A Scenario-View Based Approach for Supporting Mediated Web Service Interaction

Zhangbing Zhou[⋆]

Digital Enterprise Research Institute,
National University of Ireland at Galway,
IDA Business Park, Lower Dangan, Galway, Ireland
zhangbing.zhou@deri.org

Abstract. Web service interactions have triggered the initiative to identify and solve mismatches from a behavioral aspect. However, current approaches are limited because they mainly focus on control-flow and largely ignore data-flow. Such ignorance may cause unexpected or wrong, even unsupported interactions. To address these problems, we propose a scenario-view based approach, which considers both control-flow and data-flow, to support Web service interactions. We firstly generate scenarios and views for describing a public process. Then, the degree of compatibility of two public processes is computed based on pairwise compatibility of their views. A process mediator is generated for compatible public processes, and thus an interaction is carried out despite mismatches exist among them. This novel approach will benefit service modelers and users not only for a better understanding of public processes, but also for improved capability to identify and solve mismatches, which will further facilitate Web service interactions.

Key words: Public Process; Compatibility; Process Mediator; Mediated Service Interaction; Scenario; View.

1 Introduction

Nowadays, enterprises are able to encapsulate (parts of) their business processes and publish them as Web services [7]. Then they can discover desired Web services, compose them into a new value-added one, and carry out an interaction for achieving a given goal. An interaction in Web service domain can be described as a flow of messages, which may contain a set of data, exchanged among Web services [6]. The major challenging problems that remain are: (1) how to check whether Web services are compatible [1], (2) how to identify and solve potential mismatches, and (3) how to conduct a successful interaction. Because of the inherent *autonomy* and *heterogeneity* of Web services, messages are often different in *format* and *granularity*, and public processes, i.e. the external behavior of Web services [5], are often diverse in activities and messages in terms of *form* and *sequence*. Thus, it is difficult, if not impossible, to find two public processes

[⋆] Advisor: Prof. Dr. Manfred Hauswirth (Email: manfred.hauswirth@deri.org).

that are completely compatible [1]. A Web service interaction is usually carried out with the help of data or process mediators [5], which is called a *mediated service interaction*. For simplicity but without loss of generality, we assume that there is no heterogeneity at a data level among Web services.

Current approaches for checking compatibility, e.g. [1] [10] [11] [13] [18], for supporting process mediators, e.g. [2] [4] [8] [12], and for facilitating service interactions, e.g. [3] [15] [18], mainly focus on control-flow but largely ignore data-flow. They are limited to support mediated service interactions.

To address these problems above, we propose a scenario-view based approach to support mediated service interactions:

- We automatically generate scenarios and views for a public process considering both control-flow and data-flow. A scenario is a complete execution path [1] for a public process. Data dependencies are represented by a data dependency graph, which is optimized into a minimal data dependency graph. Then reduction rules are proposed to identify and remove unnecessary control dependencies specified by *sequence*, *And* and *Loop* blocks in a scenario, and a view is generated to represent this scenario for analysis purposes. A public process can be described as a finite set of views.
- Based on pairwise compatibility of their views, we compute the degree of compatibility for two public processes, which indicates whether they can carry out an interaction with certain conditions. However, for any two views, compatibility is a binary relation and means that mismatches existing between them are resolvable [5].
- Based on control dependencies and data dependencies, we propose to generate a process mediator automatically for facilitating the interaction among compatible public processes.

To the best of our knowledge, our scenario-view based approach is the first study to support mediated service interactions considering both control-flow and data-flow. Because our approach focuses on a behavioral aspect of Web services, it is very useful to support intra-/inter-organizational workflow cooperation [14]. The major contribution of this work includes four aspects: firstly, it will benefit service modelers and users for a better understanding of public processes. Secondly, it will provide guidelines for creation and evolution of public processes. Thirdly, it will provide a novel approach to compute the degree of compatibility of public processes although mismatches may exist among them. Finally, it will generate process mediators for solving resolvable mismatches among public processes, and thus facilitates mediated service interactions.

