimation of the best correspondence is the one that has the highest probability given in the evidence.

$$\hat{c} = \arg\max_{c} P(c|e).$$

d) Singular Correspondence Probability Model

The singular correspondence probability model defines the probability of a singular correspondence conditional on the observed evidence.

Let (S, S', f) be a singular correspondence for diagrams containing nodes n and n0. We will use $\mathcal{N}(S)$ to refer to the set of nodes in the partial partition S.We use the notation (x, ϕ) to mean that the node x in the first diagram does not match any node in the second diagram, and similarly for (ϕ, x'). Then,

$$\begin{aligned} pairs(c) \equiv & \{(x, f(x))|x \in \mathcal{N}(S)\} \\ & \cup \{(x, \phi)|x \in N \backslash \mathcal{N}(S)\} \\ & \cup \{(\phi, x')|x' \in N' \backslash \mathcal{N}(S)\} \end{aligned}$$

Conditional independence allows us to define the correspondence probability as the product of the probability of the pairs:

$$P(c|e) = \prod_{(x,x') \in pairs\ (c)} P(\langle x, x' \rangle | e(x, x'))$$

One-to-none match probability. We assume simply that a node maps to nothing with fixed probability $P(\langle x, \emptyset \rangle) = P(\langle \emptyset, x \rangle) = y_0$. Choose the numerical value of y_0 based on the empirical frequency of one-to-none pairs observed in training data. It may improve accuracy to develop a model of the probability that n maps to nothing based on the features of x. However, in this paper have not implemented such models.

One-to-one match probability model. By adopting a Bayesian model of the probability that one node matches another conditional on the evidence:

$$P(\langle x, x' \rangle | e(x, x')) = \frac{P(\langle x, x' \rangle) P(e(x, x') | \langle x, x' \rangle)}{P(e(x, x'))}$$

Because $\langle x, x' \rangle$ and $\langle x, x' \rangle$ are mutually exclusive events and exhaustive of the space of all possible outcomes with respect to (x, x'), the denominator can be rewritten using a standard normalization technique to get:

$$P(\langle x, x' \rangle | e(x, x')) = \frac{P(\langle x, x' \rangle) P(e(x, x') | \langle x, x' \rangle)}{P(\langle x, x' \rangle) \cdot P(e(x, x') | \langle x, x' \rangle) + P(\langle x, x' \rangle) \cdot P(e(x, x') | \langle x, x' \rangle)}$$

Assuming that e_i is independent of e_j for all $i \neq j$,

$$P(e(x, x') | \langle x, x' \rangle) = \prod_i P(e_i(x, x') | \langle x, x' \rangle)$$

(and similarly for $\langle x, x' \rangle$), so rewrite once more to get:

$$P(\langle x, x' \rangle | e(x, x')) = \frac{p(1)}{p(1) + p(0)},$$

Where

$$p(1) = P(\langle x, x' \rangle) \prod_i P(e_i(x, x') | \langle x, x' \rangle)$$

$$p(0) = P(\langle x, x' \rangle) \prod_i P(e_i(x, x') | \langle x, x' \rangle)$$

The factors $P(e_i(x, x') | \langle x, x' \rangle)$ and $P(e_i(x, x') | \langle x, x' \rangle)$ are the values that are computed by the probability functions a_i and b_i defined earlier for evidencers.

The factor $P(\langle x, x' \rangle)$ is referred to as a prior. a_i and b_i the prior by decomposing the match event into simpler events, and then, applying commonly used principles of prior selection. First, In this paper notice that the event $\langle x, x' \rangle$ decomposes into two events: E, the event that x matches to some node (i.e., x is not dropped), and F, the event that x matches specifically to x'. Thus, $P(\langle x, x' \rangle) = P(E)P(F|E)$. For $P(E)$, we use a simple empirical prior: $P(E) \equiv 1 - y_0$, where y_0 is the Probability that a node is dropped, as observed in training. For $P(F|E)$, we use an indifference prior: Knowing only that x matches to some node in N', we assume that all nodes are equally likely, so $P(F|E) = 1/|N'|$. This gives us our complete prior: $P(\langle x, x' \rangle) = (1 - y_0)/|N'|$.

