Inf Syst E-Bus Manage (2015) 13:235–265
DOI 10.1007/s10257-014-0251-6

ORIGINAL ARTICLE

A service pattern model for service composition with flexible functionality

Chien-Hsiang Lee · San-Yih Hwang · I-Ling Yen · Tao-Kang Yu

Received: 8 December 2013 / Revised: 15 April 2014 / Accepted: 23 June 2014 / Published online: 5 July 2014

Abstract A key feature with service-oriented-architecture is to allow flexible composition of services into a business process. Although previous works related to service composition have paved the way for automatic composition, the techniques have limited applicability when it comes to composing complex workflows based on functional requirements, partly due to the large search space of the available services. In this paper, we propose a novel concept, the *prospect service*. Unlike existing abstract services which possess fixed service interfaces, a prospect service has a flexible interface to allow functional flexibility. Furthermore, we define a meta-model to specify service patterns with prospect services and adaptable workflow constructs to model flexible and adaptable process templates. An automated instantiation method is introduced to instantiate concrete processes with different functionalities from a service pattern. Since the search space for automatically instantiating a process from a service pattern is greatly reduced compared to that for automatically composing a process from scratch, the proposed approach significantly improve the feasibility of automated composition. Empirical study of the service pattern shows that the use of the proposed model significantly outperforms manual composition in terms of composition time and accuracy, and simulation results demonstrate that the proposed automated instantiation method is efficient.

Keywords Web service composition · Service pattern · Meta-model · Variability modeling

C.-H. Lee · S.-Y. Hwang (✉) · T.-K. Yu
Department of Information Management, National Sun Yat-Sen University, 70 Lienhai Rd., Kaohsiung 80424, Taiwan, ROC
e-mail: syhwang@mis.nsysu.edu.tw

I.-L. Yen
Computer Science Department, University of Texas at Dallas, Richardson, TX, USA

1 Introduction

Service-oriented architecture (SOA) is an important technology designed to aid agile enterprise in response to the ever changing market (Cummins 2008; Erl 2005). SOA establishes an architectural model that aims to enhance the efficiency, agility, and productivity of enterprise by positioning services as the primary means (Erl 2007). By reusing services and composing services into composite version, innovative applications can be developed quickly and at lower cost. There have been CASE tools that support the development of composite Web services by considering business process management (IBM 2011; Oracle 2011; SAP 2011). However, business process analysts still have to manually plan the workflow and then consult IT staff to deploy workflow-based service composition by linking automated activities with Web service operations. It demands considerable effort for service composition.

To reduce the effort of service composition, many researchers have proposed approaches that mostly adopt reasoning techniques, such as planning and logical reasoning, to automatically compose Web service operations to satisfy a given goal (Akkiraju et al. 2006; Chen et al. 2009; Doshi et al. 2004; Pistore et al. 2005; Sirin et al. 2004; Zeng et al. 2008). However, many business processes are complex, and the automated composition techniques have limited applicability due to the difficulties in building comprehensive planning domain and the scalability problem.

In addition, agile enterprise requires flexible service composition strategy to assist information system developers in developing flexible and adaptable solutions that span a wide variety of requirements for changing environments. Flexible service interface and configurable design are required to accommodate new requirements (Kapuruge et al. 2010; Ruokonen et al. 2008). The capabilities of service composition in terms of its interface have to be changeable to support its users in different contexts (Nguyen et al. 2011b). Abstraction and reuse of core logics from existing service compositions could improve composition efficiency and produce less error-prone processes (Ruokonen et al. 2008).

Workflow templates approach is a basic solution toward building a flexible workflow in a service composition. General workflow templates are in a form of abstract workflows, where component services can be concrete or abstract. An abstract service has well-defined I/O data types, precondition, and effects, abbreviated IOPE, yet is not executable. It should be grounded to some concrete service before execution. A lot of works are proposed to consider QoS-driven service composition based on workflow templates, where different concrete services are selected to ground the abstract services to satisfy different QoS goals or request-specific constraints (Geebelen et al. 2008; Gil et al. 2011). Thus, workflow templates are adaptable in QoS but still very specific and have no flexibility in functionality. In fact, a workflow template is considered as an abstract composite service in a formal model and its IOPE definitions are fixed.

On the other hand, research in variability modeling focuses on modeling flexible workflows capable of changing control flows and substituting constitute services to accommodate possible behaviors in different contexts. Some adaption methods, such as adaptation patterns (Döhring and Zimmermann 2011), configuration operations (Gottschalk et al. 2008), and adaption rules (Kumar and Yao 2012), are proposed to instantiate different workflow variants from a base workflow. The major purpose of these works is to improve maintainability of service composition by modeling a complex service composition, which could support expected behaviors in various situations, without maintaining a lot of similar workflows (Döhring et al. 2014; Smirnov et al. 2012). They are also confined in narrow applications and incapable of supporting unexpected new requirements.

To overcome the limitations of previous works and to provide flexible service composition, we propose a *service pattern* approach by which flexible service interface and adaptable workflow are included in a process template, called service pattern, for versatile capabilities. In other words, it is desirable that the IOPE definitions of a service pattern is nondeterministic so that it can be instantiated into different abstract workflows with fixed IOPEs and subsequently grounded into different concrete workflows.

Frequently, there exist multiple workflows offering similar yet different functionalities. With the service pattern concept, it is possible to express a set of similar workflows by one service pattern. In enterprise systems, a service pattern can be used to express similar operations in the organization. With the popularity of SaaS, many providers are now building platforms to allow users to compose services to achieve configurability so as to fulfill various customer needs. Various workflows in a SaaS system are frequently similar and can be expressed as a single or a few service patterns to greatly reduce the time and efforts of the service composition tasks by the end customers. More importantly, a service pattern offers the possibility of constructing a novel abstract workflow that is similar to yet different from all the existing workflows.

In this paper, we develop a meta-model for representing service patterns and a suite of techniques to instantiate them into concrete service compositions that meet different user needs. Our contributions include:

- We extend BPMN notation to formally define the service pattern model. The unique features, such as nondeterministic service specifications and adaptable flow structures, leverage existing service composition techniques and achieve flexible functionality, resolving the functional inflexibility problem found in existing workflow templates.
- A systematic instantiation process is outlined in the paper. A designer can step by step define the instantiation parameters for the target application system without needing to fully understand the internals of the pattern.
- We develop a rule-based reasoning technique that automate the process instantiation, service selection, and grounding of a service pattern. A set of interference rules are defined to facilitate reasoning. This further reduces the efforts for service composition. Also, due to the use of service patterns, the problem space for reasoning is highly confined, making reasoning a feasible and efficient solution in the composition process.
- We conduct an empirical study to evaluate the effectiveness of the proposed meta-model and a simulation study to measure the performance of the automated instantiation method. The result of model evaluation shows that the

meta-model outperforms the other two baseline methods, which manually create process from scratch and consult similar service compositions, in terms of composition time and accuracy. The experimental results on the proposed reasoning method demonstrate that the proposed method consumes reasonable time, i.e., below two seconds for over 300 candidate Web services.

