2012 IEEE Ninth International Conference on Services Computing

Effective Reuse via Modeling, Managing, and Searching of Business Process Assets

Nanjangud C. Narendra, Karthikeyan Ponnalagu, G.R. Gangadharan, Hong-Linh Truong, Schahram Dustdar, and Aditya K. Ghose

***Abstract*— Cost and competitive pressures are forcing business organizations to reuse assets from repositories, rather than develop them from scratch. But this has been hampered by some issues that have not been addressed so far. First, there is a lack of a mechanism for the representation of business process assets as variants and versions in repositories. Second, there is no formal means to compare between different variants and versions of an asset and determine which is the best to select for reuse. Third, there is a lack of a technique to determine the extent to which a business process asset could be customized for reuse. In this paper, we address the above research issues by presenting an integrated approach for modeling, analyzing, and searching business process assets in a repository for enhancing reuse. We demonstrate our approach on a large repository of business process assets in the insurance domain.**

Keywords. Business Process, Service, Reuse, Repositories, Assets

I. Introduction

Given increasing cost and competitive pressures, business organizations are reusing existing business process assets from repositories [1]. The emergence of service-oriented architecture (SOA) [2], with its emphasis on loose coupling and dynamic binding, is seen as a promising way to enable more effective reuse by packaging assets as reusable services accessible only via their interfaces. (By business process asset we mean any software component in the repository represented as a business process, sub-process, or even a single service.) However, to realize this vision, several research challenges need to be addressed. First, there is a lack of a model for facilitating the representation of business process assets as variants and versions in business process repositories, with a view towards maximizing their reusability. Second, there is no formal way to compare between different candidate asset variants and versions in order to determine which can be a better reuse candidate. Finally, no matching technique exists that can determine the extent to which a potential business process asset can be customized for reuse.

In this paper, we resolve these research challenges by representing a business process asset as a collection of its constituent services by extending our prior work on variation-oriented engineering principles of SOA [3] and by presenting a novel mechanism for searching assets for reuse. The salient contributions of our paper are as follows.

- An asset tree representation model that represents versions and variants of business process assets together in a repository
- A formal mechanism to compare and analyze variants and versions of an asset reusability
- A matching algorithm that matches a specified requirement against an asset and determines the extent to which the asset meets the requirement.

To the best of our knowledge, our approach is the first integrated technique for asset variant and version representation in a repository, along with an enhanced matching of existing assets against a specified business process requirement. In contrast to the extensive works on business process similarity matching [4], [5], [6], [7], [1] that primarily focus on structural aspects of business processes, the presented methodology in this work investigates the *semantic* aspects through a unique asset tree representation of business process models in terms of their constituent services.

The remainder of this paper is organized as follows. In the next section, we present a running example that will be used throughout the paper for illustration. Section III presents our asset tree representation model for storing business process variants and versions. Section IV describes our matching algorithm for assets vis-a-vis requirements. Implementation of our approach over a large repository of real-life business processes is described in Section V. We present related work in Section VI, followed by concluding remarks in Section VII.

II. Running Example

Our running example starts with the scenario of a business analyst looking for an insurance claims business process asset. The inputs to this process should be the details of the customer requesting the claim, and the details of the claim. The outputs of this process should be acceptance/rejection of the claim, along with the claim amount to be paid to the customer (the claim amount will be zero in case of rejection). The analyst searches the business process repository using the inputs and outputs as her search criteria. Let us assume that she finds two candidate assets as represented by `Pr1` and `Pr2` in Figs. 1 and 2, respectively. `Pr1` consists of three major sub-processes - (i) `Record Claim`, (ii) `Verify Claim`, and (iii) `Analyze Claim & Report`. In `Verify Claim` sub-process, let us assume that the *DetermineLiability* and *PotentialFraudCheck* services are

Thanks to Jayram Thathachar & Navjot Bhogal for their feedback & inputs.

Nanjangud C. Narendra is with IBM India Software Lab, Bangalore, India; Karthikeyan Ponnalagu is with IBM Research India, Bangalore, India; G.R. Gangadharan is with Institute for Development and Research in Banking Technology, Hyderabad, India; Hong-Linh Truong and Schahram Dustdar are with Vienna University of Technology, Vienna, Austria; Aditya K. Ghose is with University of Wollongong, Wollongong, Australia; {*narendra,karthikeyan.ponnalagu*}*@in.ibm.com*, *grgangadharan@idrbt.ac.in*, {*truong,dustdar*}*@infosys.tuwien.ac.at*, *aditya.ghose@gmail.com*