e) Split-Merge Correspondence Probability Model

The split-merge correspondence probability model is like the singular correspondence probability model, except that paper deal with pairs of sets of nodes rather than pairs of individual nodes decompose a split-merge correspondence $c = (S, S', f)$ into set pairs as follows:

$$\begin{aligned} spairs(c) \equiv & \{(s, f(s))|s \in S\} \\ & \cup \{(\{x\}, \emptyset)|x \in N \backslash \mathcal{N}(S)\} \\ & \cup \{(\emptyset, \{x'\})|x' \in N' \backslash \mathcal{N}(S')\}, \end{aligned}$$

One-to-many match probability model. For the one-to-many case can use a Bayesian model similar to that for the one-to-one case:

$$P(\langle s, s' \rangle | e(s, s')) = \frac{P(\langle s, s' \rangle) P(e(s, s') | \langle s, s' \rangle)}{P(e(s, s'))},$$

and proceed similarly to the one-to-one case, ultimately arriving at the need to compute factors $P(e_w(s, s') | \langle s, s' \rangle)$ and $P(e_w(s, s') | \langle s, s' \rangle)$, where the e_w are similar to the e_i of the one-to-one case, except that

they deal with sets rather than individual nodes. As well, this need to compute a prior $P(\langle s, s' \rangle)$.

Several issues in computing the factor $P(e_w(x, \{x'_1, x'_2\})|\langle x, \{x'_1, x'_2\}\rangle)$, which is the probability according to one kind of evidence (e_w) that the node x matches the set consisting of nodes x'_1 and x'_2, i.e., that "x splits into x'_1 and x'_2", or conversely that "x'_1 and x'_2 merge into x."

One way that we address these two issues is to define a new kind of evidence based on the evidence about the merge node matching each of the split nodes individually. That is, consider evidence about pairs of nodes, each pair consisting of the merge node and one of the split nodes.

It define

$$P(e_w(x, \{x'_1, x'_2, \dots, x'_k\})|\langle x, \{x'_1, x'_2, \dots, x'_k\}\rangle) \equiv P(e_i(x, x'_j)|\langle x, x'_j\rangle),$$

Where

$$j = arg\ \min_{l=1\dots k}(e_i(x, x'_l)),$$

This define the prior for the one-to-many case as follows: We notice that the event $\langle x, \{x'_1, \dots, x'_k\}\rangle$ decomposes into two events: G, the event that x matches a set of k nodes, and the event H that n matches specifically to $\{x'_1, \dots, x'_k\}$. Thus, $P(\langle x, \{x'_1, \dots, x'_k\}\rangle) = P(G)|P(H|G)$. For $P(G)$, we use the fixed empirical prior, m_k, the observed probability that a node x will match exactly k nodes. For $P(H|G)$, we use an indifference prior: Knowing only that n matches to a set of k nodes in N', we assume that any of the k nodes is equally likely. This yield:

$$P(\langle x, \{x'_1, \dots, x'_k\}\rangle) = \frac{y_k}{\binom{|N'|}{k}}.$$

f) The Maximization Problem

The previous sections showed how to compute $P(c|e)$ for a given correspondence c and evidence e. To complete the algorithm, one should describe how to find the c with maximal $P(c|e)$.