The rest of this paper is structured as follows. In Sect. 2, we summarize some related works. Then, a motivating example is introduced to manifest the research problem in Sect. 3. In Sect. 4, we describe our proposed model and illustrate the model by using a service pattern example. Next, we describe the automatic instantiation process in Sect. 5. Experimental results for the proposed model and the instantiation method are presented in Sects. 6 and 7 respectively. Finally, Sect. 8 concludes the paper.

2 Literature review

It is well recognized that manually composing Web services to satisfy a particular requirement is a tedious work, and there have been many proposed works that seek to automatically compose Web services to achieve a given goal. Research in this line treats automatic Web service composition as a planning problem. Several planning techniques are proposed, including rule-based planning (Medjahed et al. 2003; Ponnekanti and Fox 2002), hierarchical task network planning (Sirin et al. 2004), planning based on model checking (Pistore et al. 2005), and planning based on Markov decision processes (Chen et al. 2009; Doshi et al. 2004). However, due to the high complexity pertaining to the service function and requirement description and the often large number of available services, it is difficult to reason in a realistic environment without additional assistance.

As automatic service composition proves to be a difficult task, some researches focus on workflow templates, which are generally regarded as abstract workflows with components services being abstract. This concept is actually implicitly incorporated in OWL-S where services' IOPE can be specified without specific grounding. In some works, the workflow of service composition can be adapted according to parameter values (Fu et al. 2009; Geebelen et al. 2008). Most works, however, consider dynamic composition for the workflow by selecting different services to satisfy different QoS goals or request-specific constraints (Geebelen et al. 2008; He et al. 2008; Yang et al. 2009). They generally consider verbal based flow description and then instantiate the template with specialized concrete services. They allow for the substitution of concrete Web services to achieve QoS goals but are still very specific and lack of flexibility in functionality.

In addition, there have been some works (Amarouche et al. 2011; Barhamgi et al. 2010; Hwang et al. 2012) that intend to select and compose data-providing (DP) services into an execution workflow for data integration. Barahmgi et al. (2010) employ query rewriting to find the services conforming to data semantics of query request and generate executable service composition after confirming that all variables of query request are correctly covered by the identified services. Based on the work of Barahmgi et al. (2010), Amarouche et al. (2011) propose methods to handle the semantic conflicts of data exchange between component services, such as the conversion between different measuring units. Hwang et al. (2012) represent the causal relationship between exchange variables of DP service types using Bayesian network model and intend to find the service composition that provides more available data for the users. In addition to data manipulation services, our work integrates flow adapting controls and prospect services into our proposed model to enhance the function flexibility of service patterns. To the best of our knowledge, both function flexibility and message mismatch resolution are seldom considered in previous works.

Based on the idea of variability management in software product line domain (Pohl et al. 2005), some recent works propose to accommodate several variation points in the workflow of composite Web service, and these variation points allow users to adapt workflow or replace services to build customized composite Web services. In addition to common functionality, the customized composite services provide assorted services for different users. Therefore, variability modeling is crucial and variability types differ at different abstraction levels. At the feature level, customizable function items for different user requirements are major consideration and they are categorized as common, optional, and alternative features. Common features are required function items for all users, optional features were either included or excluded, and alternative features have several variants for choice (Abu-Matar and Gomaa 2011; Nguyen et al. 2011a).

To represent flexible workflow, some works consider variability at the model level to instantiate workflow variants. The variation points, including possible alternatives under different situations, are embedded in workflow template to provision workflow adaptation. The adaptation of workflow could be realized from simple alternative selection to a series of workflow adjusting operations. For example, Mietzner and Leymann define variability descriptor including the location of variation points and their alternative, and transform a template BPEL process into customized BPEL processes according to users' choice of alternative in the descriptor (Mietzner and Leymann 2008). Kumar and Yao propose an algorithm in which the predefined adaption rules trigger some adapting operations, e.g. remove, insert, and replace, on template process to produce process variants (Kumar and Yao 2012). Döhring and Zimmermann introduce adaption patterns that define specific adapting actions for particular event to support sophisticated adaptation (Döhring and Zimmermann 2011). However, a complex workflow template that incorporates variability often times is difficult to comprehend. In addition, they only consider expected behaviors for different situations but fail to support unexpected new requirements.

3 A motivating example

Store sales data are often analyzed in a business setting to help make various business decisions. Frequently, to support effective analysis, other database tables need to be joined to provide more information about sale items for different usages, e.g. product safety stock for stock replenishment and product unit cost for profit computing. From the joined table, specific data can be selected, then aggregated horizontally or combined vertically to obtain summarized results that support various types of decision making, e.g., stock replenishment, profit computing, and order fulfillment. Additional tasks, such as currency and metric conversions, and format conversions for subsequent activities, may be needed, especially for international business. Furthermore, different execution sequences in a sub-workflow, such as skipping some services, allowing different processing orders, or providing alternative sub-workflows, etc. can be provided to offer a flexible high level business process. This general business process can be captured by a general composite service, denoted using BMPN, as shown in Fig. 1.

In Fig. 1, *GetStoreSales* and *GetProductProfile* services retrieve sales data and product data respectively, which are subsequently combined to form a joined table. The *SelectRequiredData* service then filters the table and selects data entries from the table that satisfy some given criteria. *ConvertByCurrency* and *ConvertByUnit* are offered as skippable services that process monetary or quantitative attributes to assure consistency. Then, aggregating sales data by specific attributes and combining the aggregated data with some extended data could be needed to produce more complete sales data. After that, some required data are extracted and organized according to the format needed by subsequent activities. Finally, the core services *AggregateExistingAttributes* and *DeriveNewAttributes* accept the data produced by prior services, then aggregate the input attributes and derive new attributes. For different purposes, these final services may be executed in different sequences and the latter task may be skipped in some cases as shown in exclusive choices in Fig. 1.