Our work in this study will be presented as follows. A motivation example is presented in Section 2. A definition and graphical notations for a public process is shown in Section 3. Scenarios and views are generated for describing a public process in Section 4. Then compatibility of public processes is computed based on pairwise compatibility of their views in Section 5, and process mediators are presented in Section 6. Finally, related work is discussed and a conclusion is drawn in Section 7 and 8.

2 A Motivating Example

Figure 1-a, 1-b and 1-c show three public processes for a requestor A (ReqA), a requestor B (ReqB) and a toy shop (ToyS). A definition for a public process is presented in Section 3. Since ReqA lacks *priori* knowledge about the pricing strategy of its potential providers, to get a potential discount, it is willing to provide as much information, like "*S: Customer Information*", as possible before expecting "*R: Price*". Due to the privacy concern, ReqB prefers to provide "*S: Customer Information*" only if ReqB is convinced by the price and committed to buy, thus ReqB expects "*R: Price*" before sending "*S: Customer Information*". ToyS may give a discount depending on the customer profile. That is why that, after receiving "*R: Toy Items*", it expects "*R: Customer Information*". However, a normal price is applied whenever "*R: Customer Information*" is not available. "*R: Customer Information*" is optional for "*S: Price*", but it is mandatary for "*S: Delivery*".

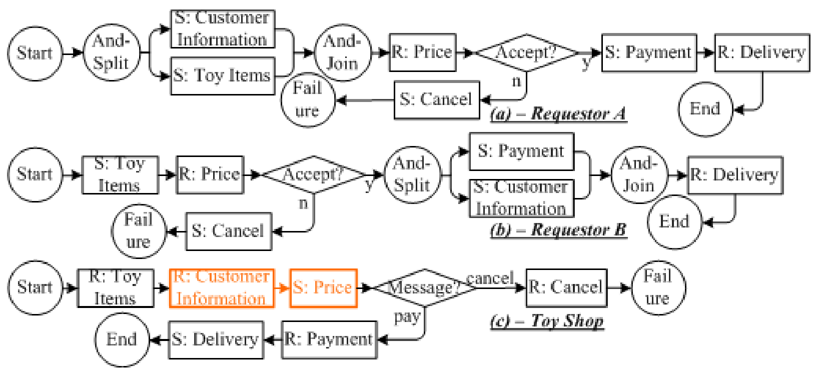


Fig. 1. Public process for the toy shop, the requestor A and B Web services

2.1 An Interaction Between the Requestor A and the Toy Shop

ReqA expects a discount from ToyS. However, Figure 2 shows a snippet of an interaction that ReqA gets a normal price: "*S: Customer Information*" is sent out by ReqA. But for some reasons, it is later than "*S: Toy Items*" of t_1. For his side, ToyS waits "*R: Customer Information*" for t_2 after receiving "*R: Toy Items*". If $t_1 > t_2$, ToyS improperly assume that ReqA should not send "*S: Customer Information*" and then give a normal price.

If there is a process mediator in the middle, ToyS will be notified by the process mediator that "*S: Customer Information*" has been sent out by ReqA. Then ToyS will wait for "*S: Customer Information*" and give a discount.

This example indicates that, without the support of a process mediator, Web services may carry out an ***unexpected***, or ***wrong***, interaction.

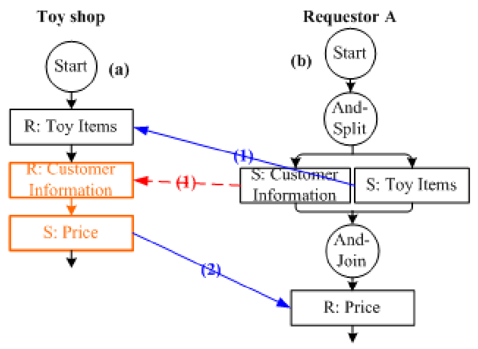


Fig. 2. A direct service interaction between the requestor A and the toy shop

2.2 An Interaction Between the Requestor B and the Toy Shop

According to current approaches, e.g. [1] [10] [11] [13] [18], for checking compatibility, ReqB and ToyS are assumed as incompatible. Process mediation approaches, e.g. [2] [4] [8] [12], regard this mismatch as unresolvable [5].