978-0-7695-4753-4/12 $26.00 © 2012 IEEE
DOI 10.1109/SCC.2012.46

IEEE computer society

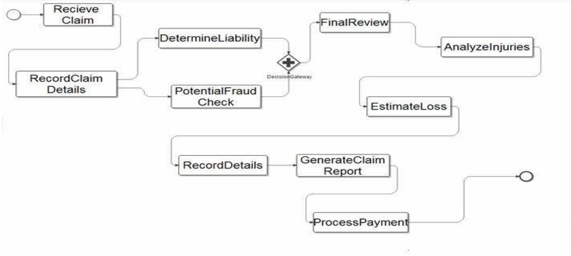


Fig. 1. Insurance Claims Process `Pr1`

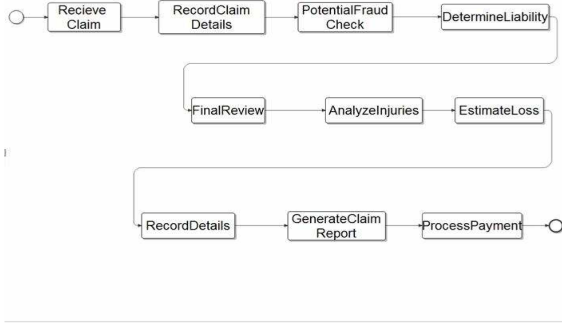


Fig. 2. Modified Insurance Claims Process `Pr2`

first executed in parallel, and then their results are sent to *ClaimInvestigation* service. A final review of the verified claim is then implemented by *FinalReview* service.

`Pr2` implements the same functionality, but differently. *DetermineLiability* and *PotentialFraudCheck* services are serialized. Then, *PotentialFraudCheck* service is modified, considering the extent of liability. Finally, a new *LiabilityPlusFraudCheck* service is added. The inputs and outputs for the services in `Pr1` and `Pr2` are depicted in Fig. 3.

The analyst can also specify specfic non-functional constraints such as performance and reliability, along with the process inputs and outputs. This could give rise to a situation where no single business process asset can match her requirement fully, thereby giving a potential trade-off, requiring user intervention to select the "best" candidate.

If the analyst wants to narrow down the search further, she would specify additional criteria, such as the services that the business process asset should contain. These would be specified in terms of their respective inputs and outputs.

III. Asset Representation in Business Process Repositories

A. Basic Definitions

Definition 1: A *business process (or sub-process)* P is defined as $P = \{S, D, C\}$, where $S = \{S_1, ..., S_n\}$ is the set of services that participate in P; $D = \{D_{ij}\}$, iff $S_i \overset{d_{ij}}{\rightarrow} S_j = true$, is the set of all data dependencies of service S_j on S_i, where $i \neq j$; and $C = \{C_{ij}\}$, iff $S_i \overset{c_{ij}}{\rightarrow} S_j = true$, where $i \neq j$, is the set of control flow dependencies between S_i and S_j, where C_{ij} is either true or false, based on whether S_i controls the execution of S_j, i.e., iff S_i precedes S_j in the control flow. ∎

If two services in a process, share the same set of required input data, but the execution of the preceding service in the process flow does not affect the execution of the succeeding service, then the services have just a data dependency relationship between them. Otherwise, they have the control flow dependency between them.

Definition 2: A *service* S_i is defined via its input and output sets respectively, i.e., $S_i = \{D_{in}, D_{out}\}$, where D_{in} is a set of input data required for invoking S_i, and D_{out} is a set of output data expected after invoking S_i. ∎

For example, the inputs for the service *DetermineLiability* in `Pr1` (see Fig. 3) are *CustomerInfo* and *ClaimInfo*, while its output is *LiabilityInfo*.

B. Metamodel-based Representation of Asset Variants

Leveraging the metamodel introduced in [3], we can separately model the static and variable parts of any software component. This metamodel consists of two parts (details are available from [3]). *variation points* are the points in the component where variations can be introduced. *variation features* refine variation points, by specifying the action semantics of the variation and its specific applicability. The same variation point can admit more than one variation feature, and one variation feature can be applied to many variation points.

This metamodel is further extended to instantiate conceptual models for modeling service-level and business process-level variations. These conceptual models can then be treated as design templates from which actual variation-oriented design can be accomplished. In our running example, an example of a variation point is a method in *DetermineLiability* service, for calculating insurance liability. A variation feature is an action to replace that method by a different method. The actual service variant is the modified *DetermineLiability* service containing the replacement method.

C. Asset Variants vis-a-vis Versions

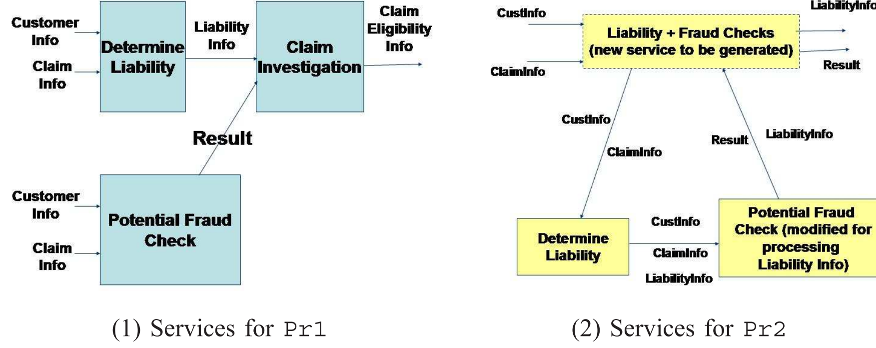
The ways in which a pair of related assets can differ from each other are *variants* and *versions*. We define them as follows.