The general composite service can be instantiated into several processes with different functionalities. In Fig. 2, three workflows instantiated from the general composite service are shown, including (1) the replenishment workflow, abbreviated RPW which derives the quantity of ordered items from their sales quantity, stock quantity, and safety stock; (2) the profit computation workflow, abbreviated PCW, which calculates net profit from sales amount and store expenses in a given time period; and (3) the order fulfillment workflow, abbreviated OFW, which aggregate customer's order amount, charge money for order, and then derive a delivery plan for shipping.

In order to achieve flexibility and efficiency, some requirements are essential for the meta-model formulating service patterns with functional flexibility. First, the control and data flow in the pattern should be adaptable. Second, the specification of component services in the pattern should be customizable to fit special requirements under different contexts, rather than a fixed functional specification. For instance, the *DeriveNewAttributes* and *AggregateExistingAttributes* services in Fig. 1 are instantiated into different concrete services with different functionalities in the three concrete workflows in Fig. 2 yet preserving the common behavior of computing based on column-wise aggregations. Furthermore, all of them have similar behavior pattern but different contents. For example, *Replenish*, *ComputeProfit*, and *Shipping* services derive new attributes, namely order quantity, store profit, and shipper, respectively, from sales and products data. Third, additional constraints should be

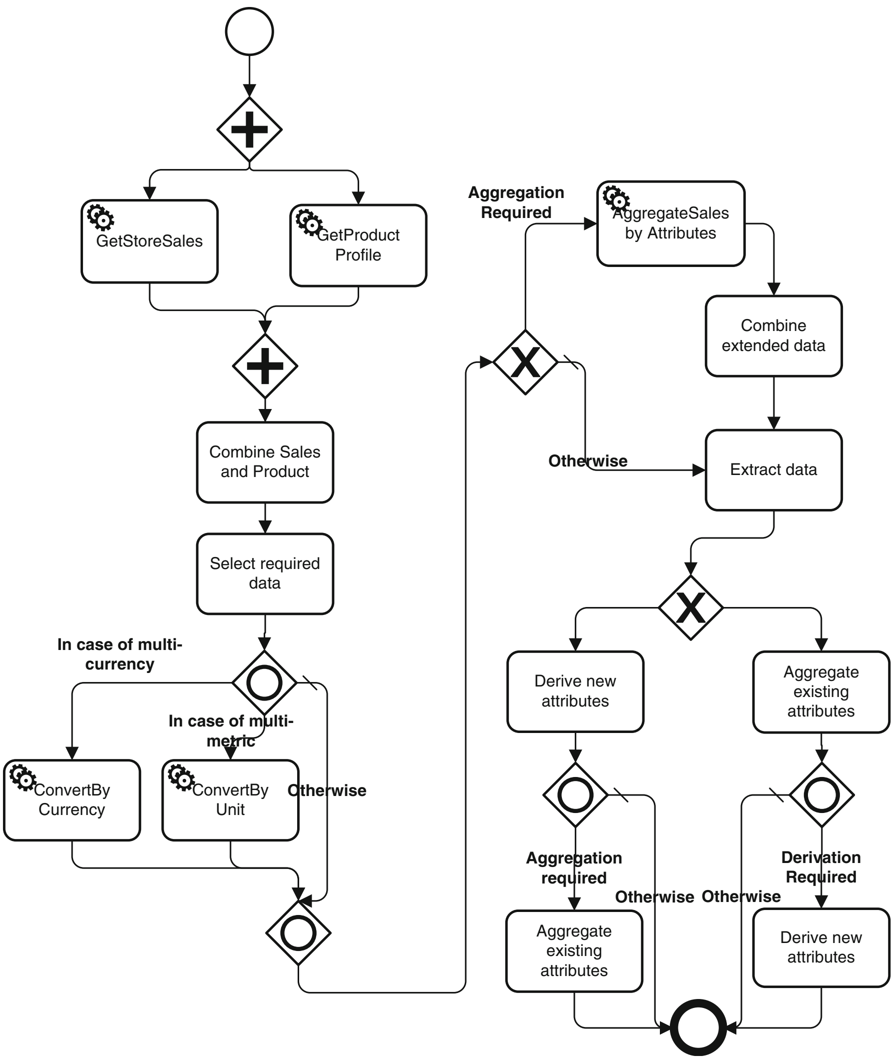

Fig. 1 An example of general composite service

enforced during pattern instantiation to avoid message mismatches between cooperative services or type incompatibility between customizable services and their concretized counterparts. For example, in Fig. 2a, *Replenish* service accepts the output of *ExtractData* service, so input type of *Replenish* service should be equivalent to output type of *ExtractData* service. Therefore, message flow declaration for message validation and message type restriction for customable message type are crucial to the proper instantiation of a service pattern.

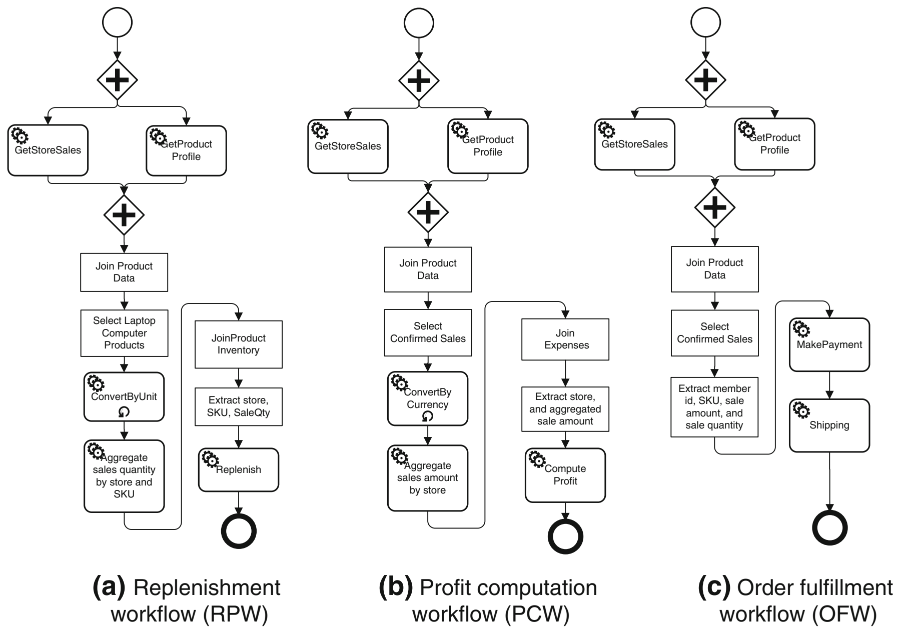
GetStoreSales
GetProduct Profile
Join Product Data
Select Laptop Computer Products
ConvertByUnit
Aggregate sales quantity by store and SKU
JoinProduct Inventory
Extract store, SKU, SaleQty
Replenish
(a) Replenishment workflow (RPW)
GetStoreSales
GetProduct Profile
Join Product Data
Select Confirmed Sales
ConvertBy Currency
Aggregate sales amount by store
Join Expenses
Extract store, and aggregated sale amount
Compute Profit
(b) Profit computation workflow (PCW)
GetStoreSales
GetProduct Profile
Join Product Data
Select Confirmed Sales
Extract member id, SKU, sale amount, and sale quantity
MakePayment
Shipping
(c) Order fulfillment workflow (OFW)