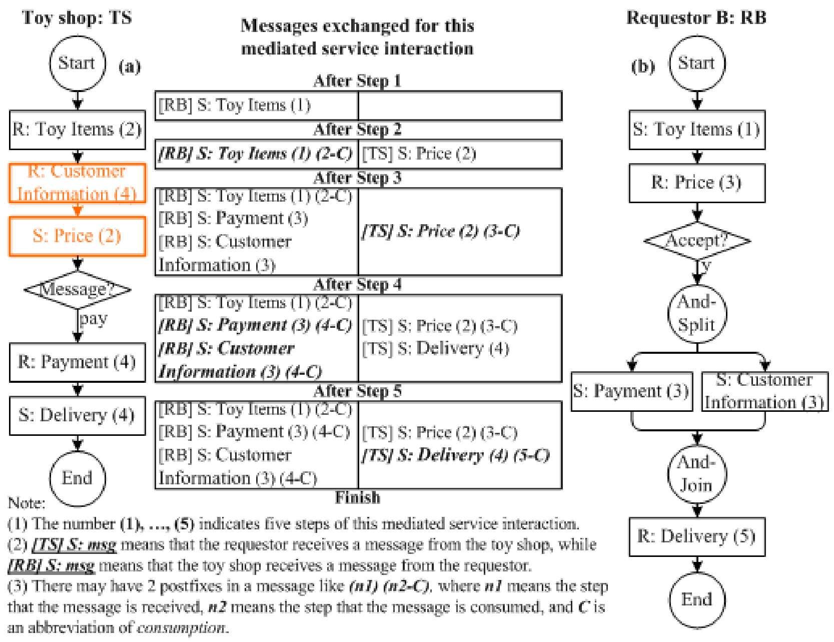
If a process mediator exists in the middle considering both control dependencies and data dependencies, a successful interaction is possible. To do this, ToyS is notified by the process mediator that ReqB will not send "*S: Customer Information*" before receiving "*R: Price*", and thus a *null* message is sent from the process mediator to ToyS as "*S: Customer Information*". Then ToyS can executes the following activity "*S: Price*". However, ToyS will update this message whenever "*S: Customer Information*" is provided by ReqB. Figure 3 shows how this mediated service interaction is carried out. For simplicity, activities that do not contribute to this interaction are not presented.

2.3 A Short Discussion

Our motivating example shows that current approaches are insufficient to check compatibility of public processes, and are limited to support mediated service interactions. Their main shortcoming is that: they only consider control dependencies, but largely ignore data dependencies. As illustrated in the toy shop shown in Figure 1-c, a sequence constraint between "*R: Customer Information*" and "*S: Price*" specifies a control dependency. However, a data dependency between them is optional. This feature causes that a direct service interaction shown in Figure 2 is *unexpected* or *wrong*, and a mediated service interaction shown in Figure 3 can be successfully carried out.

3 Public Process: A Definition and Graphical Notations

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



Note:
(1) The number (1), …, (5) indicates five steps of this mediated service interaction.
(2) *[TS] S: msg* means that the requestor receives a message from the toy shop, while *[RB] S: msg* means that the toy shop receives a message from the requestor.
(3) There may have 2 postfixes in a message like *(n1) (n2-C)*, where *n1* means the step that the message is received, *n2* means the step that the message is consumed, and *C* is an abbreviation of *consumption*.

Fig. 3. A mediated service interaction between the requestor B and the toy shop

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*=$\{grd\}$ is a finite set of guard functions related to control elements, and *ARC*=$\{arc\}$ is a finite set of arcs that connect activities and control elements.

