A Goal-Directed Modeling Technique towards Business Process

Yuqun Zhang, Dewayne E. Perry
Center of Advanced Research in Software Engineering (ARiSE)
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
Email: yuqun_zhang@utexas.edu, perry@mail.utexas.edu

***Abstract*—The modeling techniques for business process are mostly graphics-based, that is argued to be simplified when the processes become too complex or expanded to provide full details. In this paper we propose a technique for modeling composite activities by including components of data, human actors and atomic activities. Furthermore, we represent business processes with composite activities using process-oriented languages. To explore the properties of the business processes under this modeling technique, we define a set of metrics that reflect the degree of data aggregation and human actor involvement of executing composite activities. Algorithms to calculate these metrics are derived based on the goal-directed nature of the modeling technique for composite activities and business processes. At last by providing a use case, we discuss the artifacts and future work for this modeling technique.**

***Index Terms*—business process, composite activity, goal-directed**

I. Introduction

Business process modeling, the goal of which is to represent, analyze and improve the processes of enterprises in the discipline of systems engineering [2], has been an emerging research interest. The approaches in this area range from modeling techniques along with their artifacts, to the verification and evaluation of the business process properties. With current IT technologies, business processes are more and more automatically realized by, and integrated with, IT functionalities, i.e., the enterprise software systems. On the other hand, asynchronous patterns of development of business processes and IT systems easily lead to a gap between them [10] [12] [13]. This interrelation is covered in the research area of Enterprise Architecture [29] and Service-oriented Architecture [34] [11].

Traditional modeling techniques for business processes mainly include flowcharting [28] [33] [21], IDEF family [22] [19], Petri Nets [32], etc,. While each of these graphics-based techniques models business processes with their preferred emphasis, it is argued in [25] [4] [14] that these approaches are incomplete, implicit and vulnerable in handling the business process modeling for complex business goals. The BPEL family [23] [20] [6] provides XML-based executable languages that specify business processes to support web services. While it facilitates the expansion of automatic business processes, it is nevertheless viewed as being user-unfriendly [25].

Although business processes are usually modeled by the research community as encapsulations of business activities and their corresponding execution order [35] [24], the views of business activities remain ambiguous and disjoint. Some work [12] [13] simply assume business activities as actions taken towards specific business goals. BPML [27] defines business activities to two levels: complex activities that are composed of atomic activities. Web-based techniques [23] [20] [6], on the other hand, view business activities as the business logic that can be accessed via IT-system interfaces for the purpose of automatic execution. These views of business activities are proper as the basis of their corresponding approaches of business process modeling. However, they fail to provide an explicit mechanism of modeling business activities in a deterministic manner.

In this paper, we study a technique to explicitly model composite activities that represent business processes with a well-defined syntax, that is reified from modeling techniques for software processes [31] [30]. A composite activity is specified as a tuple of components: data, human actor, and atomic activity, where data is further modeled to indicate its states in data flows within a composite activity. As in [27], an atomic activity is defined as the activity which executes a single unit of work. That means partial data or activity generated within an atomic activity which does not impact on the service is not represented. Human actor refers to role executing or involved with atomic activities in the composite activity. Note that the modeling of a composite activity starts with ensuring that all atomic activities are executed by specified roles. For the sake of brevity, we do not take time or location into consideration for service modeling.

Business processes are represented with composite activities by their execution order. We develop a set of *assumption-while-result-final* functions with composite activities as their parameters to model the activity order in business processes in the manner of process-oriented languages. A business process is initiated with an *assumption()* function. A *while()* loop represents an execution block, which is determined by the divergence or convergence relations among composite activities. A *while()* loop is called when these relations are detected in a business process. A *result()* function is called to deliver composite activities as the outputs of the associated

while() loops. The *result()* function marks the end of the current execution block and the beginning of the next one. A *final()* function indicates the end of the entire business process. A business process typically consists of one *assumption()* and *final()* function, one or more *while()* loops and *result()* functions. One *while()* loop takes the activities of *result()* functions of other *while()* loops as trigger for execution, and this dependency serializes the order of execution of composite activities.

The use of this *assumption-while-result-final* function set enables a specific execution order not only of composite activities, but also of data flows and human actor involvement within and between composite activities. To explore the properties of business processes in terms of composite activities and modeling components of composite activities, we define two sets of metrics: data dependencies and labor force factor. In our approach, data dependencies indicate for a single piece of data or composite activity, how much other data or how many other composite activities impact it, and vice versa. There are four types of dependency: data-to-data dependency, data-to-activity dependency, activity-to-data dependency and activity-to-activity dependency. The labor force factor indicates the degree of human actors being involved with a business process or a composite activity. We also provide algorithms for calculating these metrics.

A goal-directed technique in business process modeling intends to model components of business processes and present their properties under the pre-set business goals [8]. Our modeling technique for business processes is goal-directed because the detection of the metrics above could be initiated by identifying the final states of their corresponding components. These final states, e.g., those located in the *final()* functions, are analogous to business "goals". They mark the end of the workflows of the corresponding components and can be used to evaluate their properties, i.e., data-to-data dependency, data-to-activity dependency, etc., by backtracking these workflows until the initialization of these components are detected.

A use case of purchasing an apartment and obtaining a loan is used in our work to illustrate the process of modeling composite activities and calculating the metrics. We compare the results of our modeling the use case with the BPMN model in [35], and discuss the uses of the metrics as guidance in evaluating business processes.

The rest of the paper is organized as follows. Section II introduces related work. Section III describes the details of our modeling technique for composite activities and business processes. Section IV explains the definition of the metrics related to the modeling technique and the corresponding algorithms to calculate them. Section V presents a use case to illustrate the process of applying the modeling technique and calculating the metrics. Section VI discusses our conclusions and future work.

II. Related Work

A number of approaches have been used to provide techniques for modeling business processes. Flowcharting, showing the scope of the whole business process and tracking the flows of its associated information, was initially proposed by Schriber in [33]. Subsequent work, such as [21], improves this technique by introducing diagrams to include the principles of decision sciences. BPMN [28] standardizes the flowcharting diagrams to bridge the communication gap between business process design and implementation, and is now widely used [12] [13]. The IDEF family integrates business processes and data structures. IDEF0 [22] is designed to model the activities in terms of organizations, while IDEF3 [19] is modeled as process flows to indicate how organizations work. Role activity diagrams [5] concentrate on modeling roles with their associated activities and relations in the context of roles execution as critical resources. These graphics-based techniques excel in the global demonstrations of business processes. However, graphics can be simplified when business processes become too complex or expanded to provide full details. The XML-based modeling languages, such as BPEL [23] and BPML [27], are designed as the executable languages for specifying the activities of business processes to support web services. Though capable of "expressing executable processes that address all aspects of enterprise business processes", it is argued that they are user-unfriendly and fail to convey the concepts of human actors and data models [25]. Our approach, while maintains the advantages of the techniques above, additionally enables the flexibility of presenting complex business processes with including more concepts, such as human actors and data models.