Fig. 2 Three workflow-based service compositions

4 Service pattern model

Various standards, such as OWL-S and BPMN, have been proposed for service and process specification. Although BPMN provides strong support in workflow specification, a workflow defined in BPMN cannot realize other functions beyond its original scope. To increase functional flexibility, we propose the new concept "prospect services" and "adaptable workflow" as the major service pattern elements, allowing them to be defined with specific yet flexible properties that are controlled by instantiation parameters. Also, we extend BPMN model to support the specification of the service patterns. In the following two subsections, we introduce the two major service pattern elements, prospect services and adaptable workflow templates, with their corresponding instantiation parameters. In Sect. 4.3, we formally define the service pattern.

4.1 Prospect service

We propose prospect service containing nondeterministic IOPE specification to fit the essential requirement of supporting customizable specification with functional flexibility as described in Sect. 3. In OWL-S, IOPE are defined in the upper ontology to describe the behavior of a service. A concrete or an abstract service has well-defined IOPE. A prospect service, on the other hand, contains partially specified IOPE. The differences of concrete service, abstract service, and prospect service are summarized in Table 1.

Table 1 The differences among three types of services

Service type	IOPE specification	Grounding information
Concrete service	Specific	Specific
Abstract service	Specific	Undefined
Prospect service	Partially defined	Undefined

The I/O messages of prospect service can have customizable data types, which includes data type parameters that are to be fully defined during instantiation. Its preconditions/effects can have customizable predicates, which includes predicate parameters to be defined at instantiation time. These data type and predicate parameters are called instantiation parameters, which are denoted with prefix "?" in this work. The value of instantiation parameters is assigned at instantiation time to concretize the specification of prospect service for realizing specific functionality. Figure 3 shows an example specification of a prospect service, where instantiation parameters are in bold.

The prospect service, *DeriveData*, derives new attributes from existing attributes. Its input and output messages use customizable data types, namely ?TypeOfDrvIn and ?TypeOfDrvOut, respectively. In addition, restrictions can be specified for customizable types using first-order logic, which can be a specific class in a domain ontology and/or the relations between the output schema and the input schema. For example, the following rules restrict that ?TypeOfDrvIn and ?TypeOfDrvOut must map to the Table concept in the domain ontology (Anonymous 2007).

$$\text{ModelRef(?TypeofDrvIn)} = \text{Ont\#Table}$$
$$\text{ModelRef(?TypeofDrvOut)} = \text{Ont\#Table}$$

Also, *DeriveData* prospect service includes two predicate parameters, ?cpDeriveDataPrecond() and ?cpDeriveDataEffect(), with which the precondition and effect of it can be customized to fit different requirements. In Fig. 3, *DeriveData* combines concrete and customizable effects to declare the desired outcomes with functional flexibility. The concrete effect of *DeriveData* specifies that each input tuple has a corresponding output tuple and each output tuple must be computed from one input tuple. At instantiation time, the specific predicates can be given to concretize these predicate parameters so that specific functionality for the service is specified. For example, to fulfill the requirement of replenishment in B2B environment, developers may want to match prospect service *DeriveData* with the concrete service that not only derives order quantity but also automatically sends orders to B2B partners. Correspondingly, the developer can do the following assignment for predicate parameter ?cpDeriveDataEffect:

$$\text{?cpDeriveDataEffect(?TypeOfDrvOut: DrvOut)} = (\forall e_3 \in \text{DrvOut, Sent2Partner}(e_3))$$

4.2 Adaptable workflow template

To make the workflow of service pattern flexible, we propose the adaptable workflow template, in which some data manipulation services, flow adapting

Name: DeriveData
In: **?TypeOfDrvIn** DrvIn
Out: **?TypeOfDrvOut** DrvOut
Precondition:
?cpDeriveDataPrecond(?TypeOfDrvIn: DrvIn)
Effect:
$(\forall\, e_1\, (e_1 \in \text{DrvOut} \Rightarrow \;\; \exists e_2\, (e_2 \in \text{DrvIn}), \text{Key}(e_2)=\text{Key}(e_1)$
$\wedge \forall a \in$ Attributes(**?TypeOfDrvOut**), $\text{ComputeFrom}(e_1.a, e_2))\,) \wedge (|\text{DrvIn}| = |\text{DrvOut}|)$
$\wedge$ **?cpDeriveDataEffect(?TypeOfDrvOut: DrvOut)**

Fig. 3 An example specification of a prospect service

controls, and message alignment are included to allow tailoring workflow for specific purpose besides prospect services. Each constituent service in an adaptable workflow template can be concrete, abstract, or prospect. We also propose a special kind of concrete services, called data manipulation service, which is capable of handling data transformation for facilitating data exchange between constituent services. Data manipulation services resolve message mismatch problem between constitute services to promote service reuse when their functions fit the requirement yet the I/O types do not perfectly match. This kind of services includes selection, join, and extraction operations like relational algebra and their features can be found in our previous work (Lee and Hwang 2009).

We define two types of flow adapting controls: the alternative selection and the skippable toggle. An alternative selection is an ordered pair $<a_e, SW>$, where a_e is an instantiation parameter with enumerated type and SW is a set of sub-workflow alternatives, each associated with a case value. At instantiation time, an instantiation parameter value of a_e is specified to choose one of the alternative sub-workflows. A skippable toggle is an ordered pair $<s, a_b>$, where s is a service and a_b is an instantiation parameter with Boolean type. During instantiation, the value $<s$, TRUE> ($<s$, FALSE>) indicates that the service s is (not) skipped. As a result, the tailored workflow retains only non-skipped services.

The routing constructs, e.g. sequence and parallel, can be used to define the execution paths in a workflow, but we also need constructs for specifying the message exchange between component services. Generally, the output of a preceding service is fed to the input of some succeeding service(s). Therefore, source parameters and destination parameters for a message exchange must be precisely specified. This is especially important in a service pattern because the specified data type parameters at instantiation time must be in line with the message exchanges specified in the service pattern. Due to the lack of such feature at design time in existing models, we define a new construct, message alignment, to connect the I/O parameters of interacting services. A message alignment is a quadruple, $<s_s, s_d, v_s, v_d>$, where s_s and s_d are the source and destination services, respectively, and v_s and v_d are the formal output variable of source service and input variable of destination service, respectively.