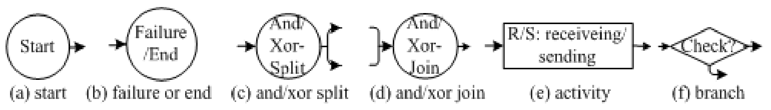


Fig. 4. Graphical notations for modeling a public process

We assume that one message contains one data. Both activities and control elements are the nodes in a public process. Figure 4 shows six basic graphical notations to model a public process. These notations are supported by *JGraph-Pad* [1], based on which our prototype is implemented. In addition, our notations can model six ordering structures specified by *WfMC* [2].

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

4 Generating Scenarios and Views for a Public process

We firstly generate all scenarios for a public process. Data dependencies are presented as 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. Then we propose three reduction rules to identify and remove unnecessary control dependencies specified by *sequence*, *And* and *Loop* blocks in a scenario. 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 includes a finite set of scenarios. A scenario has a corresponding view which represents this scenario for analysis purposes. A public process can be described as a finite set of views. For the sake of space limitation, our algorithms are not presented.

4.1 Generating Scenarios for a Public Process

Only one path of a *Xor* block can be enabled in a given execution depending on the status of guard functions. Similarly, only one branch can be enabled in a given execution for a *Branch* control elements that contribute to exclusive relation. Then, a finite set of scenarios can be generated for a public process.

Definition 2 (Scenario). A scenario *sce* is a complete execution path 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 ToyS, and Figure 3-a shows one of them. There are two scenarios for ReqB, and Figure 3-b shows one of them.

4.2 Optimizing Data Dependencies into a Minimal Data Dependency Graph

Similar as [19], we assume that data dependencies can be extracted from a public process with the help of process modelers.

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.

One data is regarded as *dependent* on another data if (1) a direct link connects them (directly dependent), or (2) several direct links form a path leading from one data to another (indirectly dependent). Then a data dependency graph can be represented as a finite set of *dependent* relations. A data dependency graph is called *functionally equivalent* to another data dependency graph if any *dependency* relation in one data dependency graph can exist in another data dependency graph directly or indirectly.

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 $DATA_{min} = DATA$, $DE_{min} \subseteq DE$. *($DATA_{min}$, 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)*.

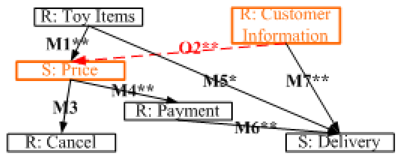


Fig. 5. (Minimal) Data dependency graphs for the toy shop and its scenario

Figure 5 shows a data dependency graph for ToyS, and a data dependency graph for a scenario shown in Figure 3-a is marked with one or two $*$, and its minimal data dependency graph is marked with $**$.

4.3 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 3-a since they are not data dependent on each other. Thus, we present three reduction rules to identify and remove unnecessary control dependencies specified by *Sequence*, *And* and *Loop* blocks. In a scenario, only one path is allowed for a *Xor* block and a branch for a *Branch* control element that specifies an exclusive relation. They are functionally equivalent to *Sequences*. We follow [16] for the function: *fold*, for replacing several contiguous nodes by a single node.

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

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

The rule for *And* block is shown by Figure 6-a and 6-b. An *And* block suggests that all its paths are 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.

Rule 3 (Loop). A *Loop* block is folded into (1) a single node if the *Loop* body can be folded into one node or (2) a sequence of nodes otherwise. Guard functions

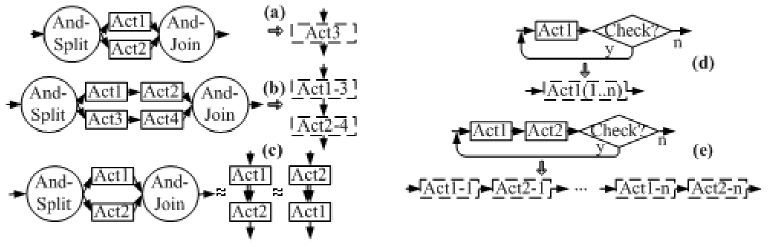


Fig. 6. Reduction rule for *And* and *Loop* Blocks

are defined in the last node to specify the exit conditions of the *Loop* block. There are possibly multiple instances for any node during execution phases.