Business activities are sometimes viewed as actions executed in business processes, as in [12] and [13]. Moreover, for different research needs, business activities sometimes require multiple views within one approach, e.g., [37], where they are viewed purely as actions of business logics in one example while viewed as compositions of actions of business logics and data in another example. Some approaches realize the modeling of business activities by viewing business activities as representations of different levels of granularity. For instance, in BPML [27], business activities are categorized into two types: complex activities and simple activities. A complex activity is an activity that contains one or more simple activities, establishes a context for their execution, and directs their execution by a definition of a specified activity set. Our approach adopts the design of composite (complex) activities and embeds data types, human actors and atomic activities inside to explore properties formed by inter-composite-activity relations.

The identification and measurement of metrics of business process modeling has recently become an interest for researchers. A group of researchers have adapted the concepts of metrics in the discipline of software engineering and map them to be the needed metrics of business process modeling. For instance, a typical software metric, lines of code (LOC), is logically mapped to derive the "number of activities" in [17]. The authors of [3] derive a set of metrics for business processes after reifying the metrics of software processes. Moreover, some researchers propose metrics for evaluating

certain nonfunctional requirements. A metric of weighted coupling is designed in [36] to evaluate the degree of coupling between different types of interrelations of business activities and processes. Metrics for evaluating the similarity between process models, are provided from the label-based, structural, and behavioral dimensions in [9]. IT-level metrics are favored to describe the artifacts and performance of IT-systems in the realm of service-oriented business processes [38]: business-level, application-system-level, and the IT-infrastructure-level attributes are considered to be critical in representing the properties of the a networked-service, which is implemented by or implements other services of business processes.

Dependency among the metrics of business process modeling refers to the degree of one component of a business process relying on other components. The researchers of [15] generate one-to-one, one-to-many, and many-to-many dependencies to describe the degree of how objects and activities of business processes rely on each other. By exploring the dependencies between local business processes and global business processes, the authors of [16] define the metric inter-process dependencies and extend a UML-based activity diagram to all levels. In our approach, we model dependency in terms of data flows to present their patterns in business processes. In addition, we design labor force factor to indicate the degree of human actor being involved with business activities and processes, which has not yet been addressed by existing approaches. These metrics help demonstrate the quality of business process modeling and provides reference for potential business process optimization.

III. Modeling Technique Overview

In this section we provide an overview of our modeling technique. Our approach provides syntax for product and policy models, as in [31] [30]. Typically, product models define the type of components that constitute composite activities. Policy models define the logical connections among composite activities to reflect the business processes.

A. Product Models

A composite activity consists of the following components: *data, role,* and *atomic_activity*. Due to the states in data flows within a composite activity, *data* is further classified to *initial_data, input_data, global_data, consumed_data, output_data,* and *final_data*. The object declarations of each component is listed as follows:

```
type initial_data     primitive;
type input_data       primitive;
initial_data application_form;
input_data ready_to_start;
...
type global_data      primitive;
type consumed_data    primitive;
type output_data      primitive;
type final_data       primitive;
...
type role             primitive;
role applicant;
role government_employee;
...
type atomic_activity  primitive;
atomic_activity submit_application;
atomic_activity obtain_approval;
...
```

These object declarations, as in [30], include type definitions, type instances, and object definitions. Types and type definitions enable the appropriate abstractions and their values. The components are considered to be primitive and their objects are accordingly assigned. Note that data types are not limited only to data objects, but can also be states. A sample composite activity "apply for loan" and the illustration of its components is stated as follows:

```
activity apply_for_loan
   {
   role applicant;
   initial_data application_form;
   input_data empty;
   global_data government_ID;
   consumed_data application_form;
   atomic_activity <submit_application_form>;
   output_data wait_for_approval;
   final_data empty;
   }
```

Each of the components of a composite activity is defined and illustrated as follows:

1) initial_data:

Definition 1: An *initial_data* is the data that is initially injected to the data flows and triggers the execution of the associated composite activity along with *global_data* and *input_data* (if there is any).

For any data, it is allowed to be an *initial_data* only once in each of its flows. An *initial_data* indicates the initial appearance of this data on the flow. In other words, this data is not included in the composite activities that are previously executed of the flow as any data type.

2) input_data:

Definition 2: An *input_data* is the data that is generated and delivered by other composite activities, and triggers the execution of the associated composite activity along with *global_data* and *initial_data* (if there is any).

Theorem 1: for any *input_data*, there must be at least an identical *output_data* in a composite activity that is executed beforehand, and vice versa.

Similar to [18], since *input_data* marks the beginning of the associated execution of a composite activity, the *output_data* where it is delivered from must be located in a composite activity which has already been executed. It is possible that the composite activity associated with this *output_data* is not necessarily executed directly ahead of the composite activity associated with the identical *input_data*. In our approach, our rule is that an *initial_data* cannot be an *input_data* in the same composite activity.

3) global_data:

Definition 3: A *global_data* is the data that is injected independently from business processes, and, triggers the execution

of the associated composite activity along with *initial_data* and *input_data* (if there is any).

A *global_data* is not an *output_data, final_data,* or *consumed_data* of any composite activity, due to the fact that it is not the output of any composite activity. An example of *global_data* is "government ID" in the sample composite activity "apply for loan".

4) consumed_data:

Definition 4: A *consumed_data* is *input_data* or *initial_data* that is consumed by the execution of the associated composite activity.

Consumed_data indicates the consumption of *input_data* or *initial_data*; it is not *output_data* in the associated composite activity. *Consumed_data* represents the end of the associated data flow (there might, however, be other running flows of this data).

5) output_data:

Definition 5: An *output_data* is the data that is the deliverable by the execution of the associated composite activity.

As in Theorem 1, for an *output_data*, there is an identical *input_data* or *initial_data* in a composite activity that is executed later, unless it is a *final_data*. For any *input_data* that is not consumed in the associated composite activity, we generate an identical *output_data* in this composite activity.

6) final_data:

Definition 6: A *final_data* is the data that is generated by the execution of a business process and not consumed by any composite activity, or a state to indicate the end of the execution of a business process.

A *final_data* is represented only in a composite activity that is an end of a business process (there might be multiple composite activities that mark the end of the execution of a business process). Note that one composite activity where an identical *output_data* is represented is not necessarily executed directly ahead of the composite activity associated with this *final_data*.

