A Scenario-View Based Approach to Analyze External Behavior of Web Services for Supporting Mediated Service Interactions

Zhangbing Zhou, Sami Bhiri, Lei Shu, Laurentiu Vasiliu and Manfred Hauswirth
Digital Enterprise Research Institute, National University of Ireland at Galway, Ireland
{firstname.lastname}@deri.org

Kaizhu Huang
Computer Science and Engineering Dept, The Chinese University of Hong Kong, Hong Kong
kzhuang@cse.cuhk.edu.hk

Abstract

Web service interactions have triggered the initiative to identify and solve mismatches from a behavioral aspect. Current approaches are limited since they mainly focus on control-flow but largely ignore data-flow. In this paper, we propose an approach to automatically generate scenarios and views for describing external behavior of Web services, i.e. the public process, considering both control-flow and data-flow. We define a scenario as a set of complete execution paths for a public process. Data dependencies are presented as a dependency graph, which is optimized into a minimal dependency graph. Then, a view is generated to describe a scenario for analysis purposes, and external behavior of a Web service is described as a finite set of views. Our approach is very useful for service modelers and users to better understand the external behavior of Web services, to identify and solve mismatches from a behavioral aspect, and thus to facilitate Web service interactions.

1. Introduction

A Web service interaction can be described as a flow of messages, which contain a set of data, exchanged among Web services. Because of the inherent *autonomy* and *heterogeneity* of Web services, messages are often different in *format* and *granularity*, and public processes [7] are often diverse in activities and messages in terms of *form* and *sequence*. Thus, it is difficult, if not impossible, to find two Web services that are completely compatible [3] from either functional, or behavioral, or both aspects, and a Web service interaction is normally carried out with the help of data or process mediators [7]. A service interaction with the help of mediators is called as a *mediated service interaction*. Since many methods, e.g. [6, 8, 12], have analyzed Web services from a functional aspect, we focus on a behavioral aspect. Since the internal implementation of Web services does not contribute to the analysis of service interactions, we focus on external behavior of Web services, which is typically described by a public process from control-flow and data-flow aspects [7]. Current methods for facilitating service interactions, e.g. [20, 21, 23], focus on control-flow and largely ignore data-flow, and are limited to identify and solve behavioral mismatches among Web services. They are insufficient to support mediated service interactions. Thus, from the perspective of supporting mediated service interactions, what is external behavior of Web services considering both control-flow and data-flow?

Current approaches for analyzing external behavior of Web services include: *Control-flow based* methods [9, 18] focus on control-flow and largely ignore data-flow. *Dependency based* methods [13, 22] analyze dependencies from data, control and other aspects. *View based* methods [4, 16, 24] investigate the relation of private and public processes from a control-flow aspect. Thus, current approaches focus on either a control-flow or a data-flow aspect and are limited to answer our question. An improved approach is necessary to analyze external behavior of Web services for supporting mediated service interactions.

To address these problems, we present a novel approach including: (1) We generate all *scenarios* for a public process. A *scenario* is a set of complete execution paths [3] for a public process. (2) Data dependencies are represented by *a data dependency graph*, which presents a finite set of mandatory or optional data dependencies in a public process or a *scenario*. However, a data dependency is redundant and can be safely removed if it is implicitly specified by other data dependencies. *A minimal data dependency graph* is generated where there are no redundant data dependencies. (3) We propose three reduction rules to identify and remove unnecessary control dependencies specified by

sequence, *And* and *Loop* blocks in a *scenario*. (4) With the help of *a minimal data dependency graph*, we generate a *view* for a *scenario* by applying three reduction rules recursively. A public process can be described as a finite set of scenarios. A scenario has a corresponding view representing this scenario for analysis purposes. External behavior of a Web service can be described as a finite set of views.

As far as we know, our *scenario-view* based approach is the first study to analyze external behavior of Web services considering both control-flow and data-flow. Its immediate benefits include: (1) it can help service modelers and users for a better understanding on external behavior of Web services, and (2) it can provide valuable instructions for facilitating Web service creation or evolution. In addition, it is the base for (3) checking compatibility of Web services from a behavioral aspect, (4) identifying behavioral mismatches among Web services and generating process mediators, and (5) facilitating mediated service interactions.

Here is the outline of this paper. A motivating example is presented in Section 2. A definition and graphical notations for a public process are shown in Section 3. Afterwards, *scenarios* are generated for a public process, data dependencies are discussed, and a *view* is generated for a *scenario* in Section 4, 5 and 6. Finally, related work is discussed and a conclusion is made in Section 7 and 8.