This rule is shown by Figure 6-d and 6-e. A *Loop* block iterates over one or several nodes until its exit conditions are satisfied, but it doesn't iterate forever in real cases. As suggested by [18], a *Loop* block can be simulated as a *sequence* of at most N repetitions of its *Loop* body, where N depends on a given execution.

4.4 Generating a View for a Scenario

We firstly introduce the concept: view, and its 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 are generated with the help of the minimal data dependency graph of a scenario. E.g., according to Figure 5, *"R: Toy Items"* is required to *"S: Price"*, but *"R: Customer Information"* not.

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 $(MSG_{vw}, CP, cp_0, cp_f, DE_{vw})$. $MSG_{vw} = MSG_{sce}$, $CP=\{cp\}$ 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.

With the help of a minimal data dependency graph, reduction rules can be applied to a scenario recursively until no nodes can be folded anymore. Then checkpoints can be generated and a view can be derived.

5 Computing the Degree of Compatibility

In this section, we focus on the compatibility of two public processes since the majority of interactions are related to two partners, and an interaction involving multiple partners can often be decomposed into several pairwise interactions.

Compatibility aims to check whether two public processes can carry out a successful interaction. Before discussing compatibility, we present what an interaction is, and how an interaction is regarded as successful. Then we compute the degree of compatibility of two public processes based on pairwise compatibility of their views. Due to the space limitation, our algorithms are not presented.

5.1 What is a Successful Interaction?

An interaction means that several public processes, if compatible, can be composed into a complex one for achieving a given goal. Since a scenario represents a complete execution path of a public process, an interaction of public processes is actually an interaction among scenarios of these public processes.

A scenario, or a public process, specifies a set of messages sent to or received from its potential partner(s) following a pre-defined order. Therefore, an interaction can be regarded as a flow of messages exchanged among scenarios. If this flow of message can lead each scenario from its *Start* control element to its final control element, this interaction is regarded as successful. Some mismatches may exist among these scenarios. However, the mismatches are resolvable and can be handled by a process mediator.

5.2 Computing the Degree of Compatibility

For any two scenarios, if a flow of messages exchanged between them can carry out a successful interaction, this indicates that this flow of messages can lead their views from their initial checkpoints to their final checkpoints. In this case, these two views are regarded as *compatible*. On the other hand, if two views are not compatible, these two views cannot interact since at least one checkpoint in a view cannot be guaranteed.

Based on pairwise compatibility of their views, we can define the degree of compatibility for two public processes p_1 and p_2. We assume that there are n_1 views in p_1. For a view v_i $(1 \leq i \leq n_1)$ in p_1, we define a function $comp(v_i \mid p_2)$ to specify whether there is a compatible view in p_2 if $comp(v_i \mid p_2) = 1$, or $comp(v_i \mid p_2) = 0$ otherwise. Thus, the degree of compatibility for p_1 to p_2 is:

$$Compatibility(p_1, p_2) = \frac{\sum_1^{n_1} comp(v_i \mid p_2)}{n_1} \quad (1)$$

Compatibility at a view level is a *symmetric* relation. However, compatibility at a public process level is an *antisymmetric* relation, and $Compatibility(p_1, p_2)$ and $Compatibility(p_2, p_1)$ are often different. Based on the degree of compatibility, we define three classes of compatibility for two public processes p_1 and p_2:

- *No compatibility* if $Compatibility(p_1, p_2) = 0$.
- *Partial compatibility* if $0 < Compatibility(p_1, p_2) < 1$.
- *Full compatibility* if $Compatibility(p_1, p_2) = 1$.

No compatibility means that two public processes cannot interact in any case, and *full compatibility* means that one public process can interact with another in any case, while *partial compatibility* means that one public process can interact with another in at least one but not all cases.