Computing the score of such correspondences using only simple evidencers can be done in polynomial time (ideally constant time per node pair, quadratic overall). To find the maximum probability correspondence in this case, construct a graph which has as its nodes the union of the nodes in the two diagrams, $N \cup N'$. Place an edge from every node n in N to every node n' in N' with edge weight $w(x, x') = P(\langle x, x'\rangle|e(x, x'))$. Now find the maximum probability correspondence in polynomial time using maximum-weight bipartite matching [4].

i. *Greedy Search*

The simplest search algorithm is greedy search. In greedy search, we keep track of only one piece of information, the current state. On each step, we examine all states reachable by a single transition from the current state, and move to the state with the greatest probability. And there is no backtracking—In this paper, only consider transitions that add a node pair to the correspondence, not those that remove a pair. If there is no next state with greater probability than the current state, the search stops.

```
GreedySearch:

BestCorr  : = emptyCorrespondence
/* Initialize the best correspondence to one  in which no node
has a corresponding match in the order diagram */
BestScore: = Score (BestCorr)
FoundBetter: = True
While foundBetter do
        FoundBetter: = False
        BestFoundSoFar: = BestCorr
        BestScoreSoFar: = BestScore
        for each pair <n,n'> that can be added to BestCorr
do
          newCorr  :=  addPairToCorr(BestCorr,<n,n'>)
          newScore  :=  Score(newCorr)
          if newScore > bestScoreSoFar then
                BestFoundSoFar: = newCorr
                BestScoreSoFar: = newScore
                FoundBetter  : = TRUE
          end if
        end for
        if foundBetter then
           BestCorr: = bestFoundSoFar
           BestScore  : = bestScoreSoFar
        end if
end while
return (BestCorr, BestScore)
end GreedySearch
```

Fig. 3 : Greedy Search

Fig. 3 gives a high-level description of the greedy search algorithm for our problem. We assume that, before this algorithm is called, for any nodes n and n' in the two diagrams, we have already computed $p(\langle n, n'\rangle|e(n, n'))$, the probability that they match, based upon the various simple evidencers.

ii. *Complexity Analysis*

Let's assume that the total number of nodes in the diagrams is $O(N)$. Then, the naive implementation of the greedy search algorithm has complexity $O(N^4)$. The outer while loop will be executed at most $O(N)$ times since each iteration removes at least one node of the diagrams from future consideration. The for loop is executed $O(N^2)$, as there are at most N^2 pairs to consider. the naïve implementation, computing Score(newCorr) at line "*" costs $O(N)$ time due to the connection evidencer, which requires $P(\langle y, y'\rangle|e(y, y'))$ to be recomputed for each pair hm;m0i in the correspondence. The connection evidencer will return different values for the probability of $\langle y, y'\rangle$. Hence, the total complexity of the algorithm is $O(N^4)$.