2. A motivating example

Figure 1-a and 1-b show the public processes of two Web services for a toy shop and a requestor, which want to interact for achieving a goal: *buying toys*. The public processes can be coded in BPEL4WS [1], and a definition of public processes are presented in Section 3. The pricing strategy of the toy shop is flexible: *a discount is applied at the children day if the requestor is a child, or a normal price otherwise*. Thus, the toy shop expects toy items and customer information before deciding the price although customer information is optional for pricing. Due to the privacy concern, the requestor expects the price firstly. If the price is acceptable, he/she pays and provides customer information for delivery purposes. We assume that no heterogeneity exists at a data level, since it is out of the focus of this paper and much research has been conducted at this aspect [14].

From a control-flow aspect, since the toy shop expects customer information and then provides the price, and the requestor expects the price and then provides customer information, they cannot carry out a direct service interaction.

Current approaches for checking compatibility of workflows, e.g. [3, 11, 17, 21], are based on control-flow and assume that the toy shop and the requestor are incompatible. Current approaches for process mediation, e.g. [2, 5, 15], aim to identify and solve behavioral mismatches from a control-flow aspect. They regard this kind of mismatches as unresolvable [5, 7] and this interaction should fail.

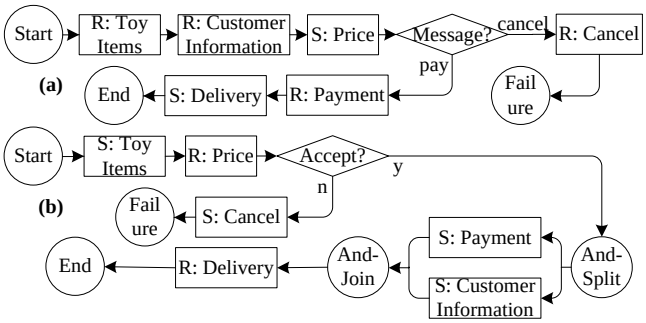

Figure 1. Public processes for a toy shop and a requestor Web services

However, if there is a process mediator in the middle considering both control-flow and data-flow, the toy shop and the requestor can carry out an interaction successfully. Since the pricing strategy of the toy shop is flexible, it can give a normal price if it is notified by the process mediator that the requestor will not provide customer information before receiving the price. Thus, the messages exchanged between the toy shop and the requestor can form a mediated service interaction leading from their *Start* nodes to their *End* nodes. Figure 2 shows how this mediated service interaction is carried out. For simplicity, the activities which do not contribute to this interaction are not presented.

This example indicates that current approaches, which check behavioral compatibility at a control-flow level, can support direct service interactions only. Current process mediators aim to solve behavioral mismatches at a control-flow level, and are limited to support mediated service interactions. Their major shortcomings are:

- Generally, only a part of activities in a public process will involve in a given interaction. A concept: *scenario*, is introduced to represent a complete execution path for a public process. A further discussion on how to generate all *scenarios* for a public process is presented in Section 4.
- Current approaches for process mediation and behavioral compatibility are control-flow based, while data-flow is largely ignored although data-flow is implicitly specified in a public process. To apply data-flow for supporting mediated service interactions, a concept: *data dependency graph*, is introduced to represent data-flow of a public process or a *scenario*. A further discussion on data-flow is shown in Section 5.
- Current approaches assume that the activities in a public process or a *scenario* must be executed following the order specified by its control-flow. In real applications, the execution order of some activities may be