ToyS and ReqB are *fully compatible* with each other. Since there are two views for ToyS and two views for ReqB, and any view in ToyS has a compatible view in ReqB and *vise versa*.

6 Generating Process Mediators

A process mediator aims to facilitate an interaction if mismatches exist among public processes. Since how to generate process mediators is our ongoing work, in this section, we present our strategies:

- There is a space for a process mediator, in which each public process has a sub-space for saving messages. Public processes do not communicate with each other directly. Instead, they send or receive messages to or from a process mediator. This means that production and consumption of messages are time-independent.
- For any message sent by one public process, (1) a process mediator immediately forward it to other public processes that are ready to receive it, and (2) a process mediator checks whether it is potentially expected by other public processes in the future. If yes, the process mediator saves this message for these public processes for possible later usage.
  It is possible that a message is consumed by several public processes. However, if no public process interests in this message now and in the future, it is dropped off immediately and silently.
- In case that a message is expected by a public process and this message is not available in its sub-space. A process mediator will check whether this message is sent out by other public processes already. If yes, this public process blocks and waits until this message is available.
- In case that all public processes are expecting to receive messages and thus an interaction blocks. A process mediator will check each public process: whether data dependencies between current activities and their following activities are not mandatory. If true, the process mediator will generate a *null* message (or an ACK message if acknowledgement is needed) for this public process to execute current activities.

7 Related Work

We discuss the related work from the following four aspects: *analysis of public processes*, *compatibility*, *process mediation* and *service interaction*.

Analysis of public processes: This aspect includes: *Control-flow based* methods [9] [17] only focus on control-flow and largely ignore data-flow. *Dependency based* methods [19] analyze dependencies from data, control, service and cooperation aspects. This work benefits much to our dependency analysis. *View based* methods [3] [20] investigate the relation of private and public processes from a control-flow aspect. Current approaches focus on either a control-flow or data-flow aspect, and are limited to support mediated service interactions.

Compatibility: In [1], the authors presented an analysis for protocol compatibility based on several general protocol operators. Two classes of protocol compatibility are defined: *partial* or *full* compatibility. This work presents a solid theoretic analysis for compatibility from the control-flow aspect. In [10] [11], the authors checked business process compatibility based on *c-graph* and *u-graph*. Two workflow modules are *semantically compatible* if they are *syntactically compatible*, and their *composed system* is *usable*. In [18], the authors established a consistent, multi-lateral collaboration by propagating *parameter constraints* and *execution sequences* among local workflows. This work may not fit to Web services where trust, privacy, and security are critical issues. In [13], the authors verified compatibility following a client/server model. However, these approaches mainly focus on control-flow, and aim to support direct service interactions.

Process mediation: There are mainly two kinds of approaches: in [4] [8], DERI [3] researchers presented five basic patterns for process mediation, integrated process mediation as a component in WSMX [4], and specified the interaction mode with other components. In [2] [12], Benatallah et al presented an adapter-based approach to semi-automatically solve business protocol mismatches. *Mismatch patterns* were used to formalize the mismatches and thus to provide a uniform mechanism to address mismatches. In addition, they proposed to identify actual mismatches semi-automatically (may need the help of service providers), and generated adapters to solve them. However, current approaches are control-flow based and ignore data-flow almost. They are limited to support mediated service interactions.

Service interaction: In [15], the authors proposed a *Public-to-Private* approach for inter-organizational workflow interoperations based on *inheritance*. This is a top-down approach and may not fit to Web service domain. In [18], a bottom-up approach was presented to establish a consistent, multi-lateral collaboration in a decentralized way. As discussed previously, it may not fit to Web service domain. A view-based approach [3] was proposed to support dynamic inter-organizational workflow cooperation including three steps: *advertisement*, *interconnection*, and *cooperation*. Taken together, current approaches mainly focus on control-flow, and aim to support direct service interactions only.