A *final_data* and *consumed_data* reflects a business goal in terms of data type. They are considered as sources of our goal-directed approach to evaluate their relevant properties by backtracking them in the chain of composite activities.

7) role:

Definition 7: A *role* is a human actor involved with the execution of a composite activity.

A composite activity is executed by only one *role* or one specified group of roles. In this paper we do not consider the case of multiple roles executing one composite activity.

8) atomic_activity:

Definition 8: An *atomic_activity* is the activity that executes a single unit of work within the associated composite activity.

An *atomic_activity* cannot be further decomposed. Any data flow or human actor involvement that is included within an *atomic_activity* is not represented in the modeling of composite activities.

Overall, a composite activity is one or more atomic activities that are executed under one *role* or one specified group along with a composition of *initial_data, input_data, global_data, consumed_data, output_data,* and *final_data*. For the sake of simplicity at this stage of our research, we do not address the issues involved in composite activities being components of composite activities and restrict our focus only atomic activities. It is reasonable because according to our activity structure, any composite activity inside of composite activities can be reduced to a composition of atomic activities, data, and human actors, and at last lead to our modeling of composite activities. In the rest of this paper, we use the product models to represent their respective notions to maintain consistency.

B. Policy Models

Policy models are classified into two categories that depict the logical connections of composite activities: execution order of composite activities and inter-activity relations. This is how composite activities incorporate each other as the trigger or results of one execution block. A sample of a business process represented by our policy models is listed as follows, where all the capital letters represent composite activities.

```
process sample
   {
   assumption(A and B);
   while(A)
      {
       C;
       G;
       result(D and E};
       }
   while(B)
      {
       result(F cooperative (E and B));
       }
   while(E)
      {
       result(F cooperative (E and B));
       }
   while(D)
      {
      ...
       }
   final(F and ...);
   }
```

1) execution order of services: A business process is represented by an execution order of composite activities, that is represented by a set of *assumption-while-result-final* functions, where composite activities or their interrelations are used as their parameters. The syntax of the functions are illustrated as follows:

- *assumption()*
  An *assumption()* function is only called in the beginning of the execution of a business process. Composite activities or their interrelations, as its parameters, are initially executed in this business process, such as composite activities A and B in the sample process. Typically, there is at least one *initial_data* and no *input_data* in the composite activities of this function.
- *while()*
  A business process is embodied with one or more *while()* loops. The boundary of one *while()* loop is typically

determined by the divergence or convergence relations among multiple composite activities. That means the initiation/end of one *while()* loop are these interrelations of multiple composite activities . A *while()* loop takes either the composite activities of an *assumption()* function or a *result()* function of other *while()* loops, to trigger its execution. In this paper, our rule is that for one *while()* loop, there is only one composite activity as its parameter. That indicates for one *result()* function with interrelations of multiple composite activities as its parameters, there are an identical number of *while()* loops as those composite activities (excluding composite activities of *final()* function), such as *while*(D) loop and *while*(E) loop corresponding to the *result*(D and E) function of *while*(A) loop.

- *result()*
  A *result()* function is located in a *while()* loop to indicate the end of the execution of this *while()* loop. It is called when the execution of the associated *while()* loop detects the divergence or convergence of multiple composite activities, or composite activities of *final()* functions. Only one *result()* function is allowed in one *while()* loop. That means the body of a *while()* loop is the execution of composite activities with a rigourous order. For instance, in *while*(A) loop of the sample process, composite activities C and G are executed until the composite activities D and E together as deliverables are detected and both encapsulated as the parameters of a *result()* function.
- *final()*
  A *final()* function is placed in the end of a business process, with one composite activity or multiple parallel composite activities connected by "and", as its parameters. Typically in composite activities that are the parameters of a *final()* function, there is at least one *final_data* that indicates the business goal(s) of the execution of the business process.

The set of *assumption-while-result-final* functions leads to a flexible expression of execution order of a business process. A *while()* loop is not necessarily placed right following *while()* loops where its parameters result from. In the sample process, *while*(D) loop is not placed following *while*(A) loop where the composite activity D is delivered. This flexibility enables an easy implementation of modeling business processes.

2) inter-activity relations: In this paper, we use a set of operators to represent the relations between composite activities of *result()* functions as follows:

- *cooperative*
  One or more composite activities together proceed to the subsequent executions. One sample form is *result*(A *cooperative* (B and C and D)). This means the composite activity A is the result of the execution of all the composite activities B, C and D.
- *exclusive*
  Only one out of multiple composite activities is allowed to proceed to the subsequent executions. One sample form is *result*(A *exclusive* (B and C and D)). This means the composite activity A is the result of the only one execution of the composite activities B, C, and D.
- *N-cooperative*
  Any N out of M composite activities (N<=M) or more together proceed to the subsequent executions. One sample form is *result*(A *2-cooperative* (B and C and D)). This means the composite activity A is the result of two or more executions of the composite activities B, C, and D.
- *N-exclusive*
  Only N out of M composite activities (N<=M) or less together proceed to the subsequent executions. One sample form is *result*(A *2-exclusive* (B and C and D)). This means the composite activity A is the result of not more than two executions of the composite activities B, C, and D.

Composite activities that are placed ahead of the operators (i.e., *cooperative, exclusive,* etc,.) are defined as result composite activities. Composite activities that are placed after these operators trigger the result composite activities under our semantics. Robust business processes are assumed in our research at this stage. That means exceptions of a business process are not considered, and result composite activities are deterministic. In other words, they are either one single composite activity or multiple composite activities connected by the connector "and". Note that the inter-activity relations of *result()* functions can be complex when multiple operators are placed between them, e.g., result(...*cooperative* (...*N-exclusive* (...) and (...))). This complexity contributes the overall complexity to the entire business process with graphics-based modeling techniques. Intuitively, our approach provides an easy way to model complex business process.

Modeling a business process by using our product and policy models is very flexible. Usually, modeling components of composite activities can be parallel to modeling a business process with composite activities. Moreover, a *whole()* loop does not have to be in order and is essentially a sub-business-process that provides partial business functionalities. This property provides a possibility that modification of a business process might be restricted to only one or more *while()* loops.

IV. Performance Metrics

To explore the interrelations of and between composite activities and their components, we develop a set of performance metrics that are classified into two categories: data dependencies and labor force factor. Data dependencies refer to the degree of data distribution, and measures for a single piece of data or its associated composite activity, how much other data or composite activities it is related to within its flows. The labor force factor measures the workload of *role*(s) being involved with a composite activity or a business process. Data dependencies are further modeled to data-to-data dependency, data-to-activity dependency, activity-to-data dependency, activity-to-activity dependency. Data types involved

in the calculation of these metrics do not include *global_data*, because they are considered to remain constant.