Definition 3: A *variant* A' of an asset A is another asset such that A' can be derived from A by applying a set of variation features, each of them applied on a variation point of A:

$A' \leftarrow A + \sum_{i=1}^{k} \delta V(A)_i$such that $\delta V(A)_i \overset{VP_x}{\rightarrow} \{VF_1, VF_2, \cdots, VF_m\}$ ∎

In Definition 3, $\sum_{i=1}^{k} \delta V(A)$ comprises a varying asset of A, VP_x is a variation point, and $\{VF_1, VF_2, \cdots, VF_m\}$ are the set of variation features applied on that variation point. Referring to Fig. 3, we see that the *DetermineLiability* service of process `Pr2` is a variant of the *DetermineLiability* service of process `Pr1`; the former containing the additional input data *LiabilityInfo*.

Definition 4: A *version* A'' of an asset A is another asset such that the following holds: A'' can be derived from A through a combination of changes; each is either a change of a static part of A (as defined in Section III-B) or a change to the composition of variation model of A itself:



(1) Services for `Pr1` (2) Services for `Pr2`

Fig. 3. Services in `Pr1` & `Pr2`, and their inputs and outputs

$A'' \leftarrow A + \partial V(A)$ such that $\partial V(A) = \begin{cases} \Delta_i(A), i = 1, 2, \cdots, n, & \text{or} \\ VM(A) + VM(A'') \end{cases}$

with

$$VM(A) = \begin{array}{c} \\ VP_1 \\ VP_2 \\ \\ VP_n \end{array} \begin{array}{c} \begin{array}{cccc} VF_1 & VF_2 & \ldots & VF_m \end{array} \\ \begin{pmatrix} A1 & A2 & \ldots & Am \\ B1 & B2 & \ldots & Bm \\ & & & \\ Z1 & Z2 & \ldots & Zm \end{pmatrix} \end{array}$$

and

$$VM(A'') = \begin{array}{c} \\ VP_{n+1} \\ VP_{n+2} \\ \\ VP_{n+y} \end{array} \begin{array}{c} \begin{array}{cccc} VF_{m+1} & VF_{m+2} & \ldots & VF_{m+x} \end{array} \\ \begin{pmatrix} a1 & a2 & \ldots & am \\ b1 & b2 & \ldots & bm \\ & & & \\ z1 & z2 & \ldots & zm \end{pmatrix} \end{array}$$

▌

In Definition 4, $\partial\ V(A)$ represents the difference in A'' over A, $\Delta_i(A)$ is a change in the static part of A, and $VM(A'')$ is the change in the variation model of A itself. That is, $VM(A)$ represents the variation model of A; whereas, $VM(A'')$ represents the difference in the variation model of A'' over that of A.

The matrices $VM(A)$ and $VM(A'')$ depict the variation points in the rows and variation features in the columns. An entry in either matrix represents the existence of a variation created by applying a variation feature on a variation point - otherwise it will be a null entry.

In our running example, versions are illustrated via the two business processes `Pr1` and `Pr2` themselves. The variation models of these two processes being identical, the varying component of `Pr1` that appears in `Pr2`, is the new *LiabilityPlusFraudCheck* service added in `Pr2`.

D. Asset Tree Representation

Since most repositories are based on version control systems (e.g., CVS), we can represent the repository's assets in a version tree. Each version therefore has a unique parent, from which it has been derived. Differences between the version and its parent are typically represented manually in the form of user-editable comments. As this is an undesired practice, we introduce the novel concept of *asset tree* to represent business process assets with their variants and versions.

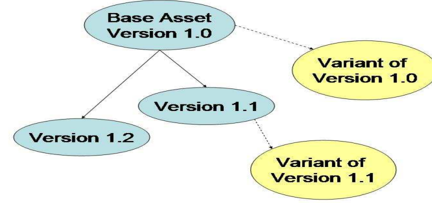


Fig. 4. Asset Tree Representation

Definition 5: An asset tree is a tree $T = (N, E)$, where N is the set of nodes, E is the set of edges, with the following properties:

- The root node of the tree is called its *base asset*
- Every node in the tree is labeled as either a version node or a variant node, and satisfies the following conditions:
  - The parent of a version node can only be a version node.
  - The parent of a variant node can only be a version node. Furthermore, variant nodes cannot have children. They are leaf nodes of the asset tree.

The asset tree also has two types of edges. A *version edge* links a version node to the version node which is its child on the asset tree, whereas a *variant edge* links a version node to one of its variants. ▌

In our design, a node in an asset tree can represent different scopes, individually or collectively: the whole process, a sub-process, or a service. This asset tree representation, illustrated in Figure 4, is an enhancement over traditional version tree representations in repositories which only represent versions.

As per Definitions 3 and 4, every variant or version of A is already represented as the asset A augmented by an enhancement of its functionality. Mapping these definitions onto our asset tree representation, we see that every asset on the asset tree (except the root) is a variant or version of its parent.

E. Asset Tree Creation and Evolution

Existing and newly-created business processes can be stored under the asset tree representation in different ways. An existing process can be transformed automatically using parsing techniques [8] or manually by the process developer to produce an asset tree. A newly-created process will be modeled using our asset tree representation.