8 Conclusion

In this paper, we firstly identified that, without a process mediator, some Web service interactions could be unexpected or wrong, and some could fail. We also revealed that current approaches are insufficient to check compatibility and to support process mediators because they mainly focus on control-flow and largely

[3] http://www.deri.org/.
[4] http://www.wsmx.org/.

ignore data-flow. To solve these problems, we have proposed a novel approach to automatically generate scenarios and views for describing public processes, to compute the degree of compatibility of public processes based on pairwise compatibility of their views, and to generate process mediators for supporting mediated service interactions.

We have implemented the prototype to generate scenarios and views for public processes, and to compute the degree of compatibility. Furthermore, we propose to support replacement [1] besides interaction. Due to the space limitation, we have not presented them in this paper.

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 Sami Bhiri and Laurentiu Vasiliu for their continuous discussion and valuable comments on purging research problems and solutions.

References

1. Benatallah, B., Casati, F. and Toumani, F.: Representing, analysing and managing web service protocols. Data and Knowledge Engineering. 58, 3, 327–357 (2006)
2. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M. and Toumani, F.: Developing Adapters for Web Services Integration. Proc. of CAiSE (2005)
3. Chebbi, I., Dustdar, S. and Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. Data and Knowledge Engineering. 56, 2, 139–173 (2006)
4. Cimpian, E. and Mocan, A.: WSMX Process Mediation Based on Choreographies. Proc. of 1st Intl. Workshop on Web Service Choreography and Orchestration for Business Process Management at the BPM 2005. (2005)
5. Fensel, D. and Bussler, C.: The Web Service Modeling Framework WSMF. Journal of Electronic Commerce Research and Applications. 113–137 (2002)
6. Fu, X., Bultan, T. and Su, J.: Analysis of interacting BPEL web services. Proc. of WWW (2004)
7. Hao, Y., Zhang, Y. and Cao, J.: WSXplorer: Searching for Desired Web Services. Proc. of CAiSE (2007)
8. Haselwanter, T., Kotinurmi, P., Moran, M., Vitvar, T. and Zaremba, M.: WSMX: A Semantic Service Oriented Middleware for B2B Integration. Proc. of ICSOC (2006)
9. Kim, K.: WSMX: A Semantic Service Oriented Middleware for B2B Integration. Proc. of SKG (2005)
10. Martens, A.: On Compatibility of Web Service. Petri Net Newsletter. 65, 12–20 (2003)
11. Martens, A.: Analyzing Web Service based Business Processes. Proc. of FASE'05, Part of ETAPS'05 (2005)
12. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F. and Casati, F.: Semi-Automated Adaptation of Service Interactions. Proc. of WWW (2007)
13. Shi, Y., Zhang, L., Liu, F., Lin, L. and Shi, B.: Compatibility Analysis of Web Services. Proc. of WI (2005)

14. van der Aalst, W.M.P.: Modeling and analyzing interorganizational workflows. Proc. of CSD (1998)
15. van der Aalst, W.M.P. and Weske, M.: The P2P Approach to Interorganizational Workflows. Proc. of CAiSE (2001)
16. van der Aalst, W.M.P. and Lassen, K.B.: Translating unstructured workflow processes to readable BPEL: Theory and implementation. Information and Software Technology. 50, 3, 131–159 (2008)
17. van der Aalst W.M.P., de Medeiros, A.K.A. and Weijters, A.J.M.M.: Process Equivalence: Comparing Two Process Models Based on Observed Behavior. Proc. of BPM (2006)
18. Wombacher, A.: Decentralized establishment of consistent, multi-lateral collaborations. PhD Thesis at Facultiy of Informatics, Technical University Darmstad. (2005)
19. Qinyi Wu, Q., Pul, C., Sahai, A. and Barga, R.: Categorization and Optimization of Synchronization Dependencies in Business Processes. Proc. of ICDE (2007)
20. Zhao, X. and Liu, C.: Tracking over Collaborative Business Processes. Proc. of BPM (2006)