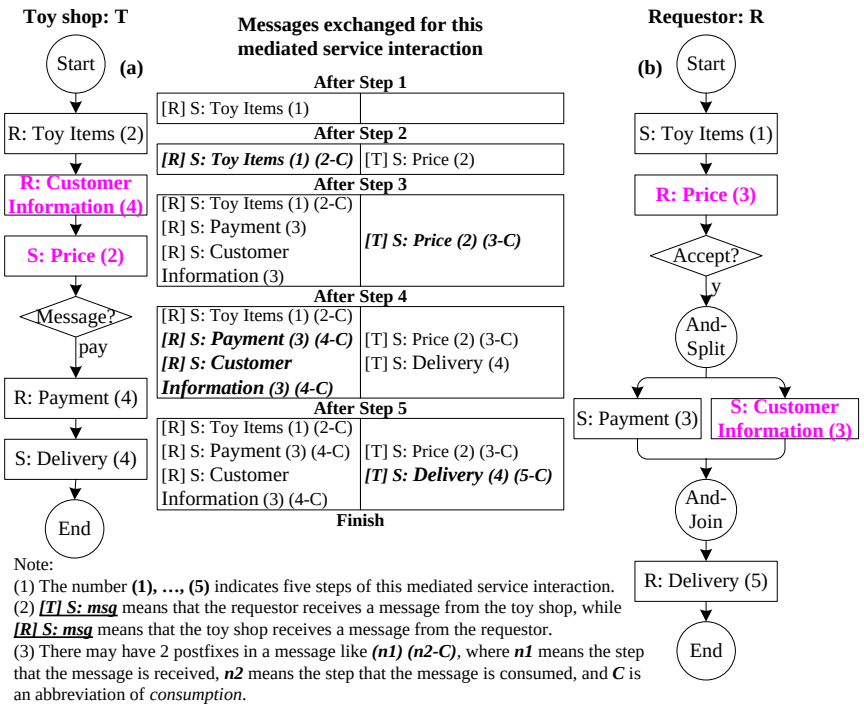

Figure 2. A successful mediated service interaction between the toy shop and the requestor

changed if there are no mandatary data dependencies among them. A concept: *checkpoint*, is introduced to represent a finite set of contiguous activities that can be executed in any order, and another concept: *view*, for a sequence of *checkpoints* in a *scenario*. A further discussion on how to generate a *view* for a *scenario* is presented in Section 6.

3. Modeling a public process

Below we give a definition for a public process in which messages and guard functions are the first-class elements.

Definition 1 (Public Process). A public process p is the five-tuple *(MSG, ACT, CNT, GRD, ARC)*, where $MSG=\{msg\}$ is is a finite set of messages, $ACT=\{act\}$ is a finite set of activities for sending or receiving messages, $CNT=\{Start, Failure, End, Xor\_Split, Xor\_Join, And\_Split, And\_Join\}$ are control elements, $GRD=\{grd\}$ a finite set of guard functions, and $ARC=\{arc\}$ a finite set of arcs that connect activities and control elements.

We follow [3] for the function *Polarity(msg)* to specify whether a message is received if *Polarity(msg)=R* or sent if *Polarity(msg)=S*. For simplicity but without loss of generality, we assume that a message contains one data. Both activities and control elements are the nodes in a public process. We model a public process as a structured workflow [10]. Figure 3-a to 3-f shows six basic graphical notations for modeling a public process. These notations are supported by *JGraphPad* [1], based on which our prototype is implemented. Six ordering structures: *sequence*, *And-Split/Join*, *Xor-Split/Join* and *Loop* are defined by *WfMC* [2]. *And-Split/Join* and *Xor-Split/Join* can be modeled directly by Figure 3-e and 3-f. *Sequence* and *Loop* are complex structures and can be modeled by our basic notations as shown in Figure 3-g and 3-h.

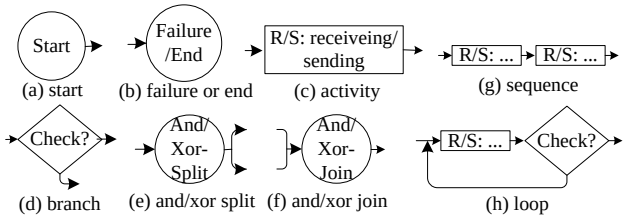

Figure 3. Graphical notations for modeling a public process

4. Generating scenarios for a public process

A *Xor* block in a public process is a sub-process delimited by a *Xor-Split* with its *Xor-Join*. Only one path can

[1] http://www.jgraph.com/jgraphpad.html.
[2] http://www.wfmc.org/standards/docs.htm.

be enabled in a given execution depending on the status of guard functions. *Branch* control elements are used for two purposes: *specifying an exclusive relation* or *modeling a loop*. In this section, only *Branch* control elements that contribute to exclusive relations are considered. Similarly, after a *Branch* control element, only one of its branches can be enabled in a given execution. Exclusive paths or branches are called as *alternative paths* to each other. A finite set of scenarios can be generated for a public process where each scenario includes only one alternative path in a *Xor* block or after a *Branch* control element.

Regarding to other control elements in a public process, such as *And* blocks and *Branch* control elements for modeling loops, since either none or all of their activities will be executed in a given execution, they are inherited by one or several scenarios. However, these activities may be executed following different orders in different executions. This suggests that a scenario may include several complete execution paths.

Definition 2 (Scenario). A scenario *sce* is a set of complete execution paths for a public process p, which is defined by the five-tuple $(MSG_{sce}, ACT_{sce}, CNT_{sce}, GRD_{sce}, ARC_{sce})$ generated from those of p. For any node in a scenario except *Start*, *Failure/End*, *And_Split*, *And_Join*, and *Branch control elements for modeling loops*, it has only one entering and one leaving edge.