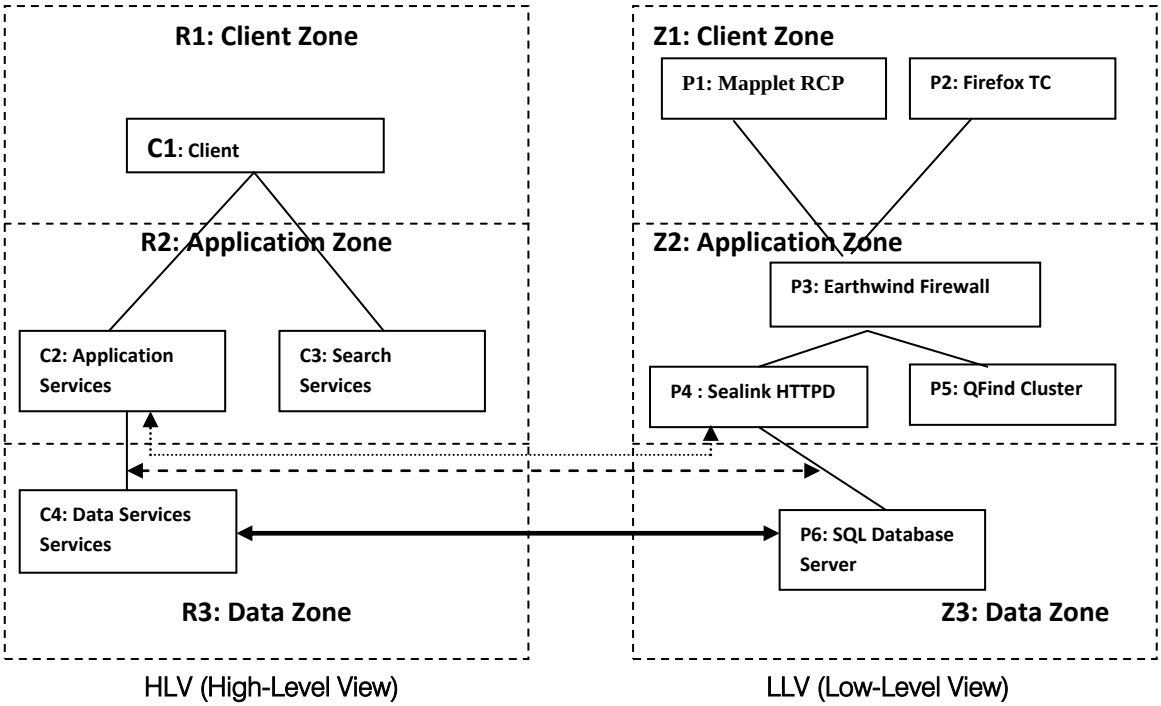

Fig. 4 : Connection matching on the example

iii. *Incremental Algorithm*

It is possible to implement the Score() function so that it takes time proportional to the number of neighbors of the added nodes—probability is only recomputed for pairs that might possibly be affected by a newly added pair. Assuming bounded degree graphs, this *incremental* version takes complexity $O(N^3)$.

VI. Prototype Evidencer

It Describes the set of evidencers that was designed and implemented as part of prototype implementation of the AMMO algorithm. The prototype evidencers can calculate

a) Simple Evidencer

Label Evidencer measures the similarity between text labels of a node pair. Python standard library function difflib.Sequence- Matcher.ratio().

Region Evidencer, A region may have a name, a set of neighboring regions, and a set of nodes that are located within it.

Type Evidencer. Some diagrams have nodes typed as being hardware components or infrastructure software components or application software components (or EJBs or ManagedComponents), while other diagrams have nodes typed as being actors or information flows or use cases or systems.

Position Evidencer Similarity values returned by position evidencer and expect the euclidean distance between matching nodes to be small.

b) Complex Evidencer

A complex evidencer to be an evidence which requires information from more than just the node pair for which it is finding a similarity value. In addition to that node pair, it also takes as input a partial correspondence between the two diagrams.

Connection evidencer. The Connection evidencer is based on the connections, or edges, that each node has to its immediate neighbors.

Fig.3 illustrates connection similarity computation for the pair (B2, Q4) in our sample diagram pair. In this figure, the solid curved line indicates that at this point in the search, the match $\langle B4, Q6\rangle$ is already part of the correspondence. The dotted curved line indicates that we are considering the node pair (B2, Q4). By virtue of the facts that B2 has two neighbors (B1 and B4), Q4 has two neighbors (Q3 and Q6), and one of B2's two neighbors (B4) matches one of Q4's two neighbors (Q6), as indicated by the dashed line, the connection similarity for (B2, Q4) is $avg\left(\frac{1}{2},\frac{1}{2}\right) = 0.5$. Ultimately, connections turn out to be strong evidence that B2 and Q4 match.

c) Split Evidencer

A Split-Merge Model which defined the probability of a split-merge correspondence conditional on the observed evidence. Recall that a split-merge correspondence is one containing split-merge matches—matches between one node and a set of nodes. Further, recall that, to evaluate the probability of such correspondences, two types of evidencers are used: simple (pair) evidencers and split evidencers. The simple evidencers that were implemented as part of our prototype, and this section describes the split evidencers of our prototype.

The motivation for creating special-purpose split evidencers arose out of the observation that split-merge correspondences exhibited different characteristics than singular correspondences and that these characteristics were not taken into account by the simple evidencers.