In summary, we extend BPMN by developing five more new symbols for the above extended features as shown in Fig. 4. They can be applied to BPMN process models to prescribe adaptable workflow template of service patterns.

4.3 Service pattern

A service pattern is specified by an adaptable workflow template and flexible IOPE that are partially defined by instantiation parameters. To support instantiation, a service pattern interface should also include the specification of its instantiation parameters.

Definition (*Service pattern*) A service pattern is defined by the tuple, <*I*, *O*, *P*, *E*, *IP*, *WT*>, where *I*(*O*) is the set of input (output) parameters, *P* and *E* are sets of preconditions and effects, respectively, *IP* is a set of instantiation parameters, and *WT* is an adaptable workflow template.

The interface definition for the example service pattern, *SalesCube*, is shown in Fig. 5. Note that *SalesCube* can be instantiated to any of the three example workflows in Fig. 2. The adaptable workflow template of *SalesCube* is shown in Fig. 6 using the extended BPMN symbols as shown in Fig. 4; and, the IOPE specification of the involved prospect services are shown in Fig. 7. In Fig. 5, *SalesCube* takes several concrete input arguments, including the store list (*In_sl*), the extracting attributes (*In_ea*), the period of data (*In_tp*), the target currency (*In_cu*), the aggregating attributes (*In_ga*), and a customizable input *In_ed*, which has a customizable data type **?TypeOfEd**. The output variable *Out* also has a customizable data type **?TypeOfOut**. The concrete precondition and effect are also specified to restrict that the input store list cannot be empty, the period of sales data must be at least 7 days earlier, and the output cannot be empty.

The instantiation parameters in Fig. 5 include data type parameters and predicate parameters for IOPE customization of service pattern and prospect services, as well as workflow adapting parameters for workflow adaptation. The restriction of data type parameters, as mentioned before, can also be specified. For example, at "restriction of data type parameters" section,**?TypeOfEd**, **?TypeOfOut**, and several other data type parameters, are restricted to be of "Table" type, which is defined in an ontology namespace "Ont". In addition, the key attributes of ?TypeOfJedIn, ?TypeOfed, and ? TypeOfJedOut must be the same, and the attributes of ?TypeOfJedIn must be a subset of the attribute union of TSalesData and ? TypeOfed. The values of these parameters determine the workflow of service composition during instantiation time.

The adaptable workflow template of *SalesCube* given in Fig. 6 is similar to Fig. 1 except that prospect services, data manipulation services, and flow adapting controls are incorporated into it. Four prospect services, shown in Fig. 7, provide a wide variety of functionalities to choose from. As a result, *SalesCube* can be instantiated into several service compositions with dissimilar functionalities when the involved prospect services are concreted as disparate services for different purposes. In addition, two service skippable toggles are attached to *ConvertByCurrency* and *ConvertByMetric* services in order to flexibly combine these features on demand, and two alternative selections are encompassed to construct two alternative execution paths. Also, message alignments are added in Fig. 6 to facilitate compatibility validation of the connected I/O arguments after their types are concretized. Consider mf_{10} message alignment, it connects the output variable of

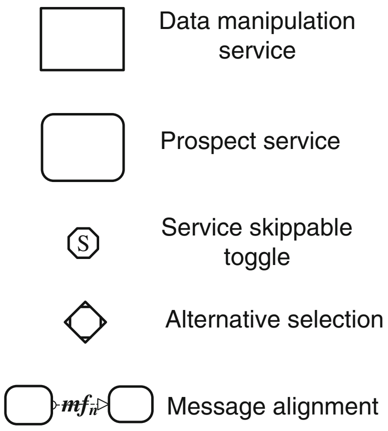

Fig. 4 Symbols of extended features for the adaptable workflow template

Name: SalesCube pattern
IN: TStore:In_sl, TSalesClass:In_ea, TTimePeriod:In_tp, TMoney:In_cu,
TSalesClass:In_ga, ?TypeOfEd In_ed
OUT:?TypeOfOut Out
Precondition:
|In_sl|≠0∧Day(In_tp)<CurrentDate()-7
Effect:
|Out|>0
Instantiation parameters:
Data type parameters:
?TypeOfEd, ?TypeOfOut, ?TypeOfJedOut, ?TypeOfExtrIn, ?TypeOfExtrOut,
?TypeOfDrvIn, ?TypeOfDrvOut, ?TypeOfAggIn, ?TypeOfAggOut
Restriction of data type parameters:
ModelRef(?TypeofEd)=Ont#Table,
ModelRef(?TypeOfOut)=Ont#Table,
ModelRef(?TypeOfExtrOut)=Ont#Table,
ModelRef(?TypeOfJed)=Ont#Table,
ModelRef(?TypeOfJedOut)=Ont#Table,
ModelRef(?TypeOfDrvIn)=Ont#Table,
ModelRef(?TypeOfDrvOut)=Ont#Table,
ModelRef(?TypeOfAggIn)=Ont#Table,
ModelRef(?TypeOfAggOut)=Ont#Table,
KeyAttr(?TypeOfJedIn)=KeyAttr(?TypeOfed)=KeyAttr(?TypeOfJedOut),
Attrs(?TypeOfJedOut)⊆Attrs(TSalesData)∪Attrs(?TypeOfEd)
Predicate parameters:
?fpSelectData(TSalesData:SelIn),
?cpDeriveDataPrecond(?TypeOfDrvIn: DrvIn) ,
?cpDeriveDataEffect(?TypeOfDrvOut: DrvOut),
?cpAggregateDataPrecond(?TypeOfAggIn: AggIn),
?cpAggregateDataEffect(?TypeOfAggOut: AggOut)
Workflow adapting parameters:
?SkipConverByCurrency: Boolean;
?SkipConvertByUnit: Boolean;
?SkipDeriveData: Boolean;
?SkipAggregateData: Boolean;
?ChooseFormat: {'Cube','Flat'};
?ChooseComputeOrder: {'AggregateFirst','DeriveFirst'};

Fig. 5 IOPE specification and instantiation parameters of *SalesCube* service pattern