A. Data-to-Data Dependency

In our approach, a successful generation of an *output_data* relies on the execution of its associated composite activity when all the *input_data/initial_data* are ready. Accordingly, in one composite activity, *output_data* depends on *input_data/initial_data* with this dependency being reflected by the number of the *input_data/initial_data*. Typically, the more *input_data/initial_data* a composite activity has, the more dependent its generation of *output_data* is, and vice versa.

In a business process, any data (non-global-data) ends as either *consumed_data* or *final_data* in its flows. By localizing its final state and backtracking its flows, (that is, its *output_data* type in every composite activity it is associated with), the *input_data/initial_data* of all these composite activities can be obtained. The accumulation of the number of these *input_data/initial_data* is defined as the data-to-data dependency of this data.

The calculation of data-to-data dependency of the target data, namely D_{target} is initiated with the localization of the composite activity, namely A_{sink}, where D_{target} is its *final_data* or *consumed_data*. A tracking process is then conducted to detect the composite activity $A_{intermidiate}$ that is latest executed with D_{target} as its *output_data* before A_{sink} in the context of a business process. The number of all the *input_data/initial_data* of $A_{intermidiate}$ is added to data-to-data dependency of D_{target}. If D_{target} is *input_data* in $A_{intermidiate}$, that indicates at least one composite activity is executed that delivers D_{target} to be the *input_data* of $A_{intermidiate}$. This process of tracking and calculating data-to-data dependency of D_{target} continues until the composite activity A_{source} is detected where D_{target} is its *initial_data*. That means this A_{source} is the very beginning of the generation of D_{target}.

Ideally, backtracking flows of D_{target} is conducted in a transparent manner where only one composite activity is executed before or after another. This tracking process can be challenging when a composite activity that is associated with D_{target} is detected in a *result()* function with complex interrelations between multiple composite activities. The execution order of these composite activities needs to be well understood for tracking a business process. To explicitly present the execution order of composite activities in a business process, we use a sophisticated technique, namely, an *abstract syntax tree (AST)*, that is a tree representation of the abstract syntactic structure of source code written in a programming language [1] [26]. An example of an *AST* to depict *result*((A *2-exclusive* (E and F and G)) *cooperative* (B and C and D)) is shown in Fig.1 with all the capital letters representing composite activities.

The *AST* under our syntax provides following properties.

- Operators (defined to include only *cooperative, exclusive, N-cooperative, and N-exclusive*) are placed as none-leaf nodes. Leaf nodes must be composite activities. Composite activities can be both non-leaf and leaf nodes.
- Composite activities are placed left when its parent node is one of the operators (i.e., cooperative, exclusive, N-cooperative, N-exclusive). Multiple result composite activities under one operator are individually placed as the child nodes of the operator. They are not allowed to be placed with the connector "and" as their parent nodes. Composite activities that trigger these result composite activities are placed under "and" following the operators, as B, C, and D of Fig.1. This syntax restricts the use of "and" and provides determinism.
- Leaf nodes that are placed in lower levels are executed before those placed in higher levels.
- Operators are associated with the connector "and" as one of their child nodes. The lowest leaf nodes (composite activities) are either placed under "and" or other composite-activity-nodes. To prove this property, assume the lowest leaf nodes can be placed under operators. Since those operators are defined to be followed with "and", that indicates there are lower leaf nodes that are placed under "and". This disproves the assumption that the lowest leaf nodes can be placed under the operators, such as E, F, and G in Fig.1

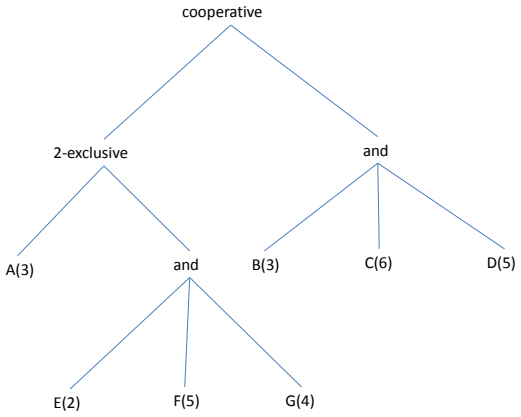

Fig. 1. An AST example

These properties contributes to the algorithm of calculating data-to-data dependency, that is initiated by identifying the composite activity, namely CA_{target}, where D_{target} is its *consumed_data* or *final_data* in Algorithm1. By setting D_{target} or its relevant operator as the root, an abstract syntax tree whose leaf nodes are composite activities where D_{target} is *initial_data*, is built to represent the business process. The paths from the root to each leaf node, namely *P* are printed and tailored by replacing the composite-activity-nodes with the respective number of their *input_data/initial_data*, when D_{target} is their *output_data*; and with zero otherwise. Then these tailored paths, namely *TP* are sorted by their lengths and mutual commonality. In our approach, leaf nodes that are placed under the same operators, e.g., E, F, and G of

Fig.1 distinguish their corresponding set of *P* from each other only by their last one elements (i.e.,leaf nodes). This property provides the possibility to merge the corresponding set of *TP* by simply manipulating their last one elements (i.e., the number of the *input_data/initial_data*) under the specific syntax (details of calculation under the syntax offered in Algorithm.1). A new *TP* is merged by deleting the last two elements of the original set of *TP*s and complementing the manipulated number. This merged *TP* is sorted with the rest of *TP*s. Starting from the longest *TP*s, this sorting and merging process is iterated until all of the *TP*s are merged and the final manipulated number is obtained as the data-to-data dependency. Note that in our algorithm, details of *AST*, printing path, sorting, tailoring, and merging are not addressed because they are sophisticated techniques that are not our concern.