Label Sim evidencer. The similarity determined by the Label Sim evidencer is the minimum similarity among the labels of the nodes.

Label Intersect evidencer. The similarity determined by the Label Intersect evidencer is the similarity between the label of x and the longest suffix or prefix commonto the labels of the x_i' nodes.

Label Concat evidencer. The Label Concat Evidencer similarity function uses the Label Evidencer similarity function to obtain the similarity between the label of x and the concatenation of the labels of the x_i' nodes.

Inner Connect evidencer. This is a discrete measure of similarity based on whether or not all of the x_i' nodes are connected to each other.

Outer Connect evidencer. This is a continuous measure of connection similarity between x and the cluster of x_i' nodes taken as a whole.

VII. Ammo-Lite: Improving Performance and Scalability

Although the greedy search algorithm described performed well for diagrams with dozens of nodes, it was not practical for diagrams with hundreds of nodes. the major scalability problem with AMMO is that every time it has to decide which node pair to add next, it must compute an exact probability for each possible correspondence that would result from adding one more node pair. Our incremental version of greedy search helps avoid some of this recomputation, but not enough to be practical for larger-scale diagrams. To solve this problem, we designed a new algorithm, AMMO-LITE, which approximates AMMO's behavior but uses a simpler search that is driven by pair probabilities rather than correspondence probabilities. This approach avoids repeated calculation of correspondence probabilities and, in practice, achieves much better performance with only a small loss of precision.

```
AMMO-LITE
Use simple evidencers to precompute and store
probabilities of all pairs < n,  m >
PotentialPairs := list of all pairs < n,  m > in descending
order of      probability
Corr := emptyCorrespondence
Done = False
While PotentialPairs is not Empty and
      Prob(first(PotentialPairs)) > threshold do
     <n,  m> :=  removeFirst (PotentialPairs)
     if <n, m> can be added to Corr then
           must_re_sort := False
           Corr := addPairToCorr(Corr, <n,  m>)
           for each pair <<nn, mm> in PotentialPairs do
              if nn == n or mm == m then
  use split evs to update the probability of <nn, mm>
                 must_re_sort  : =  True
              else if nn is neighbor of n and
                 mm is a neighbor of m then
  use connect ev to update the probability of <<nn,mm>
                 must_re_sort  : =  True
              end if
           end for
           if must_re_sort then
              PotentialPairs  : = re-sort(PotentialPairs)
           end if
     end if
end while
```

Fig. 5: Ammo-Lite

a) Algorithm Description

It uses the probabilities of the pairs to determine the order in which pairs should be added to the correspondence. This is done as follows:

As in the case of AMMO, the first thing that the algorithm does is to precompute probabilities of all possible node pairs, using the simple evidencers. It then creates a sorted list Potential Pairs, which contains the node pairs sorted in descending order by probability.

The main loop of AMMO-LITE goes through Potential-Pairs, adding the highest probability pair (the one at the head of the list) to the correspondence, provided that it is permissible to add that pair. It is not permissible to add a pair. It is not permissible to add a

pair to the correspondence if that would result in a many-to-many match. Each time a new pair $\langle x, y\rangle$ is added to the correspondence, the algorithm goes through the list again, in order to determine if the precomputed probability of any remaining pair $\langle xx, yy\rangle$ has been affected. The probability of pair $\langle xx, yy\rangle$ in PotentialPairs will be affected in two different circumstances:

- If xx or yy is one of the nodes in the pair we just added, then adding $\langle xx, yy\rangle$ would result in a split/merge. Thus, we change the precomputed probability stored for $\langle xx, yy\rangle$ to be the probability of the split/merge that would result from adding $\langle xx, yy\rangle$ to the correspondence.
- If xx and yy are neighbors of x and y, respectively, then adding $\langle x, y\rangle$ to the correspondence will affect the connectivity similarity of $\langle xx, yy\rangle$. Thus, $P(\langle xx, yy\rangle)$ must be recomputed, this time using the connection evidencer as well as the simple evidencers.