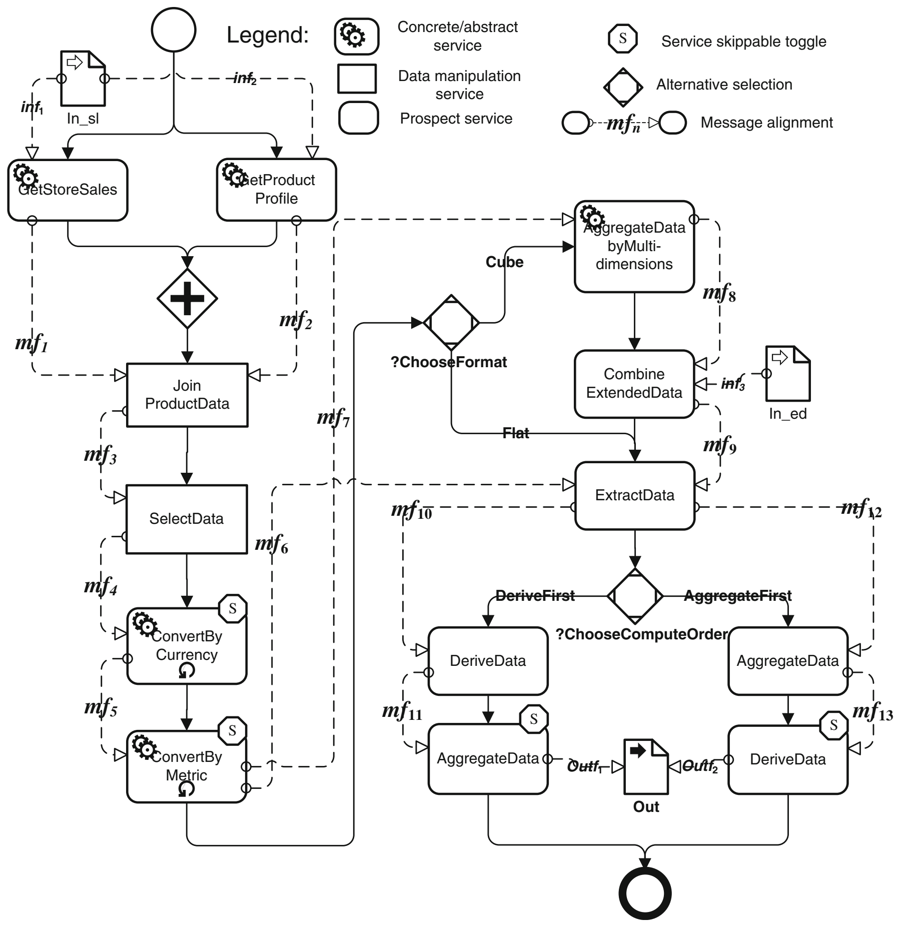

Fig. 6 The adaptable workflow template of *SalesCube* service pattern

ExtractData service to the input variable of *DeriveData* service. Thus, the data type of *ExtractData*'s output should be compatible with the data type of *DeriveData*'s input. The validation of type compatibility should be performed at the instantiation time. Other message alignment mf_i, for each i, can be validated in the same way.

4.4 Service pattern instantiation

Based on application requirements, the instantiation of a service pattern involves the assignment of instantiation parameters, which will have the following effects: (1) unselected alternative sub-flows and skipped services are eliminated according to the assignment of workflow adapting parameters; (2) prospect services are transformed into abstract services by instantiating nondeterministic data types and predicate parameters according to the assignment of data type parameters; (3) message alignments are checked, instantiated data types are validated against the

Name: SalesCube pattern
Prospect Services:

Name: ExtractData
In: ?TypeOfExtrIn ExtrIn, TAttrNames ExtrAttr
Out: ?TypeOfExtrOut ExtrOut
Precondition:
ExtraAttr$\subseteq$Attriubtes(TSalesData)
Effect:
ExtraAttr=Attributes(?TypeOfExtrOut)$\wedge$|ExtrIn|=|ExtrOut|$\wedge$
$(\forall e_1\ (e_1 \in \text{ExtrOut} \Rightarrow \exists e_2\ (e_2 \in \text{ExtrIn}), (\forall a \in \text{ExtrAttr}\ e_1.a = e_2.a)))$

Name: CombineExtendedData
In: TSalesData JedIn, ?TypeOfEd JedEd
Out: ?TypeOfJedOut JedOut
Precondition: null
Effect: null

Name: DeriveData
In: ?TypeOfDrvIn DrvIn
Out: ?TypeOfDrvOut DrvOut
Precondition:
?cpDeriveDataPrecond(?TypeOfDrvIn: DrvIn)
Effect:
$(\forall\ e_1\ (e_1 \in \text{DrvOut} \Rightarrow\ \exists e_2\ (e_2 \in \text{DrvIn}), \text{Key}(e_2)=\text{Key}(e_1)$
$\wedge \forall a \in$ Attributes(?TypeOfDrvOut), ComputeFrom$(e_1.a, e_2))$)$\wedge$(|DrvIn |=| DrvOut|)
$\wedge$?cpDeriveDataEffect(?TypeOfDrvOut: DrvOut)

Name: AggregateData
In: ?TypeOfAggIn AggIn
Out: ?TypeOfAggOut AggOut
Precondition:
?cpAggregateDataPrecond(?TypeOfAggIn: AggIn)
Effect:
|AggOut|$\leq$|AggIn|$\wedge$?cpAggregateDataEffect(?TypeOfAggOut: AggOut)

Fig. 7 The IOPE specification of involved prospect services in *SalesCube* service pattern

constraints of the corresponding customizable data types and mismatches of data types between interacting services are resolved; (4) same as the regular matchmaking process, the abstract services are matched with concrete instances to build an executable workflow-based Web service composition. Note that step 4 will not be discussed in this paper as it is the typical QoS-based service selection problem and a lot of researches have been devoted to it, as described in Sect. 2.

Based on the specification of *SalesCube* service pattern, as shown in Figs. 5, 6, and 7, we use RPW, as shown in Fig. 2a, as an example to demonstrate the instantiation process. At first, we assign values to the instantiation parameters for RPW. In this case, the extended data of type TProdInv, indicating product inventory, and the desired output of type TOrder, indicating product order, and they are specified as follows.

?TypeOfEd = TProdInv;
?TypeOfOut = TOrder

Among the four skippable services, RPW does not need *ConvertByCurrency* and *AggregateData* services but requires *ConvertByUnit* and *DeriveData* services. Accordingly, the binding of instantiation parameters can be specified as follows:

?SkipConvertByCurrency = True;
?SkipConvertByUnit = False;
?SkipDeriveData = False;
?SkipAggregateData = True;

On the other hand, RPW has to aggregate sales by store and SKU, and combine inventory information. Therefore, the instantiation parameters of the two alternative selections in Fig. 6 can be specified as follows:

?ChooseFormat = 'Cube';
?ChooseComputeOrder = 'DeriveFirst'

Figure 8 lists the assignment of all instantiation parameters for RPW process. Based on the values of workflow adapting parameters, the customized workflow template is shown in Fig. 9.