Algorithm 1 DTD_calculation(data): calculating data-to-data dependency

Input:
D_{target},
CA_{target} := composite activities where D_{target} is consumed_data or final_data,
T_{AST} := tree generated by *AST*,
P := set of paths with the number of *l*,
TP := set of tailored paths with the number of *l*
temporal, m, k := interger
Output:
$Dependency_{DTD}(D_{target})$:= data-to-data dependency of D_{target}

1: T_{AST}:= *AST*(CA_{target});
2: {*P*} := *PrintPath*(T_{AST});
3: {*TP*} := *tailor*({*P*});
4: {*TP*} := *sort*({*TP*});
5: k := number of elements of TP_l;
6: **while** {*TP*} is not empty **do**
7: m := number of *TP* that differs from TP_l with only the last one element;
8: **if** (TP_l[k-2] := '*cooperative*' or '*N-cooperative*' or '*exclusive*' or '*N-exclusive*') or (TP_l[k-2] := 'and' and TP_l[k-3] := '*cooperative*' or '*N-cooperative*' or 'and' or empty) or (TP_l[k-2] is a number) **then**
9: **for each** *TP* from TP_{l-m} to TP_l **do**
10: *temporal* := *temporal* + *TP*[k-1];
11: **end for**
12: TP_{l-m} := TP_{l-m} -TP_{l-m}[k-1] - TP_{l-m}[k-2] + *temporal*;
13: delete from TP_{l-m+1} to TP_{l-m+1};
14: *l* := *l* - m;
15: k := k - 1;
16: *sort*({*TP*});
17: **end if**
18: **if** TP_l[k-2] := 'and' and TP_l[k-3] := '*exclusive*' **then**
19: **for each** *TP* from TP_{l-m} to TP_l **do**
20: find the largest *TP*[k-1];
21: **end for**
22: *temporal* := the largest *TP*[k-2];
23: TP_{l-m} := TP_{l-m} -TP_{l-m}[k-1] - TP_{l-m}[k-2] + *temporal*;
24: delete from TP_{l-m+1} to TP_{l-m+1};
25: *l* := *l* - m;
26: k := k - 1;
27: *sort*({*TP*});
28: **end if**
29: **if** TP_l[k-2] := 'and' and TP_l)[k-3] := '*N-exclusive*' **then**
30: **for each** *TP* from TP_{l-m} to TP_l **do**
31: find the N largest *TP*[k-1];
32: **end for**
33: *temporal* := *temporal* + the N largest *TP*[k-1];
34: TP_{l-m} := TP_{l-m} -TP_{l-m}[k-1] - TP_{l-m}[k-2] + *temporal*;
35: delete from TP_{l-m+1} to TP_{l-m+1};
36: *l* := *l* - m;
37: k := k - 1;
38: *sort*({*TP*});
39: **end if**
40: **end while**
41: $Dependency_{DTD}(D_{target})$:= *temporal*;

We use Fig.1 to illustrate how sorting and merging process works. Assume D_{target} is initialized in E, F, and G, consumed in A, and is *output_data* in the other composite activities. By applying the *abstract syntax tree (AST)* and sorting techniques, the set of paths from root (*cooperative*) to every leaf node is represented as {(*cooperative*, and, B), (*cooperative*, and, C), (*cooperative*, and, D), (*cooperative*, *2-exclusive*, A), (*cooperative, 2-exclusive*, and, E), (*cooperative, 2-exclusive*, and, F), (*cooperative, 2-exclusive*, and, G), }. Then they are transformed to the "compound" paths by replacing the composite-activity-nodes with their respective number of *input_data/initial_data* (the number associated with the composite activities in Fig.1), that is, {(*cooperative*, and, 3), (*cooperative*, and, 6), (*cooperative*, and, 5), (*cooperative*, *2-exclusive*, 3), (*cooperative, 2-exclusive*, and, 2), (*cooperative, 2-exclusive*, and, 5), (*cooperative, 2-exclusive*, and, 4) }. The last three paths are detected to be similar to each other with the only difference of their respective number of *input_data/initial_data*. Because the composite activities in the original paths are executed as the trigger of the operator "*2-exclusive*", the two largest number of *input_data/initial_data* (i.e., 4 and 5) of them are added to be the last one element of a merged path, represented as (*cooperative, 2-exclusive*, 9). This merging process repeats and the results in the interim set of paths are represented as {(*cooperative*, and, 3), (*cooperative*, and, 6), (*cooperative*, and, 5), (*cooperative, 2-exclusive*, 3), (*cooperative, 2-exclusive*, 9)}; {(*cooperative*, 12), (*cooperative*, and, 3), (*cooperative*, and, 6), (*cooperative*, and, 5)}; {(*cooperative*, 12), (*cooperative*, 14), }. At last, the data-to-data dependency of D_{target} is calculated as 26.

Data-to-data dependency, along with other dependencies in our approach, are considered as "worst-case" metrics, that is, the calculation of those metrics aims at providing a possibly maximum value to depict the dependencies among objects, such as the example above.

Note that backtracking composite activities might be nondeterministic. For instance, a piece of data might be *consumed_data* or *final_data* of multiple composite activities, because they are in parallel executed under the same operators, directly or indirectly. To calculate data-to-data dependency of this case, we simply need to make a root that includes these composite activities as its child nodes before proceeding to *AST*. Moreover, one composite activity might be detected in multiple *result()* functions of *while()* loops under our modeling syntax. In this case, a *result()* function with its associated *while()* loop is randomly selected in the context of the business process to apply *AST* for subsequently backtracking. Assuming a robust modeling of a business process under our policy models, this random backtracking is able to provide the complete paths of composite activities.

Typically, data-to-data dependency describes the degree of D_{target} flows being impacted by other data flows of a business process. A larger data-to-data dependency indicates D_{target} flows relying on more data flows associated of the business process, and vice versa.

B. Data-to-Activity Dependency

In a business process, D_{target} is delivered in its flows by the successful execution of the composite activities where it functions as *output_data*. On the other hand, malfunctions of the executions of the composite activities cause the failure of delivering D_{target} and the business process. The more composite activities where D_{target} is their component, the more D_{target} is impacted by the executions of these composite activities, and vice versa. This number of the composite activities where D_{target} is detected as their component is defined as data-to-activity dependency.

The calculation of data-to-activity dependency differs from that of data-to-data dependency only in the way of accumulation, i.e., data-to-activity dependency is only incremented by 1 each time when it is updated as in Algorithm1.

C. Activity-to-Data Dependency

Generally, the successful execution of a composite activity relies on all its *input_data/initial_data* being ready, and ensures the data flows and business process by delivering its *output_data* to the following composite activities. The more *input_data/initial_data* one composite activity has, the more it is impacted by their flows; On the other hand, the more *output_data/final_data* it generates, the more data flows it impacts, and vice versa. The interrelation of this composite activity and its relevant data flows is defined as activity-to-data dependency, that is accurately reflected as the summation of data-to-data dependencies of all its *initial_data, input_data, output_data*, and *final_data* of a composite activity.

Note that we do not simply define activity-to-data dependency as the number of all the data of a composite activity, because only the data number of a composite activity does not necessarily depict the degree of the interrelations of the composite activity and its relevant data flows. For instance, composite activity A includes a lot of data yet their flows are limited in the business process; composite activity B includes a close number of data as A and its relevant data flows span all over the business process. It is not accurate to perceive A and B have the similar activity-to-data dependency just by their close data numbers.

The algorithm to calculate activity-to-data dependency is illustrated in Algorithm2.