After going through PotentialPairs, if any probabilities have been recomputed, the list is resorted. The algorithm then continues with another iteration of the main loop to add another pair to the correspondence. The algorithm terminates when either the list is empty or the probability of the pair at the head of the list is less than some threshold value. This value is determined by experimentation with training data, and can be easily changed. In our implementation, this threshold is m_0, the empirically determined probability that a node does not correspond to any node in the other diagram.

b) Complexity Analysis

Let the total number of nodes in a diagram be $O(N)$, as in the analysis of AMMO. Depending on the value of threshold, the outer while loop could be executed $O(N^2)$ times, once for every possible node pair. However, the outer if statement (immediately within the while loop) will only be true $O(N)$ times since each pair added must add at least one new node to the correspondence, due to the many-to-many restriction, and hence, add at most $O(N)$ pairs. Thus, the nested for loop will be reached on only $O(N)$ iterations of the while loop. Each time the for loop is reached, it will execute $O(|PotentialPairs|) = O(N^2)$ iterations. The resulting total complexity is $O(N^3)$. Similarly, like the nested for loop, the statement resort(PotentialPairs) will be reached at most $O(N)$ times. Sorting being $O(NlogN)$, each sort of the $O(N^2)$ items in PotentialPairs will have complexity $O(N^2logN)$. Thus, the resulting total complexity of the algorithm due to all sorting is $O(N^3logN)$. That dominates the $O(N^3)$ of the nested for loop, and therefore, the overall worst-case total complexity of the AMMO-LITE algorithm is $O(N^3logN)$.

Table 7: Experiment Results: Average Algorithm Recall, Precision and Runtime (in Seconds)

Algorithm	Recall %	Precision %	Time
Baseline (non-Bayesian)	75	70	3
AMMO (all evidencers)	82	85	82
AMMO-LITE (all evidencers)	80	84	3

To see why AMMO-LITE performs better than AMMO in practice, consider the following: In AMMO-LITE, each timewe add a pair $\langle x, y\rangle$ and make a pass through the list PotentialPairs. Although this list can be $O(N^2)$, it is a "quick" pass over the list—most of the pairs are just skipped. "Real" computation only takes place if $\langle xx, yy\rangle$ meets certain criteria in which case, we recompute its associated probability. So, in practice, our performance is better than $O(N^3logN)$ would suggest.

In fact, employing a priority queue along with an incremental approach to updating pair probabilities, and assuming a bounded-degree graph, we could achieve an overall total complexity of $O(N^2logN)$ as follows: This can implement PotentialPairs as a priority queue in which pairs are ordered according to their probability, there by obviating the need for separate explicit sorts. Initially, we construct PotentialPairs by inserting all of the $O(N^2)$ pairs into it. With a priority queue implementation for which insert, get_max, and delete are $O(logN)$, the complexity of constructing PotentialPairs is $O(N^2logN)$. In that way, we avoid having to reexamine all of the $O(N^2)$ remaining pairs in PotentialPairs. Assuming bounded-degree graphs, with the number of neighbors of a pair $\langle x, y\rangle$ being bounded by a constant k, the number of pairs whose probability must be recomputed due to connectivity is k. Whenever we recomputed the probability of a pair and delete it from Potential-Pairs and reinsert it with its new probability (or we could simply do a change_priority operation). With delete and insert being $O(logN)$, the total complexity due to recomputing probabilities of neighbors of all of the $O(N)$ added pairs is $O(N * k * logN) = O(NlogN)$. Similarly, when a pair $\langle x, y\rangle$ is added to the correspondence, the number of pairs $\langle xx.yy\rangle$ whose probability must be recomputed because they would now result in splits or merges is at most $O(N)$ because there are at most $O(N)$ pairs for which $x = xx$ or $y = yy$. Hence, the total complexity due to recomputing probabilities due to split/merge considerations is $O(N * N * logN) = O(N^2logN)$. Thus, the overall total complexity of the algorithm would be $O(N^2logN)$.

c) AMMO-LITE Experiments

Table 7 compares the results of running AMMO-LITE against those obtained by running AMMO and

Baseline. The results include accuracy as well as the runtime (in seconds) required by each algorithm.