We then consider data type compatibility checking and message alignment and filter out those concrete services that do not satisfy some constraints. The remaining concrete services can be used for grounding abstract services. Finally, a concrete service is chosen and bound for each abstract service based on semantic constraints or QoS tradeoffs. The instantiation of *SalesCube* service pattern into PCW and OFW composite services can be conducted by following similar instantiation process.

5 Automated instantiation parameter assignment

As mentioned in Sect. 4.4, instantiation parameters have to be assigned in order to instantiate a service pattern. However, the user may not be capable of specifying all instantiation parameters properly. The values of instantiation parameters depend on the business requirements and software development environment such as existing Web services and message schemas. Specifically, business requirements define the goal, which involves input types, output types, and some of the instantiation parameter values. The values of the other unspecified instantiation parameters, however, can be determined by considering the existing Web services and message schemas. For example, a data type parameter can be assigned a message type only if all the prospect services that involve the data type have corresponding concrete Web services that use the message type. Therefore, an assignment for instantiation parameters is feasible only if it can be supported by the current software environment. In this section, we describe a reasoning method that automatically infers a feasible assignment for the unspecified instantiation parameters, if any, from the existing Web services.

5.1 The system architecture

Consider the instantiation of a service pattern *sp* into a target composite service *ts*. The first step is for the developer to specify the IOPE of *ts*, denoted *ts*.IOPE as well as some instantiation parameters needed for the goal of *ts*. Next, instead of manually

Fig. 8 The assignment of instantiation parameters for RPW process

Composition name: RPW
Data type parameters:
?TypeOfEd=TProdInv;
?TypeOfOut=TOrder;
?TypeOfJedOut=TSalesInvQty;
?TypeOfExtrIn=TSalesInvQty;
?TypeOfExtrOut=TReplenishReq;
?TypeOfDrvIn =TReplenishReq;
?TypeOfDrvOut=TOrder;
?TypeOfAggIn=null;
?TypeOfAggOut=null;
Predicate parameters:
?fpSelectData= (Product.Cateogry="LaptopComputer");
?cpDeriveDataPrecond=null;
?cpDeriveDataEffect=($\forall e \in$ DrvOut, Sent2Partner(e));
?cpAggregateDataPrecond=null;
?cpAggregateDataEffect=null;
Workflow adapting parameters:
?SkipConverByCurrency=True;
?SkipConvertByUnit=False;
?SkipDeriveData=False;
?SkipAggregateData=True;
?ChooseFormat='Cube';
?ChooseComputeOrder='DeriveFirst';

building *ts* based on *sp* and *ts*.IOPE, we propose an automated tool to reason and derive *ts* from *sp*, *ts*.IOPE, and the specified instantiation parameters. Since the derivation is based on the blueprint provided by *sp*, the search space for reasoning is greatly reduced, making it a feasible solution to significantly reduce the composition efforts.

Figure 10 shows the architecture of our automated instantiation system. The specifications of both service patterns and concrete services in the system are represented as facts and stored in SP repository. When a developer wants to build a target service *ts*, she/he first specifies *ts*.IOPE, which is converted into facts on the fly and stored in the fact working memory.

We used Jess v7.1p2 as the rule engine and employ the forward-chaining method in the automated instantiation reasoning process. In Jess, some rules have to consider the related facts as a whole but each fact can only trigger one rule at a time. Therefore, we divide the reasoning process into three stages to gradually build the needed facts.

First, the matchmaking rules are applied and the set of concrete services that matches each prospect service in the pattern is identified, resulting in a number of facts added into the fact working memory. Second, for each potential set of service selections, all the customizable data type parameters declared in message alignment constraints are examined against the data consistency rule. If it is confirmed, a valid message flow assertion will be put into the working memory. Finally, the condition of feasible assignment is checked to verify whether an assignment is feasible. The three stages and corresponding inference processes are introduced in the following sections.

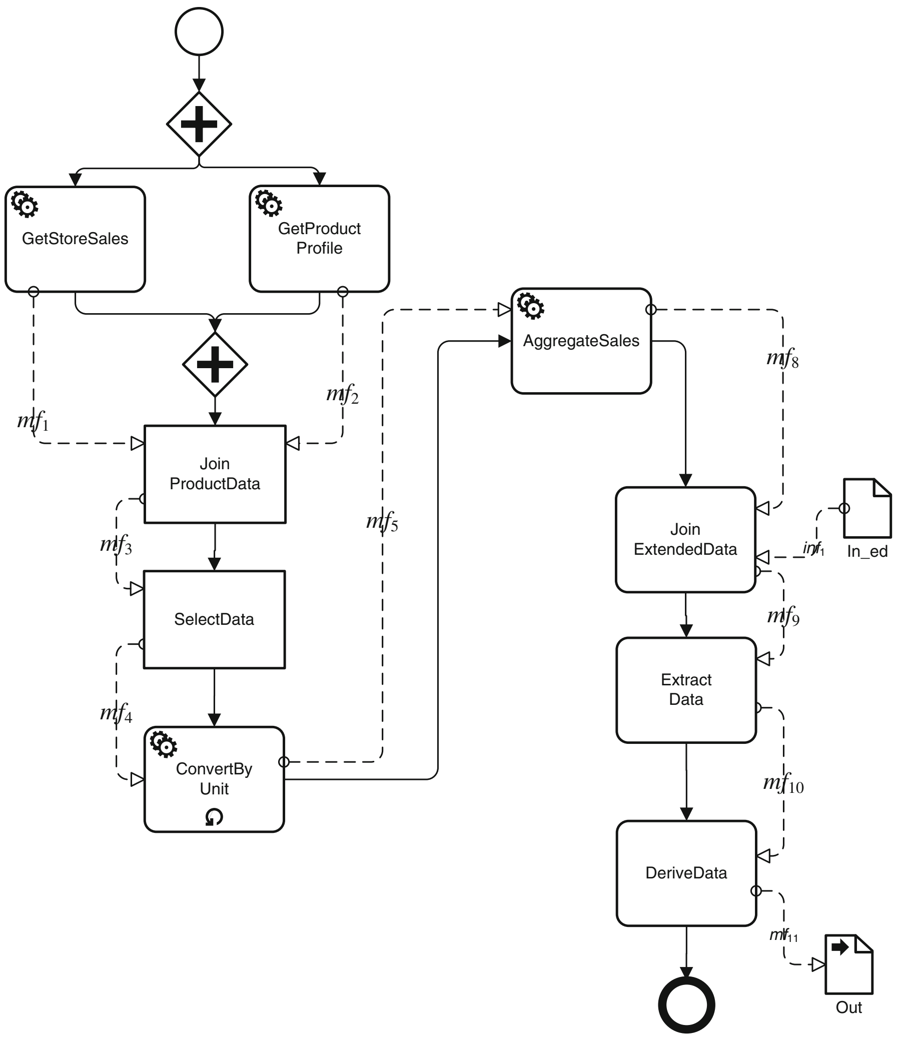

Fig. 9 The customized workflow for RPW

5.2 Matching rules in the first stage

The purpose of the first stage is to concretize nondeterministic IOPE of the prospect services with assuring that the concretized specification conforms to restriction rules on data type parameters and their precondition/effect specifications. We define two rules to achieve the goal.