There are two scenarios for the toy shop service, and Figure 2-a shows one of them. There are two scenarios for the requestor service, and Figure 2-b shows one of them.

5. Optimizing data dependencies into a minimal data dependency graph

Data dependencies of a public process can be extracted from its specification in terms of a BPEL process [13]. We record data dependencies as mandatory or optional. A mandatory data dependency means that it must be held during execution phases. An example is "*R: Toy Items*" to "*S: Price*" in Figure 2-a. An optional data dependency means that it may, or may not, be held during execution phases. An example is "*R: Customer Information*" to "*S: Price*" in Figure 2-a. All data dependencies in a public process form a *directed*, *connected*, and *acyclic* graph, where nodes are the data and directed links indicate the dependency relations among data.

Definition 3 (Data Dependency Graph). A data dependency graph *dg* for a public process *(MSG, ACT, CNT, GRD, ARC)* is a *directed*, *connected* and *acyclic* graph, which is defined by the two-tuple $(DATA_{dg}, DE_{dg})$, where $DATA_{dg} = \{data\}$ is a finite set of data generated from *MSG*, which are the nodes in this graph. $DE_{dg} = DE_{dg}^{(M)} \cup DE_{dg}^{(O)}$ a finite set of edges, which are the direct links in this graph specifying the dependency relations among data. $DE_{dg}^{(M)}$ is for mandatory dependencies, and $DE_{dg}^{(O)}$ is for optional dependencies.

A data dependency graph defines a finite set of partial-order relations for data. These relations are *asymmetric*, *irreflexive* and *transitive*. One data is regarded as *mandatorily (or optionally) dependent* on another data if (1) a direct link, which specifies a *mandatory (or optional)* dependency relation, connects them (called *directly dependent*), or (2) several direct links, all of which are *mandatory (or optional)* dependency relations, form a path leading from one data to another (called *indirectly dependent*). It is possible that a data is both *mandatory* and *optional* dependent on another data. This suggests that a data dependency graph can be represented as a finite set of *mandatorily* or *optionally dependent* relations. A data dependency graph is called *functionally equivalent* to another data dependency graph if any *mandatory* or *optional dependency* relation in one graph can exist in another graph directly or indirectly. In a data dependency graph, some dependencies may impose same *dependent* relations. If any *mandatory* (or *optional*) dependency relation cannot be specified by other *mandatory* (or *optional*) dependency relations, a data dependency graph is called *minimal*.

Definition 4 (Minimal Data Dependency Graph). A minimal data dependency graph dg_{min}: $(DATA_{min}, DE_{min})$ is generated from a data dependency graph *(DATA, DE)*, where $DATA_{min}$ = *DATA*, $DE_{min} \subseteq$ *DE*, and $(DATA_{min}, DE_{min})$ is *functionally equivalent* to *(DATA, DE)*, but $\forall$ *de* $\in DE_{min}$: $(DATA_{min}, (DE_{min} - \{de\}))$ is not *functionally equivalent* to *(DATA, DE)*.

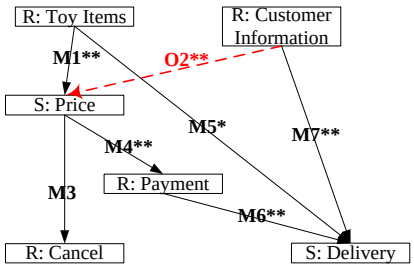

Figure 4. (Minimal) data dependency graphs for the toy shop service and its scenario

Figure 4 shows a data dependency graph for the toy shop, where the edge *O2* shows an optional dependency. A data dependency graph of a scenario is generated from that of a public process by removing (1) data that are not related to this scenario and (2) edges if an edge connects to any data that is not related to this scenario. In Figure 4, a data dependency graph for a scenario shown in Figure 2-a is marked with one or two $*$, and its minimal data dependency graph

is marked with $**$.

6. Generating a view for a scenario

In this section, we firstly introduce the concepts of *checkpoint* and *view*. Then, we propose three reduction rules to identify and remove redundant control dependencies in a scenario, and thus to generate *checkpoints*. Finally, we propose our *genView* algorithm to generate a *view* for a scenario using reduction rules recursively with the help of a minimal data dependency graph.

6.1. What is a view?

Before introducing the concept of *view*, we present a related concept: *checkpoint*.

Definition 5 (Checkpoint). A checkpoint *cp* includes a finite set of contiguous activities in a scenario in which data dependencies among them are not mandatory. A checkpoint is defined by the four-tuple *(label, ACT, DATA, GRD)*, where *label* for its label. *ACT* for activities, *DATA* for required data, and *GRD* for guard functions, are generated from those of the scenario.