Algorithm 2 ATD_calculation(activity): calculating activity-to-data dependency

Input: CA_{target} := composite activity for the calculation of activity-to-data dependency
Output: $Dependency_{ATD}(CA_{target})$

1: **for each** *data* $\in$ *input_data, initial_data, output_data*, and *final_data* **in** CA_{target} **do**
2: DTD_calculation(*data*);
3: $Dependency_{ATD}(CA_{target})$:= $Dependency_{ATD}(CA_{target})$ + $Dependency_{DTD}$(*data*);
4: delete *data*;
5: **end for**

Note that for the data that is both input_data and output_data of CA_{target}, the replication of it being counted to $Dependency_{ATD}(CA_{target})$ needs to be avoided. This is ensured by conducting "delete *data*" in Algorithm.2.

D. Activity-to-Activity Dependency

Similar to activity-to-data dependency, the dependency between composite activity CA_{target} and other composite activities in the context of data flows are determined by CA_{target} generating *output_data* and delivering them to other composite activities, and vice versa. The degree of this dependency is reflected by the number of the composite activities that all the data of CA_{target} "traverse to" in their flows. This number is defined as activity-to-activity dependency in our approach.

The calculation of activity-to-activity dependency is similar to that of activity-to-data dependency except for each *input_data, initial_data, output_data*, and *final_data* of CA_{target}, their data-to-activity dependencies are counted to the activity-to-activity dependency of CA_{target}.

E. Labor Force Factor

Labor force factor is the metric to depict the degree of human actor being involved with business activities and processes, that is hardly to find in the previous approaches. This degree is reflected by the workload of human actors, similar to man-month, etc,. [7] of traditional concepts of software engineering. In our approach, *atomic_activity* is defined as the activity that is executed in a single unit of work and represented as one component of a composite activity. Accordingly, a new unit, namely *role-activity*, is developed to depict the degree of roles being involved with *atomic_activity*. One *role-activity* is the workload of one *role* being involved with one *atomic_activity*.

Labor force factor can be calculated in both dimensions of a composite activity and a business process. The labor force factor of a composite activity is calculated as multiplying the number of roles by the number of *atomic_activity* of the composite activity. Labor force factor of a business process is the accumulation of that of the composite activities involved with the same roles.

Labor force factor is also considered as "worst-case" metric. That indicates labor force factor is to provide the "possibly maximum" *role-activity*. We apply the methodologies of backtracking composite activities of calculating data-to-data dependency for calculating labor force factor of a business process. Similar to the calculation of data-to-data dependency, when multiple composite activities that are executed by identical roles are detected under the operators "*exclusive*" or "*N-exclusive*", the labor force factor of these composite activities is counted with the largest or the summation of the N largest labor force factor of the composite activities.

Data dependencies and labor force factor can be used to evaluate the quality of modeling business process with our methodologies in the domain of data flows and human actor involvement. Typically, a high data dependency or labor force factor implies a composite activity or data as a potential bottleneck, that might hinder the execution of a business process severely when encountering exceptions. In other words, data dependencies and labor force factor can be used to optimize business process modeling.

The evaluation of data dependencies are considered goal-directed because it is initiated with identifying the final states of objects, i.e., data and composite activities, that reflect business goals. These data dependencies can be evaluated by subsequently backtracking the objects in the workflow of business processes. Using our product and policy models, data dependencies introduced by our goal-directed approach helps access the quality of business processes, establish optimistic ones and provides reference for the evolving them.

V. A Use Case

In this section we verify our approach by applying it to a use case from [35] represented in Fig.2. This use case combines purchasing an apartment and applying for a housing loan. In this BPMN model, e_1 indicates the beginning of the business process while e_2 and e_3 indicate the end of the business process. g_5 is the gateway that reflects the divergence relation between multiple activities while the rest of g_is reflect the convergence relations.

We use our product and policy models to model the business process with representing the activities by composite activities. The policy model of this use case is listed as follows:

```
Process  apartment_purchasing_and_loan {
  assumption (apartment_purchase_application()
              and loan_application());
  while(apartment_purchase_application()) {
    down_payment();
    result((issue_certificate() and
           insurance()) cooperative
           (down_payment() and
           loan_agreement()));
    }
  while(loan_application()) {
    asset_evaluation();
    loan_agreement();
    result(((public_notification() and
           document_archive()) exclusive
           (insurance() and loan_agreement()))
           and ((issue_certificate() and
           insurance()) cooperative
           (down_payment() and
            loan_agreement())));
    }
 while(insurance()) {
    result((public_notification()
           and document_archive()) exclusive
           (insurance() and loan_agreement());
    }
    final(issue_certificate() and
          public_notification() and
          document_archive());
}
```

One example of the composite activities is stated as follows:

```
activity  issue_certificate{
  role applicant, government_employee,
       government_officer;
       \\issue_certificate_group
  initial_data  empty;
  input_data approved_loan_application_form,
             approved_apartment_application
             _form,
             certificate_of_loan,
             ready_to_issue_certificate;
  global_data government_ID;
  consumed_data ready_to_issue_certificate;
  atomic_activity <submit_materials>,
                  <verify_materials>,
                  <approve_certificate>,
                  <issue_certificate>;
  output_data empty;
  final_data approved_loan_application_form,
             approved_apartment_application
             _form,
             certificate_of_loan;
  }
```

The complete models of the use case is attached in the link https://webspace.utexas.edu/yz4466/www/use_case.pdf

A. Results and Analysis

TABLE I
DATA-BASED DEPENDENCY

data	col1	col2	col3	col4	col5
data-to-data dependency	18	22	19	2	8
data-to-activity dependency	5	7	5	1	2

Table I lists data-to-data dependency and data-to-activity dependency of some data in the use case, where col1 to col5 respectively represents approved_apartment_application_form, approved_loan_application_form, certificate_of_loan, estimated_loan_rate, and insurance_policy. The first three pieces of data significantly impact the whole business process due to their high weights of data-to-data and data-to-activity dependency.

Data-to-data and data-to-activity dependency together provide a scope for the degree of data flows being distributed on activities. Typically a high data-to-data dependency and a low data-to-activity dependency together indicate an unbalanced data flows over the entire business process. That means these data have a large impact on a few composite activities while a comparably small impact on others.