Initially, an asset tree will have a base business process. Over time, the asset tree will be enhanced or utilized for the development of new business processes. The evolution of asset trees in the repository makes the number of asset trees as a static or a steadily growing structure. Thus, the asset tree representation helps reduce the space and time in the management and search of business processes, compared to the exponential growth for the actual number of processes that causes a serious performance issue and space concern with respect to the search and management of assets.

In the beginning, the asset tree will have an initial feature log, representing all the features available with the root node. But on the addition of every other node, the meta data of the asset tree is enhanced with the new set of features getting added for each new node (either version or variant). This also helps reduce the search time.

IV. Searching Assets

The asset tree helps optimize the search and matching of assets against user requirements. We represent asset requirements and constraints via the Asset Requirements and Constraints Model (ARCM) and asset capabilities via the Asset Capabilities and Analysis Model (ACAM). Following this, we present the algorithm for matching the appropriate asset variants/versions against the specified requirement.

A. ARCM & ACAM Models

The ARCM represents the requirements of a user looking for a business process solution or asset, along with some constraints that the user would specify. It is defined as follows.

Definition 6: An ARCM is $M_{RC} = \{R, C\}$, wherein R is a set of requirements, comprising the following:

- Input data to the required business process solution $\{d_{in_i}\}$
- Output data from the required business process solution $\{d_{out_i}\}$
- The set of services S_q that need to be part of the business process solution; each service $S_i \in S_q$ is defined via its input set $\{d_{in_i}\}$ and output set $\{d_{out_i}\}$

C is a set of constraints, with each constraint specified as per the following:

- ID, which represents the constraint's name
- Type T, which defines the unit in which the constraint is measured
- Order Ord, which could one of the following: increasing or decreasing
- Value Val, which is the minimum (resp. maximum) value of the constraint, depending on whether Ord is increasing (resp. decreasing)

▮

Please note that our ARCM model can be extended to accommodate the actual semantics of matching the types of input and output data, via preconditions and effects, e.g., as per [9]. We have omitted them in our paper, not only for reasons of simplicity, but also because they are orthogonal to the main ideas in our paper.

Our constraint representation model is inspired by QML [10], the QoS modeling language. We chose QML due to its simplicity and ease of adoption. Some well-known constraints are performance, reliability, and cost. A constraint of increasing (resp. decreasing) order implies that an asset whose non-functional property corresponds to that constraint, should have a value greater (resp. lower) than that of the constraint, in order for that particular non-functional property to be considered a match.

The ACAM model is defined as follows.

Definition 7: An ACAM is $M_{CA} = \{C_p, A\}$, wherein C_p is a set of capabilities, comprising the following:

- Input data to the asset $\{d_{in_i}\}$
- Output data from the asset $\{d_{out_i}\}$
- The set of services S_i (including its versions and variants) in the asset tree; each service $S_i \in S_q$ is defined via its input set $\{d_{in_i}\}$ and output set $\{d_{out_i}\}$

A is a set of analyses (non-functional properties), with each property specified as per the following:

- ID, which represents the property's name
- Type T, which defines the unit in which the property is measured
- Value Val, which is the value of the property

▮

It is to be noted, however, that our ACAM model is generic enough to be independent of the actual process model formalism used, e.g., whether BPMN, BPEL, etc., but is only stated in generic terms in terms of input/output data and the services that comprise the business process.

B. Matching ARCM against ACAM

Matching an ARCM against an ACAM involves functional matching followed by non-functional matching.

1) Functional Matching: Functional matching of an ARCM against the ACAMs is done as explained below (see Algorithm 1 for a formal description). Our matching algorithm combines both version feature matching and variant matching. The former is implemented via checking inputs, outputs and existence of a service that can meet the requirement S_i. The latter is implemented via variant matching. Hence it is possible for a version on the asset tree to achieve a partial match with the ARCM, but a (variant) child to achieve a perfect match. There exist several service matching algorithms in the literature; however, for partial matching, to the best of our knowledge, the existing algorithms fail to consider the input and output differences in the variation model, thereby making our algorithm novel.

Step 1: Given an ARCM, we traverse trees in the repository to determine relevant assets by matching the inputs and outputs.

Step 2: In order to determine the degree of match, there are three basic matching criteria: inputs, outputs, and the set of services in the ARCM against which the ACAM should be

Algorithm 1 ComputeFunctionalMatchScore(M_{RC})

```
1: for all M_CA do
2:   if ( I(M_RC) ∩ I(M_CA) = ∅ ) or (O(M_CA) ∩ O(M_RC) = ∅ ) then
3:     exit(-1)
4:   end if
5:   for all S_i ∈ M_RC & S_a ∈ M_CA do
6:     SC_i = MatchServiceVariants(S_i, S_a)
7:     WeightedSC_i = w_i * SC_i
8:   end for
9: end for
procedure MatchServiceVariants(S_i, S_a)
1: score = 0
2: if MatchService((S_i, S_a) ) then
3:   score = 2
4: else
5:   if ( {d_in_i} ⊃ {d^a_in_i} ) then
6:     {d^diff_in_i} = {d_in_i} − {d^a_in_i}
7:     score = 1
8:   end if
9:   if ( {d_out_i} ⊃ {d^a_out_i} ) then
10:    {d^diff_out_i} = {d_out_i} − {d^a_out_i}
11:    score = 1
12:  end if
13:  if ( {d_in_i} ⊂ {d^a_in_i} ) then
14:    {d^diff_in_i} = {d^a_in_i} − {d_in_i}
15:    score = 1
16:  end if
17:  if ( {d_out_i} ⊂ {d^a_out_i} ) then
18:    {d^diff_out_i} = {d^a_out_i} − {d_out_i}
19:    score = 1
20:  end if
21: end if
22: return(score)
```

matched.