Required data can be generated with the help of the minimal data dependency graph of a scenario. For example, *M1* in Figure 4 suggests that the data: *"R: Toy Items"*, is required for *"S: Price"*. However, since the data dependency *O2* for *"R: Customer Information"* and *"S: Price"* is optional, *"R: Customer Information"* is not required for *"S: Price"*. Table 1 [3] shows lists six checkpoints for the scenario shown in Figure 2-a.

label	ACT	DATA
cp_0	*Start*	
cp1	*Receive Toy Items* *Receive Cust. Info.*	
cp2	*Pick Price*	*data(Receive Toy Items)*
cp3	*Receive Payment*	*data(Pick Price)*
cp4	*Invoke Delivery*	*data(Receive Cust. Info.)* *data(Receive Payment)*
cp_f	*End*	

Table 1. Checkpoints for the scenario shown in Figure 2-a

Definition 6 (View for a Scenario). A view *vw* for a scenario $(MSG_{sce}, ACT_{sce}, CNT_{sce}, GRD_{sce}, ARC_{sce})$ is the five-tuple $(MSG_{vw}, CP, cp_0, cp_f, DE_{vw})$, where $MSG_{vw} = MSG_{sce}$, $CP=\{cp\}$ is a finite set of checkpoints, cp_0 is the initial checkpoint, and cp_f is the final one, while $DE_{vw} = \{de\}$ is a finite set of direct links connecting checkpoints to specify data dependencies among them.

A checkpoint is a point in a view in which the verification is conducted for analysis purposes.

[3] Guard functions are not presented in this table.

6.2. Reduction rules

Control-flow structures of a scenario specify execution orders of activities. However, the execution of some activities may not follow these orders. An example is *"R: Toy Items"* and *"R: Customer Information"* in Figure 2-a since they are not data dependent on each other. Thus, in this section, three reduction rules are presented to identify and remove redundant control dependencies specified by *Sequence*, *And* and *Loop* blocks. Since only one alternate path is enabled for a *Xor* block and a *Branch* control element that specifies an exclusive relation, they are functionally equivalent to *Sequences*.

We follow previous report [19] for the function: *fold*, for replacing several contiguous nodes by a single node. Data dependencies and guard functions of these nodes are inherited by the folded node. We call a folded node *a virtual node*, which is shown as a rectangle with dashed lines afterwards. Three reduction rules introduced in this section can be applied to a scenario recursively until no (virtual) nodes can be folded into a virtual node anymore.

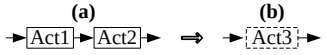

Figure 5. Reduction rules for Sequences

Rule 1 (Sequence). A *sequence* of activities are folded into a virtual node if data dependencies among them are not mandatary.

This rule is shown by Figure 5. *Sequence* in Figure 5-a indicates that *Act1* should be executed before *Act2*. It is actually a control dependency: *Act1* should happen before *Act2*. However, this control dependency is inappropriate if *Act2* is not data dependent on *Act1*. An example for this rule is the activities *"R: Toy Items"* and *"R: Customer Information"* in Figure 2-a.

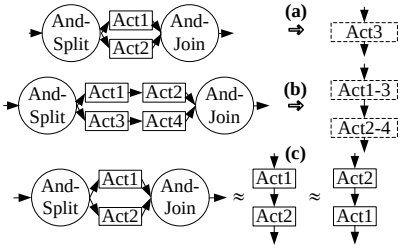

Figure 6. Reduction rules for And blocks

Rule 2 (And). An *And* block with its *And_Split* and *And_Join* is folded into (1) a virtual node if all activities in each path can be folded into a virtual node or (2) a sequence of virtual nodes otherwise.

An *And* block indicates that all its paths should be executed in parallel. However, there are no control and data dependencies among activities of different paths. This means that an *And* block can be converted into a *sequence* of activities as shown in Figure 6-c.

The rule for *And* block is presented by Figure 6-a and 6-b. Figure 6-a shows a case that, for any path in an *And* block, all nodes can be folded into one virtual node. This *And* block can be folded into a single virtual node. An example is the And block: *"S: Payment"* and *"S: Customer Information"*, shown in Figure 2-b.

Another case is shown in Figure 6-b. At least one of its paths has more than one activity with mandatary data dependencies among them. Thus, at least one path cannot be folded into a virtual node. This *And* block can be translated into a *sequence* of virtual nodes.