TABLE II
ACTIVITY-BASED DEPENDENCY

activity	col1	col2	col3	col4	col5
activity-to-data dependency	69	73	47	21	24
activity-to-activity dependency	21	23	17	8	9

Table II lists the activity-to-data dependency and activity-to-activity dependency of some composite activities in the use case, where col1 to col5 respectively represents public_notification, insurance, loan_agreement, apartment_purchasing_application, and loan_application. We

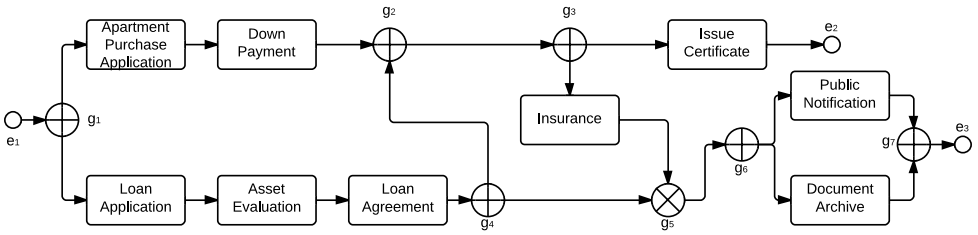

Fig. 2. A BPMN example

find when it approaches the later executed composite activities, their activity-to-activity and activity-to-data dependency tend to become higher. Particularly, it is because more data is generated and delivered along with the execution of the business process. In other words, the later executed part of the business process of this use case is more vulnerable to handle the exceptions from the previously executed part of the business process.

TABLE III
LABOR FORCE FACTOR

role	col1	col2	col3	col4	col5
labor force factor	23	5	17	12	10

Table III lists the labor force factor of some human actors in the use case, where col1 to col5 respectively represents applicant, insurance_employee, government_employee, government_officer, and bank_employee. In this use case, applicant needs to make the most effort to accomplish the goals of this business process.

Compared with a execution order of business activities delivered from the original BPMN model of this use case, our product and policy models provide more information including data flows and human actor involvement, and deliver a well-understood and flexible modeling techniques for users. Furthermore, along with the metrics, our modeling technique provides ways to verify the validity of business process models, and possible ways to optimize it.

B. Verification of Validity

One important property of a modeling language is to accurately determine the validity of a business process, that is, to ensure the execution of the modeled business process is logically reasonable. Our product and policy models enable viewing a business process from the dimensions of data dependencies and human actor involvement, that could be adopted as the perspectives for verifying the validity of the modeled business process.

In the BPMN model of this use case, the business process is in parallel initialized by the composite activities apartment_purchase_application and loan_application. That implies the two of the composite activities are not able to realize any data transactions in between. However, it is safe and sound to conduct loan_application with the official document, e.g., approved_appartment_application_form as an evidence for banks to verify the loan qualification of the applicant. While this can be easily deduced by applying data flows of our approach, it is difficult for the original BPMN model to deliver.

C. Optimization of Business Process

Our design of the metrics can be used for the optimization of business processes. Particularly, people want to maintain a low level of data dependencies and labor force factors. High data dependencies of any data or composite activity reflects a high possibility of exceptions, because it is significantly impacted by the large amount of other data or activities. In addition, a high labor force factor for a role represents the unbalanced distribution of the workload of the entire business process, that potentially leads to a waste of resources and low efficiency.

Some heuristics to optimize business processes under high data dependencies include restricting the data flow, e.g., by replacing some data types of one piece of data with other data. To reduce the labor force factor, one can design more parallel sub-processes where more roles can participate. This can be a future research interest.

VI. Discussion, Conclusions and Future Work

In this paper we propose a goal-directed technique for modeling a business process with composite activities and designing its relevant metrics. A business process is modeled by our product and policy models with composite activities. A composite activity embodies data types, human actors, and atomic activities as its components. Composite activities integrating with their associated components construct business processes with multiple domain of views, that is, execution order of composite activities, data flows, and human actor involvement. Moreover, These views potentially provide a basis for optimizing business process models. To explore this potential basis, we design a set of metrics of data dependencies and labor force factor. We also develop goal-directed algorithms, that is, modeling business goals as final states of those metrics and backtrack them for their evaluation. Data dependencies are comprised of four metrics to depict the degree of interrelations of data and composite activities, and their respective impact on the execution of business processes. Labor force factor provides the level of human actors being involved with the execution of a business

process and its associated composite activities. We design a use case to illustrate our modeling of business process and composite activities, evaluate the metrics, and explain the use of our modeling techniques.

Our product and policy models are designed in a both rigid and flexible manner. They are designed with rigorous syntax that enables an explicit execution order of composite activities, paths of data flows, and amount of human actor involvement, and derives abstract-syntax-tree-based algorithms of calculating the relevant metrics. On the other hand, this rigorous syntax helps provide flexibility for business process designers to use policy models, that is, execution blocks (i.e., *while()* loops) do not have to be placed in the context of business process with strict chronological order. The use of *while()* loops possibly separate the entire business processes into multiple individually functioned sub-processes that enable a possible enactment on specific *while()* loops, without interfering with others.

There is a variety of future work that can be implemented to improve the quality and usability of our approach. For instance, the principles of defining components of composite activities need to be well formed. Researchers of [14] propose that business process modeling techniques should be capable of presenting one or more of functional (i.e., what), behavioral (i.e., when and how), organizational (i.e., where and whom), and informational (i.e., data) perspectives. In this paper we include the functional, informational and part of behavioral and organizational perspectives. In the future, components of time and location can be taken into consideration for designing our product and policy models and thus provide more perspectives for optimization and evolution of business processes. One possibility to realize this is to combine our approach with other existing approaches, such as the graphics-based techniques [28] [32] [19].

This rigidity and flexibility provide a possibility to transform our product and policy models into executable languages for automation for business process modeling. By developing tools out of executable languages, the algorithms to calculate metrics can be automatically implemented to provide results that are the basis of the optimization and evolution of business processes, especially for large-scale business processes.

The metrics of our approach are coarse-grained at the current stage. Data dependencies and human actor involvement that are accumulated from that of each composite activity simply reflect the degree of overall data aggregation and workload of roles on business processes. They do not explicitly present the patterns of how these metrics are distributed among different composite activities. Developing fine-grained metrics to provide more explicit patterns of the objects can be interesting in future researches.

Moreover, our approach is established on deterministic business processes, that is, the results out of the execution of previous composite activities is deterministically either one composite activity or a combination of composite activities. While in reality, execution of business processes are possibly nondeterministic and provide diverse results. It can be challenging to take nondeterministic business processes into consideration to improve our modeling technique.

To evaluate the efficacy of our approach, we would like to conduct case studies in industry to gain the feedback from business actors and enterprise software engineers for its improvement in the future. Moreover, we would like to apply our approach in realistic environment, that is, use our approach to construct and analyze business processes of enterprises.

Finally, in Service-oriented Architecture, business processes, as a layer, interact with IT-applications through the orchestration of a service layer in between [10]. Business services that are designed based on the business activities impact and guide the modeling of the services of IT-applications. Our long term goals are to integrate modeling services with our current approaches of business processes and activities and to explore the interrelations between business services and IT-services.