Matching a service requirement against the services in the ACAM produces one of three results: *exact, partial, or disjoint* functional match, designated by scores of 2, 1, and 0, respectively. In an exact functional match, the requirement matches perfectly. For exact match, we can use any existing service matching algorithm (e.g., [11]) to detect exact service matching and returns true for exact match. If we do not get an exact match, then we deduce the input and output data differences and invoke *MatchServiceVariants()* from Algorithm 1 that uses the service variation model to detect the possibility and the score of creating a requested variant. In a partial functional match, the asset capabilities are a subset of the inputs and/or outputs of the requirement. A disjoint functional match is one that is neither exact nor partial.

The core step in functional matching involves the matching of a service requirement S_i against a single service S_a in an ACAM. Since S_i is specified via its input and output sets $\{d_{in_i}\}$ and $\{d_{out_i}\}$, this matching checks whether the service S_a possesses the same input and output sets $\{d^a_{in_i}\}$ and $\{d^a_{out_i}\}$. This also considers the case where S_a's input-output set does *not* match that of S_i, but whether a variant of S_a can be generated whose input-output set can match that of S_i. This involves several sub-cases depending on whether S_a's input (resp. output) set is a subset or superset of the input (resp. output) set of S_i, and is detailed in Algorithm 1.

If this check is successful for a suitable combination of variants, then a variant match exists, at least for the input set. The difference in the output sets is also checked similarly. A variant match with a variant of S_a, for a collection of variants applied on S_a, then exists.

Let the user-assigned relative weight of each service S_i in the ARCM be w_i, such that $\sum w_i = 1$. Let the match score for each service be Sc_i. The total match score for the specified services is $\sum w_i Sc_i$, and would be in the range [0-2$\sum w_i$].

2) Non-functional Matching: Non-functional matching is implemented against the resulting matches from functional matching. The constraints in the ARCM are matched against the advertised non-functional properties in the ACAM. For any constraint Cs_i, if its order is increasing, then the advertised non-functional property Ns_i should be such that $value(Ns_i) \geq value(Cs_i)$. For a constraint with a decreasing order, the condition to be met would be $value(Ns_i) \leq value(Cs_i)$ (see Algorithm 2).

Algorithm 2 ComputeNonFunctionalConstraintScore (M_{CA})

```
1: for all M_CA do
2:   totalscore=0
3:   for all C_i do
4:     score = 0
5:     if ( ord(C_i) = increasing ) then
6:       if ( NS_i > CS_i ) then
7:         score = 1
8:       end if
9:     end if
10:    if ( ord(C_i) = decreasing ) then
11:      if ( NS_i < CS_i ) then
12:        score = 1
13:      end if
14:    end if
15:    WeightedSC_i = w_i * score
16:    totalscore = totalscore + WeightedSC_i
17:  end for
18: end for
```

In an exact non-functional match, each constraint from the ARCM is satisfied by a non-functional property advertised in the asset's ACAM. In a partial non-functional match, at least one constraint is not satisfied. Given a set of constraints, each match (resp. non-match) against each constraint is tagged with a score of 1 (resp. 0).

Since different users rate non-functional properties differently, we assume similar user-assigned weights as for functional matching, i.e., a weight of κ_i for each non-functional property, such that $\sum \kappa_i = 1$. Then the aggregate match score for the ARCM-ACAM pair in question would be $\sum \kappa_i C'_i$, where $C'_i \in [0, 1]$ is the match score of the individual non-functional property.

3) Search Optimization via Asset Trees: A naive way to implement matching would be to compute the match score for S_i repeatedly for the same features (whether in the static or dynamic part) for every ACAM. We perform a depth-first traversal of the asset tree. When a particular version node in the asset tree has already been matched against the ARCM, we annotate that node as "visited" and move to its child. If the child in question is a variant node, then Definition 3 would detail the variations over the parent node, and the matching for that node would require only the checking of those particular variations. Alternatively, if the child is a version node, then Definition 4 would be used to determine matches only against those additional features that the child possesses over and above those of the parent.

The above optimization may still result in some variations being considered multiple times for matching, especially if a version node has many variant nodes as children, with many variations common among them. In order to eliminate this wastefulness, we maintain a hash table of variations and their respective match scores against the service requirement S_i being used. If that particular entry is encountered in any other variant node, then that match score can be reused.