Table 8: Experimental Results: AMMO-LITE versus AMMO Runtimes (in seconds) for Various Diagram Sizes

Pair	Nodes	Edges	AMMO-LITE	AMMO	Ratio
Pair 1	9	13	0.83	2.74	3.3
Pair 2	12	12	0.96	3.68	3.8
Pair 3	15	24	2.63	12.19	4.6
Pair 4	22	41	5.11	41.49	8.1
Pair 5	35	31	11.30	98.66	8.7
Pair 6	41	68	15.51	257.27	16.6
Pair 7	637	968	6175.00	-	-

The values in the table were obtained by averaging the Recall, Precision, and Time metrics for each algorithm across all of our model pairs.

The AMMO-LITE algorithm did not do quite as well as AMMO in terms of both Recall and Precision, but it still did significantly better than the non-Bayesian approach. Furthermore, if one examines the cases where AMMO-LITE did poorly in comparison to AMMO, most of these cases involved complex correspondences with a number of challenging matches and multiple split/merges.

VIII. Results

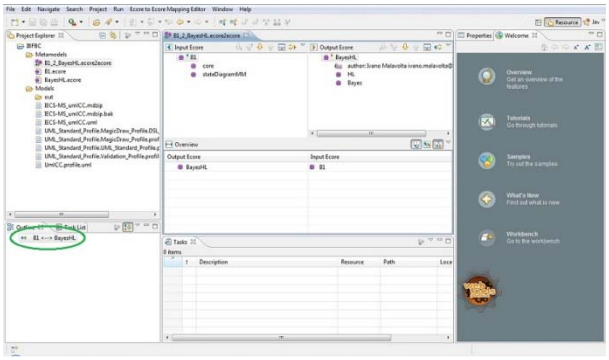

Fig. 6 : (a). Mapping between two ADL Models

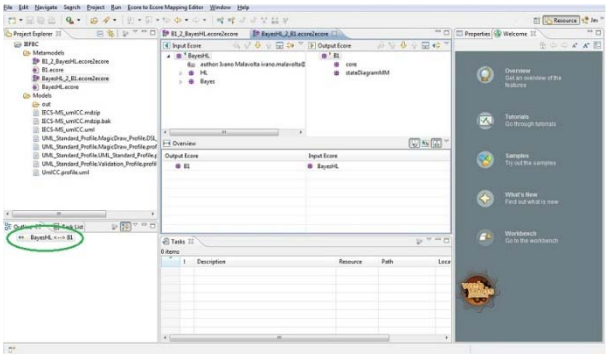

Fig. 6 : (b). Mapping between two ADL Models

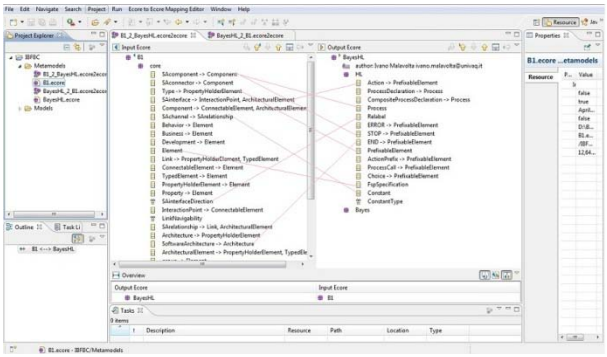

Fig. 7 : (a). Mapping the objects between two ADL Models

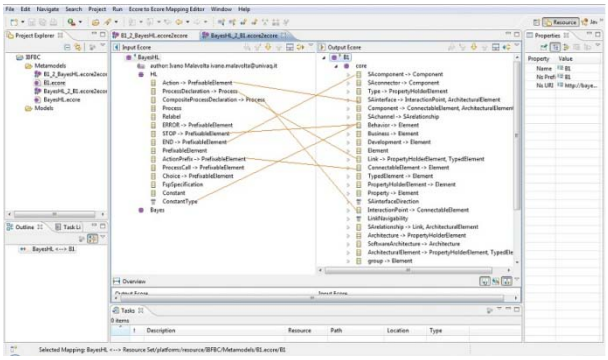

Fig. 7 : (b). Mapping the objects between two ADL Models

IX. Conclusion

We have identified and described the model correspondence problem, an important problem in software engineering. We have designed a Bayesian framework that supports the reasoning needed to solve the model correspondence problem. And we have implemented and tested a matching algorithm based on our framework, finding that it achieved high accuracy on a set of test diagram pairs. We believe that this work holds great promise for the future.

References Références Referencias

1. Garland and R. Anthony, Large-Scale Software Architecture. John Wiley and Sons, 2003.
2. R. Youngs, D. Redmond-Pyle, P. Spaas, and E. Kahan, "A Standard for Architecture Description," IBM Systems J., vol. 38, no. 1, pp. 32-50, 1999.