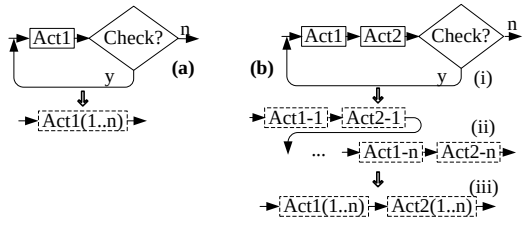

Figure 7. Reduction rules for Loop blocks

Rule 3 (Loop). A *Loop* block is folded into (1) a virtual node if the *Loop* body can be folded into a virtual node or (2) a sequence of virtual nodes otherwise. Guard functions are defined in the last virtual node for specifying the exit conditions of the *Loop* block. There are possibly multiple instances for a virtual node during execution phases.

This rule is shown by Figure 7. A *Loop* block iterates over one or several nodes until its exit conditions are satisfied, but it doesn't iterate forever in real situations. As suggested by [21], a *Loop* block can be simulated as a *sequence* of at most *N* repetitions of the *Loop* body, where *N* depends on a given execution. As shown in Figure 7-a, if all nodes in a *Loop* body can be folded into one virtual node, this *Loop* block can be simulated as a single virtual node. Guard functions are defined in this virtual node to specify the exit conditions of this *Loop* block.

Another case is shown in Figure 7-b. If data dependencies are mandatary and thus a *Loop* body cannot be folded into one virtual node, the *Loop* block can only be simulated as a *sequence* of virtual nodes as shown by Figure 7-b-(ii). Since *n* is nondeterministic for analysis purposes, we transfer the *sequence* from Figure 7-b-(ii) to Figure 7-b-(iii) for facilitating the analysis of the *Loop* block. However, the *sequences* shown by 7-b-(ii) and Figure 7-b-(iii) are not semantically equivalent. Figure 7-b-(ii) specifies dependencies for (1) *Act2-i* and *Act1-(i+1)*, (2) *Act1-i* and *Act1-(i+1)*, and (3) *Act2-i* and *Act2-(i+1)*. In Figure 7-b-(iii), the first kind of dependencies is preserved, but the latter two are lost. However, the latter two are implicitly satisfied in Figure 7-b-(iii) if the exit conditions of the *Loop* block are satisfied. Thus, it is reasonable to analyze the *Loop* block following the *sequence* as shown in Figure 7-b-(iii).

6.3. Generating a view for a scenario

We propose our *genView* algorithm to generate a view for a scenario, which applies three reduction rules upon a scenario recursively until no (virtual) nodes can be folded into a virtual node anymore (line 3-9). Afterwards, checkpoints are generated (line 11) and a view is derived (line 10-14). The time complexity of this algorithm is $O(n^2)$, where *n* is the number of nodes in a scenario, because in the worst case, two (virtual) nodes can be folded into one virtual node at each iteration. This procedure repeats until there are three nodes left, in which one for the initial and another for the final checkpoints. A view for a scenario includes a sequence of checkpoints leading from its initial checkpoint to its final checkpoint.

7. Related work

Related work can be categorized into three types: *control-flow*, *dependency*, and *view* based analysis.

Control-flow based analysis. In [18], the authors recognized that different parts in a process model are often not equally important. Thus, they checked process similarity in terms of *typical behaviors*, which are *typical* executions extracted from an event log. This method does not fit for new processes where no event logs can be used for identifying their *typical behaviors*. Another work [9] presented a framework for supporting workflow intelligence and quality improvement using *workcases* generated from a workflow, and *execution cases* discovered from event logs. The *workcase* is a concept similar to our *scenario*. In Summary, control-flow is the focus, and data dependencies are largely ignored. These approaches contribute to the generation of scenarios for a workflow only.

Dependency based analysis. Dependency, which specifies an ordering and synchronization relation between activities, is an important and well-studied method for program analysis and optimization. However, few studies have applied dependency for workflow analysis. One work is [22] which proposed *data*, *control*, *service* and *cooperation* dependencies for describing constraints in a business process. All dependencies are optimized into a minimal dependency

Algorithm 1: ***genView***

in :
- *sce* : $(MSG_{sce}, ACT_{sce}, CNT_{sce}, GRD_{sce}, ARC_{sce})$: A scenario
- dg_{min} : $(DATA_{min}, DE^{(M)}_{min} \cup DE^{(O)}_{min})$ for *sce*

out : $view : (MSG_{vw}, CP, cp_0, cp_f, DE_{vw})$ for *sce*

data: - *ND*: a set of nodes in *sce* which form a *Sequence*, a *Flow* block or a *While* block
- ND_{Seq}: a sequence of nodes