This optimized match, along with depth-first traversal, can be implemented in $O(n+F)$ time, where n is the number of nodes in the asset tree, and F is the total number of version features plus variations among the assets in the asset tree.

V. Implementation and Experiments

The purpose of our implementation is to demonstrate our asset tree representation model and matching algorithm. We used a large collection of 900 insurance provisioning and claims business processes stored in an internal process repository following OMG's RAS (`http://www.omg.org/spec/RAS/`) specification. We created an ARCM with 7 functional requirements and 8 non-functional constraints, and used our matching algorithm to search against the 900 assets. The details of the requirements and constraints, along with their user-defined weights, are accessible from `http://bit.ly/fbKCF9`.

For conducting our experiments, we have constructed asset trees identifying processes that are functionally similar. This includes first identifying versions belonging to all assets and establishing the corresponding branches connecting the version nodes to their common base asset node with the help of version history. This basically exhibits parent-child relationships across the different levels for a given asset. This process is repeated for all candidate assets in the repository in a way, that an asset already identified as a version of some base asset is removed from the candidate assets.

Then the functional similarity is evaluated only against the identified common base assets to evaluate the variant relationship between the base assets. This involves identifying the variant base assets based on conducting the search from common filters associated with each of the base assets. In this step, a base asset identified and marked as a variant of another base asset is subsequently excluded from the search on other base assets. This ensures that a base asset (or its versions) can have a variant relationship to only one base asset. Similarly when a search is conducted for a base asset marked as variant (child) to another base asset (parent), the parent base asset is excluded from this search. This ensures that cyclic relationships between two variants is prevented. For our experimentation, we followed a simple convention of declaring a base asset as parent, if it is created before its identified variant which becomes the child. Finally the asset trees are constructed for each of the base assets as the root node, i.e., neither a version nor a variant to other base assets. This involves simpler integration of all the branches recursively for each child node (variant or version) of the root base asset node. Therefore an asset tree thus constructed contains a family of assets all originated from a common base root asset node either as versions or as variants. Our experimentation resulted in 126 asset trees for the total of 900 assets. Each asset tree on an average contained 7 processes either as versions or variants to the base process asset node.

Fig. 5 displays two charts. The top chart depicts the growth in the repository size over a period of 3 years; it can be seen that while the number of stored processes grows rapidly, the growth in the asset trees is close to linear. This is because in most cases, newly submitted assets were identified as either versions or as variants based on the search initiated with the submitted asset's metadata on all identified base assets. Once the relationship is identified with an existing base asset, then accordingly the new asset is positioned as a variant in the corresponding asset tree based on the position of the related base asset (root node or not) in that asset tree. The asset trees thus help in consolidating additions to the repository by grouping together variants and versions of existing business processes, thereby slowing down the growth in the search time as the repository size grows, to close to linear growth. This is because as the repository grows, the number of processes per asset tree grows (as shown in the top chart of Fig. 5), resulting in larger groupings of business processes per asset tree. Hence this grouping eliminates larger number of non-matched candidate assets as the repository grows in size. The bottom chart of Fig. 5 depicts the growth in the search times (in minutes) over the 3-year period. As Fig. 5 clearly illustrates, the search times for the repository with asset tress is less than those for the repository without them. This is because our approach now helps in searching against 126 assets (each representing the process tree) compared to 900 assets previously. Subsequently the ACAM descriptions for each of the associated nodes in the asset tree are matched for final selection.

The output of our matching algorithm is an ordered sequence of all such matches, with each match describing an ACAM that (fully or partially) meets the requirements in the ARCM, along with the non-functional match score. For each match, and for each matching feature, the details of the match are also pre-sent to the user. Such a display method also provides the user with sufficient flexibility to change their weights for either functional or non-functional match later on, after viewing the results..

We illustrate an implementation of our plugin on a set of 4 assets from the repository, using the asset tree depicted in Fig. 6. Let us assume an ARCM comprising requirements $R1$ through $R7$, and constraints $C1$ through $C8$. The relative weights as defined by the user are depicted in Table I; in this case, the user has decided not to consider $C7$ and $C8$, hence they are assigned zero weights.