REFERENCES

[1] Abstract syntax tree. http://en.wikipedia.org/wiki/Abstract_syntax_tree.

[2] Business process modeling. http://en.wikipedia.org/wiki/Business_process_modeling.

[3] Elvira Roln Aguilar, Francisco Ruiz, Flix Garcła, and Mario Piattini. Applying software metrics to evaluate business process models. *CLEI Electron. J.*, 9(1), 2006.

[4] R.S.Ruth Sara Aguilar-Saven. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, july 2004.

[5] C. Badica, M. Teodorescu, C. Spahiu, A. Badica, and C. Fox. Integrating role activity diagrams and hybrid idef for business process modeling using mda. In *Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC 2005. Seventh International Symposium on*, pages 4 pp.–, 2005.

[6] M. Bischof, O. Kopp, T. van Lessen, and F. Leymann. Bpelscript: A simplified script syntax for ws-bpel 2.0. In *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, pages 39–46, 2009.

[7] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, Boston, August 1995.

[8] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. Grail/kaos: An environment for goal-driven requirements engineering. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 612–613, New York, NY, USA, 1997. ACM.

[9] Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärik, and Jan Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, April 2011.

[10] Gregor Engels and Martin Assmann. Service-oriented enterprise architectures: Evolution of concepts and methods. In *EDOC*. IEEE Computer Society, 2008.

[11] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, Jan-Peter Richter, Markus VoB, and Johannes Willkomm. A method for engineering a true service-oriented architecture. In Jose Cordeiro and Joaquim Filipe, editors, *ICEIS (3-2)*, pages 272–281, 2008.

[12] C. Gerth, M. Luckey, J.M. Kuster, and G. Engels. Detection of semantically equivalent fragments for business process model change management. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 57–64, 2010.

[13] C. Gerth, M. Luckey, J.M. Kuster, and G. Engels. Precise mappings between business process models in versioning scenarios. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 218–225, 2011.

[14] George M. Giaglis. A taxonomy of business process modeling and information systems modeling techniques. *International Journal of Flexible Manufacturing Systems*, 13(2):209–228, apr 2001.

[15] Frank Goethals, Manu De Backer, Wilfried Lemahieu, Monique Snoeck, Jacques Vandenbulcke, and SAP-leerstoel Extended Enterprise Infrastructures. Identifying dependencies in business processes. In *Communication and Coordination in Business Processes (LAP-CCBP) Workshop, Kiruna, Sweden, June*, volume 22, 2005.

[16] Georg Grossmann, Michael Schrefl, and Markus Stumptner. Modelling inter-process dependencies with high-level business process modelling languages. In *Proceedings of the fifth Asia-Pacific conference on Conceptual Modelling - Volume 79*, APCCM '08, pages 89–102, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[17] Volker Gruhn and Ralf Laue. Complexity metrics for business process models. In *in: W. Abramowicz, H.C. Mayr (Eds.), 9th International Conference on Business Information Systems (BIS 2006), Lecture Notes in Informatics*, pages 1–12.

[18] A. Nico Habermann and Dewayne E. Perry. Well formed system composition. Technical Report CMU-CS-80-117, Department of Computer Science, Carnegie-Mellon University, March 1980.

[19] Mounira Harzallah, Giuseppe Berio, and Andreas L. Opdahl. Incorporating idef3 into the unified enterprise modelling language. In *EDOCW*, pages 133–140. IEEE Computer Society, 2007.

[20] Dave Ings, Luc Clment, Dieter Konig, Vinkesh Mehta, Ralf Mueller, Ravi Rangaswamy, Michael Rowley, and Ivana Trickovic. Ws-bpel extension for people (bpel4people) specification version 1.1. OASIS Committee Specification, August 2010.

[21] Jerry L. Jones. *Structured programming logic - a flowcharting approach.* Prentice Hall, 1985.

[22] Finn Jorgensen. Overview of function modelling - idef0. In Heimo H. Adelsberger, Jiri Lazansky, and Vladimir Marik, editors, *Information Management in Computer Integrated Manufacturing*, volume 973 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 1995.

[23] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006.

[24] Rong Liu, Anil Nigam, Zhe Shan, and Frederick Y. Wu. Uniform modeling of resources and business processes using business entities. In *IEEE SCC*, pages 693–700, 2011.

[25] Hafedh Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi. Business process modeling languages: Sorting through the alphabet soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, December 2010.

[26] Hanspeter Mossenbock. *How to Build Abstract Syntax Trees with Coco/R*. 2011.

[27] Object Management Group (OMG). Business process modeling language. Technical report, jan 2003.

[28] Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Technical report, jan 2011.

[29] Carla Marques Pereira and Pedro Sousa. A method to define an enterprise architecture using the zachman framework. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 1366–1371, New York, NY, USA, 2004. ACM.

[30] D.E. Perry. Policy-directed coordination and cooperation. In *Software Process Workshop, 1991. Communication and Coordination in the Software Process., Proceedings of the 7th International*, pages 111–113, 1991.

[31] Dewayne E. Perry. Enactment control in interact/intermediate. In Brian Warboys, editor, *Software Process Technology, Third European Workshop, EWSPT 94, Villard de Lans, France, February 7-9, 1994, Proceedings*, volume 772 of *Lecture Notes in Computer Science*, pages 107–113. Springer, 1994.

[32] Wolfgang Reisig. *A primer in Petri net design.* Springer Compass International. Springer, 1992.

[33] T.J. Schriber. *Fundamentals of flowcharting*. Wiley, 1969.

[34] Stefan Schulte, Kay Kadner, Nicolas Repp, and Ralf Steinmetz. Applied service engineering for single services and corresponding service landscapes. In Robert C. Nickerson and Ramesh Sharda, editors, *AMCIS*, page 472. Association for Information Systems, 2009.

[35] Yutian Sun and Jianwen Su. Computing degree of parallelism for bpmn processes. In *Proceedings of the 9th international conference on Service-Oriented Computing*, ICSOC'11, pages 1–15, Berlin, Heidelberg, 2011. Springer-Verlag.

[36] Irene T. P. Vanderfeesten, Jorge Cardoso, and Hajo A. Reijers. A weighted coupling metric for business process models. In Johann Eder, Stein L. Tomassen, Andreas L. Opdahl, and Guttorm Sindre, editors, *CAiSE Forum*, volume 247 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[37] Zan Xiao, Donggang Cao, Chao You, and Hong Mei. Towards a constraint-based framework for dynamic business process adaptation. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 685–692, 2011.

[38] Sen Zeng, Shuangxi Huang, and Yushun Fan. Service-oriented business process modeling and performance evaluation based on ahp and simulation. In *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 469–476, 2007.