3. Z. Xing and E. Stroulia, "Umldiff: An Algorithm for Object-Oriented Design Differencing," Proc. 20th IEEE/ACM Int'l Conf. Automated Software Eng., pp. 54-65, 2005.
4. D.E. Tarjan, Data Structures and Network Algorithms. SIAM, Nov. 1983.
5. B. Ramesh and M. Jarke, "Towards a Reference Model for Requirements Traceability," IEEE Trans. Software Eng., vol. 27, no. 1, pp. 58-93, Jan. 2001.
6. Jossic, M.D.D. Fabro, J.-P. Lerat, J. Bezivin, and F. Jouault, "Model Integration with Model Weaving: A

Case Study in System Architecture," Proc. Int'l Conf. Systems Eng. and Modeling, pp. 79-84, 2007.
7. P. Kruchten, "Architectural Blueprints-The 4 + 1 View Model of Software Architecture," IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995.
8. D. Mandelin, D. Kimelman, and D.M. Yellin, "A Bayesian Approach to Diagram Matching with Application to Architectural Models," Proc. 28th Int'l Conf. Software Eng., May 2006.
9. N. Rozanski and E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison- Wesley, 2005.
10. G. Antoniol, G. Canfora, G. Casazza, and A.D. Lucia, "Maintaining Traceability Links during Object-Oriented Software Evolution," Software-Practice and Experience, vol. 31, pp. 331-355, 2001.
11. Handbook on Architectures of Information Systems, pp. 669- 692. Springer, 2006.
12. IBM Insurance Application Architecture-Executive Summary,http://www.03.ibm.com/industries/insurance/us/detail/solution/P669447B27619A15.html, 2009.
13. TM Forum-Information Framework (SID). http://www.tmforum.org/InformationFramework/1684/home.html, 2009.
14. NGOSS Shared Information/Data Model, http://en.wikipedia.org/wiki/NGOSS_Shared_Information/Data_Model, 2009.
15. WebSVN-Diplomarbeit,http://surprise.wh-stuttgart.de/websvn/log.php?repname=diplomarbeit**path=%2Ftrunk%2Fdesign%2FMiddlewareUML.xmi**rev=87**sc=1**isdir=0, 2009.
16. S. Abrams, B. Bloom, P. Keyser, D. Kimelman, E. Nelson, W. Neuberger, T. Roth, I. Simmonds, S. Tang, and J. Vlissides, "Architectural Thinking and Modeling with AWB: The Architects Workbench," IBM Systems J., vol. 45, no. 3, pp. 481-500, 2006.