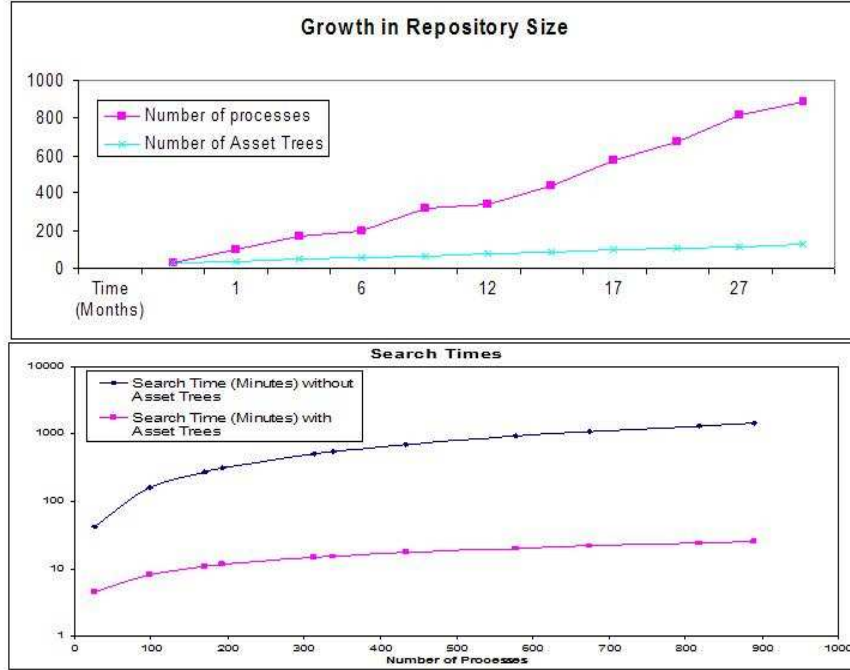


Fig. 5. Repository Growth and Search Times for Business Processes

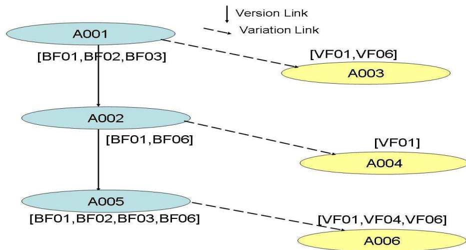


Fig. 6. Evolved Asset tree representation for running example

TABLE I
WEIGHTS OF FUNCTIONAL REQUIREMENTS AND NON-FUNCTIONAL CONSTRAINTS

Requirements / Constraints	Weights
R1,R2,R3,R6,R7,C3,C4,C5	0.1
R4,C1	0.3
R5,C2,C6	0.2
C7,C8	0.0

Let us assume that the Asset A001 of Fig. 6 possesses features BF01, BF02 and BF03, with Asset A002 possessing an additional feature BF06. Let us also assume that Variant A003 of Asset A001 comprises variations VF01 and VF06, whereas Variant A004 of Asset A002 contains only VF01 as its variation. Table II represents the respective weighted scores for the features that all the assets in the asset tree possess - this table is used to calculate the final score for each asset as depicted in Table III. We consider the values for full Match score and Partial Match score as 2 and 1 respectively

TABLE II
WEIGHTED SCORES OF INDIVIDUAL FEATURES

Feature	Full Matches	Partial Matches	Weighted Score for Feature
BF01	R4	C1	(2 *.3 + 1 * .3) = 0.9
BF02	R2, C5	R1	(2 * .1 + 2 * .1 + .1) = 0.5
BF03	R5	C2	(2 * .2 + 1 * .2) = 0.6
BF06	R6, C8	C4	(2 * .1 + 2 * 0.0 + 1 * .1) = 0.3
VF01	R3	C3	(2 * .1 + 1 * .1) = 0.3
V04	C6		(2 * .2) = 0.4
VF06	R7	C7	(2 * 1 + 1 * 0.0) = 0.2

for this illustration. Table III therefore displays Asset A006 as having the best match. Further details of these four assets are also accessible from `http://bit.ly/fbKCF9`.

VI. RELATED WORK

Workflow Reuse & Repositories: One of the earliest systematic attempts at formal workflow reuse was workflow patterns in order to facilitate reuse [12]. Recent work has also focused on requirements for process modeling tools to support pattern-based reuse [13]. However, issues with workflow reuse still persist as in [14]; the most relevant issues among them are lack of a comprehensive discovery model, lack of workflow fragment rankings, and difficulties in acquiring and storing process knowledge. The citation [15] presents the concept of BPEL process fragments in order to enhance BPEL process reuse. We view [15] as being complementary to ours, since process fragments can be used to enhance the reusability of business process implementations whose specifications are expressed using our model.

The citation [16] discusses how process model repositories can be "refactored", i.e., simplified, as they grow in size.

TABLE III

TOTAL WEIGHTED SCORES OF ASSETS

Asset	Total Weighted Score	Contained Base and Variation Features
Asset A001	(0.5 + 0.9 + 0.6) = 2.0	[BF01,BF02,BF03]
Asset A002	(0.9 + 0.3) = 1.2	[BF01, BF06]
Asset A003	(0.5 + 0.3 + 0.2) = 1.0	[BF01,BF02,BF03,VF01,VF06]
Assest A004	(.3 + 0.3) = 0.6	[BF01,BF06,VF01]
Asset A005	(0.5 + 0.9 + 0.6 + 0.3) = 2.3	[BF01,BF02,BF03, BF06]
Asset A006	(0.5 + 0.9 + 0.6 + 0.3 + .4 + .3 + .2) = 3.2	[BF01,BF02,BF03, BF06,VFO1,VF04,VF06]

This would enable process designers to effectively deal with model complexity by making process models better understandable and easier to maintain. *Business Process Similarity Search*: The citations [1], [17] discuss the storage, representation and searching of business process variants in a repository. However, they not distinguish between versions and variants, and mainly focus on structural aspects of business process models. As we have already argued in our paper, such a representation is not sufficient for the research problem that we are investigating. Similarity search algorithms have been proposed in [4], [5], [18], [6]. However, they primarily focus on the structural aspects of business process models represented as graphs.