func:
- $foldSce(ND, sce)$: to fold *ND* in *sce* into one or a *Sequence* of nodes: ND_{Seq}, using reduction rules
- $isSeq(ND, sce)$, $isFlow\ (ND,\ sce)$, $isWhile($ *ND*, $sce)$: a boolean function to check if a ***Sequence***, ***Flow*** or ***While*** block satisfies reduction rules
- $getData(nd, sce)$: to get data from the message related to a node *nd*
- $foldMinDG(getData(nd, sce), dg_{min})$: to fold a dg_{min} by replacing several data generated from *getData(nd, sce)* as a data set
- $genReqData(getData(nd, sce), dg_{min})$: to generate required data for a node *nd*
- $genCP(nd, \{data\})$: to generate a checkpoint from a node *nd* with required data $\{data\}$
- $insCP2View(cp, view)$: to insert a checkpoint *cp* into a view *view*

1 **begin**
2 $MSG_{vw} \leftarrow MSG_{sce}$; $sce_{cur} \leftarrow sce$; $do \leftarrow false$
3 **while** do = *false* **do**
4 $do \leftarrow true$
5 **if** $ND \subset ACT_{sce_{cur}} \cup CNT_{sce_{cur}}$ **and** ($isSeq(ND, sce_{cur})$ = $true$ **or** $isFlow(ND, sce_{cur})$ = $true$ **or** $isWhile(ND, sce_{cur})$ = $true$) **then**
6 ND_{Seq}, $sce_{tmp} \leftarrow foldSce(ND, sce_{cur})$
7 **for** $nd_i \in ND_{Seq}$ **do**
8 $dg_{min} \leftarrow foldMinDG($ $getData(nd_i, sce_{tmp}), dg_{min})$
9 $sce_{cur} \leftarrow sce_{tmp}$; $do \leftarrow false$
10 **for** $nd_{cur} \in ACT_{sce_{cur}} \cup CNT_{sce_{cur}}$ **do**
11 $cp_{tmp} \leftarrow genCP(nd_{cur}, genReqData($ $getData(nd_{cur}, sce_{cur}), dg_{min}))$
12 $view \leftarrow insCP2View(cp_{tmp}, view)$
13 $cp_0 \leftarrow cp_{tmp} \Leftarrow nd_{cur}$ is *Start* node of sce_{cur}
14 $cp_f \leftarrow cp_{tmp} \Leftarrow nd_{cur}$ is final node of sce_{cur}
15 **end**

set for supporting high concurrency and minimal maintenance cost. Control dependencies in this work are functionally equivalent to our guard functions. [13] proposed to extract data dependencies from BPEL processes. These work benefits much to our analysis of dependency graphs. However, the authors did not consider possible conflicts among different kinds of dependencies. For the sake of *autonomy* and *privacy*, *service* and *cooperation* dependencies may not be available. This effort may unfit for Web service domain.

View based analysis. A process view is an abstract process to support *interaction*, *security*, and *privacy*. A view [4] was proposed to support *cross-organizational* workflow execution. Workflows and resources can be partially visible to potential partners and thus provide a powerful method for *inter-organizational* workflow configuration. This view is similar to the public process. Based on the *tracking structure* of *a relative workflow model*, a view [24] was proposed to perform *workflow tracking* across organizational boundaries. Different views can be generated for different organizations based on a pre-existing *collaborative business process*. This work follows a top-down approach and is not suitable to Web service domain. Another view [16] aimed to selectively hide the details of private processes, to support state-oriented communication, and to facilitate *cross-organizational* workflow execution. Two types of interactions are supported: unmediated and mediated. A mediator presented in this paper is not designed to solve possible mismatches, but to route messages among processes. It actually acts like a gateway or a message broker. Therefore, a view based approach is promising to support *cross-organizational* workflow cooperation. However, they investigate more to the relation of public and private processes, rather than to what a public process is.

Taken together, current approaches are helpful for answering our question to some extent but cannot provide a complete solution. To the best of our knowledge, our study is the first effort to integrate these three aspects to analyze external behavior of Web services for supporting mediated service interactions.

8. Conclusion and future work

We have discussed that current approaches of checking behavioral compatibility and process mediation are limited to support mediated service interactions, because they mainly focus on control-flow but largely ignore data-flow. We have proposed a scenario-view based approach to analyze external behavior of Web services considering both control-flow and data-flow. A scenario is a set of complete execution paths for a public process. Data dependencies of a public process or a scenario are presented as a data dependency graph, which is optimized into a minimal data dependency graph. A view, which is a sequence of checkpoints, is generated to represent a scenario for analysis purposes. External behavior of a Web service can be described as a finite set of views for supporting mediated service interactions.

This study is our first step towards the support of mediated service interactions, where we check compatibility of Web services from a behavioral aspect. In this direction, we are taking this work further to identify behavioral mismatches and to generate process mediators for solving these behavioral mismatches among Web services, and thus to facilitate mediated service interactions.

Acknowledgments

The work presented in this paper was supported (in part) by the EU funded TripCom Specific Targeted Research Project under Grant No. FP6-027324, and (in part) by the Lion project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I131.

We thank Brahmananda Sapkota, Hak Lae Kim, Ke Ning and Xia Wang for their valuable comments.

References

[1] S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, S. Commerce, C. K. Liu, S. Thatte, P. Yendluri, and A. Yiu. Web services business process execution language version 2.0. Technical report, OASIS. Available at http://www.oasis-open.org/apps/org/workgroup/wsbpel/, 2005.

[2] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani. Developing adapters for web services integration. Proceedings of International Conference. of Advanced Information System Engineering (CAiSE'05), 2005.

[3] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *Data and Knowledge Engineering*, 58(3):327–357, 2006.

[4] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data and Knowledge Engineering*, (2):139–173, 2006.

[5] E. Cimpian and A. Mocan. Wsmx process mediation based on choreographies. Proceedings of the first International Workshop on Web Service Choreography and Orchestration for Business Process Management at the BPM 2005, 2005.

[6] J. de Bruijn, C. Bussler, and J. Domingue. D2v1.3. web service modeling ontology (wsmo). In WSMO Final Draft 21 October 2006. Available at http://www.wsmo.org/TR/d2/v1.3/, 2006.

[7] D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, pages 113–137, 2002.

[8] U. Keller, H. Lausen, and M. Stollberg. On the semantics of functional descriptions of web services. Proceedings of the 3rd European Semantic Web Conference (ESWC'06). Budva, Montenegro, 2006.

[9] K.-H. Kim. Control-path oriented workflow intelligence analysis on enterprizeworkflow grids. Proceedings of the 1st International Conference on Semantics, Knowledge and Grid (SKG'05), 2005.

[10] R. Liu and A. Kumar. An analysis and taxonomy of unstructured workflows. Proceedings of International Conference on Business Process Management (BPM'05), 2005.

[11] A. Martens. Analyzing web service based business processes. Proceedings of International Conference on Fundamental Approaches to Software Engineering (FASE'05), Part of the 2005 European Joint Conferences on Theory and Practice of Software (ETAPS'05), 2005.

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services. Draft available at http://www.ai.sri.com/daml/services/owl-s/1.2/overview/, 2006.

[13] S. Moser, A. Martens, K. Gorlach, W. Amme, and A. Godlinski. Advanced verification of distributed ws-bpel business processes incorporating cssa-based data flow analysis. Proceedings in IEEE International Conference on Services Computing (SCC'07), 2007.

[14] M. Nagarajan, K. Verma, A. P. Sheth, J. Miller, and J. Lathem. Semantic interoperability of web services - challenges and experiences. Proceedings of the 4th IEEE International Conference on Web Services (ICWS'06), 2006.

[15] H. R. M. Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. Proceedings of the 16th international conference on International World Wide Web Conference (WWW'07), 2007.

[16] K. A. Schulz and M. E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data and Knowledge Engineering. Special issue: Contract-driven coordination and collaboration in the internet context*, 51(1):109–147, 2004.

[17] W. M. P. van der Aalst. Inheritance of interorganizational workflows to enable business-to-business e-commerce. *Electronic Commerce Research*, 2(3):195–231, 2002.

[18] W. M. P. van der Aalst, A. A. de Medeiros, and A. Weijters. Process equivalence: Comparing two process models based on observed behavior. Proceedings of International Conference on Business Process Management (BPM'06), 2006.

[19] W. M. P. van der Aalst and K. B. Lassen. Translating unstructured workflow processes to readable bpel: Theory and implementation. *Information and Software Technology*, 50(3):131–159, 2008.

[20] W. M. P. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. Proceedings of The 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), 2001.

[21] A. Wombacher. *Decentralized establishment of consistent, multi-lateral collaborations*. PhD thesis, Facultiy of Informatics, Technical University Darmstad, 2005.

[22] Q. Wu, C. Pul, A. Sahai, and R. Barga. Categorization and optimization of synchronization dependencies in business processes. Proceedings in IEEE 23rd International Conference on Data Engineering (ICDE 2007). Istanbul, Turkey., 2007.

[23] J. Zdravkovic. *Process Integration for the Extended Enterprise*. PhD thesis, Royal Institute of Technology, 2006.

[24] X. Zhao and C. Liu. Tracking over collaborative business processes. Proceedings of International Conference on Business Process Management (BPM'06), 2006.