Case-based Reasoning in Workflows: Using case-based reasoning (CBR) as a means of improving workflow management has been discussed in [19], [20]. In particular, the citation [20] provides a mechanism for storing process deviations as cases, which can be retrieved by providing appropriate contextual information. We view these works as complementary to ours.

Modeling Variability in Workflows: Works such as [21], [22], [23] have discussed how variability in workflows can be represented along with the workflow models themselves (in particular, via feature modeling such as in [21]), so as to help the business analyst choose a collection of variants that do not conflict with each other. However, those approaches are primarily targeted at modeling variants in a business process towards ease of representation and manipulation of users, and are complementary to our approach.

VII. CONCLUDING REMARKS

Modeling business process solutions as reusable assets facilitates reuse. We show that our contributions, working together, considerably improve the efficacy of service asset reuse; and we have implemented and demonstrated this on a large real-life collection of business processes in the insurance claims domain. Future work would investigate the following: extending the implementation of the asset tree to cover much coarser business architecture elements, incorporating case-based reasoning techniques, representing variants on the asset tree as derivable from other variants, enhancing our scoring technique to incorporate fractional scoring (by representing different scores for different variants in Algorithm 1) and incorporating past performance and user feed back of reusable assets into our matching algorithm.

REFERENCES

[1] R. Lu, S. W. Sadiq, and G. Governatori, "On managing business processes variants," *Data Knowl. Eng.*, vol. 68, no. 7, pp. 642–664, 2009.

[2] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, 2005.

[3] N. C. Narendra, K. Ponnalagu, B. Srivastava, and G. S. Banavar, "Variation-oriented engineering (voe): Enhancing reusability of soa-based solutions," in *IEEE SCC (1)*, 2008, pp. 257–264.

[4] Z. Yan, R. M. Dijkman, and P. Grefen, "Fast business process similarity search with feature-based similarity estimation," in *OTM Conferences (1)*, 2010, pp. 60–77.

[5] T. Jin, J. Wang, N. Wu, M. L. Rosa, and A. H. M. ter Hofstede, "Efficient and accurate retrieval of business process models through indexing - (short paper)," in *OTM Conferences (1)*, 2010, pp. 402–409.

[6] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," *Inf. Syst.*, vol. 36, no. 2, pp. 498–516, 2011.

[7] R. M. Dijkman, M. Dumas, and L. García-Bañuelos, "Graph matching algorithms for business process model similarity search," in *BPM*, 2009, pp. 48–63.

[8] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data Knowl. Eng.*, vol. 68, no. 9, pp. 793–818, 2009.

[9] H. Guo, A. Ivan, R. Akkiraju, and R. Goodwin, "Learning ontologies to improve the quality of automatic web service matching," in *ICWS*, 2007, pp. 118–125.

[10] S. Frolund and J. Koistinen, "Qml: A language for quality of service specification," *HP Labs Technical Report*, 1998.

[11] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic matching of web services capabilities," in *International Semantic Web Conference*, 2002, pp. 333–347.

[12] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[13] L. H. Thom, J. M. Lau, C. Iochpe, and J. Mendling, "Extending business process modeling tools with workflow pattern reuse," in *ICEIS (3)*, 2007, pp. 447–452.

[14] A. Goderis, U. Sattler, P. W. Lord, and C. A. Goble, "Seven bottlenecks to workflow reuse and repurposing," in *International Semantic Web Conference*, 2005, pp. 323–337.

[15] Z. Ma and F. Leymann, "Bpel fragments for modularized reuse in modeling bpel processes," in *ICNS*, 2009, pp. 63–68.

[16] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers, "Refactoring large process model repositories," *Computers in Industry*, vol. 62, no. 5, pp. 467–486, 2011.

[17] C. Li, M. Reichert, and A. Wombacher, "Mining business process variants: Challenges, scenarios, algorithms," *Data Knowl. Eng.*, vol. 70, no. 5, pp. 409–434, 2011.

[18] M. Kunze and M. Weske, "Metric trees for efficient similarity search in large process model repositories," *Proceedings of International Workshop "Process in the Large" (IW-PL10)*, 2010.

[19] T. Madhusudan, J. L. Zhao, and B. Marshall, "A case-based reasoning framework for workflow model management," *Data Knowl. Eng.*, vol. 50, no. 1, pp. 87–115, 2004.

[20] B. Weber, M. Reichert, S. Rinderle-Ma, and W. Wild, "Providing integrated life cycle support in process-aware information systems," *Int. J. Cooperative Inf. Syst.*, vol. 18, no. 1, pp. 115–165, 2009.

[21] S. Deelstra, M. Sinnema, and J. Bosch, "Variability assessment in software product families," *Information & Software Technology*, vol. 51, pp. 195–218, 2009.

[22] E. Santos, J. Castro, J. Sánchez, and O. Pastor, "A goal-oriented approach for variability in bpmn," in *WER*, 2010.

[23] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing variability in business process models: the provop approach," *Journal of Software Maintenance*, vol. 22, no. 6-7, pp. 519–546, 2010.