The first rule is to find out candidate data types for the nondeterministic I/O types of each prospect service by referring to potentially matched services, which have the same number of arguments as the prospect service and their I/O types match the fixed I/O types of prospect service. Then, the candidate data types are obtained from

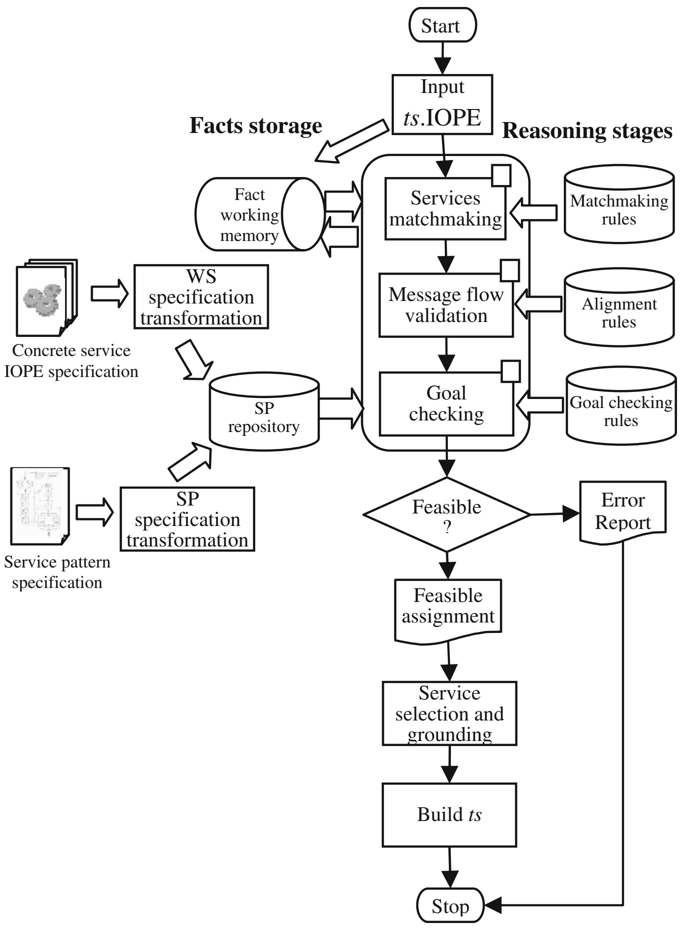

Fig. 10 The architecture of automated assignment reasoning

unmatched I/O part of potentially matched services. For example, in Fig. 11 *CombineExtendedData* prospect service has two nondeterministic I/O types, ?TypeOfEd and ?TypeOfJedOut. Suppose that there are two potentially matched services, *CombineStoreExpense* and *JoinProductInv*, which have the same number of arguments and the same concrete data types, i.e., TSalesData in this example. Accordingly, there will be two assignments for the data type parameters ?TypeOfED and ?TypeOfJedOut, namely TExpense and TRevenuExpense from *CombineStoreExpense*, and TProdInv and TSalesQty from *JoinProductInv*.

The second rule is to confirm the satisfaction of the restriction rules given in the service pattern. If the candidate data types obtained from applying the first rule comply with all corresponding restrictions, the second rule will insert type assignment facts into the fact working memory. After the above reasoning, one nondeterministic I/O type may be assigned with several specific data types. We will

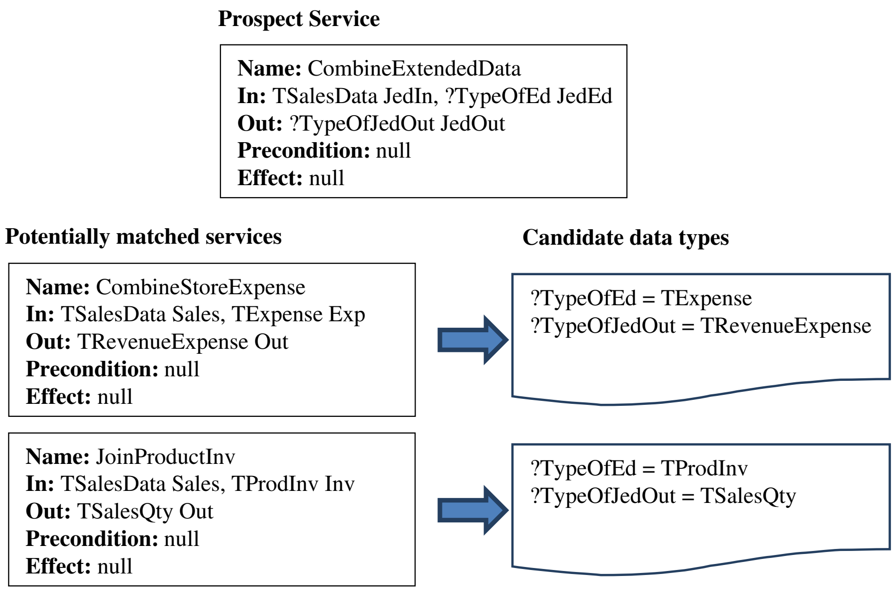
Prospect Service
Name: CombineExtendedData
In: TSalesData JedIn, ?TypeOfEd JedEd
Out: ?TypeOfJedOut JedOut
Precondition: null
Effect: null
Potentially matched services
Candidate data types
Name: CombineStoreExpense
In: TSalesData Sales, TExpense Exp
Out: TRevenueExpense Out
Precondition: null
Effect: null
?TypeOfEd = TExpense
?TypeOfJedOut = TRevenueExpense
Name: JoinProductInv
In: TSalesData Sales, TProdInv Inv
Out: TSalesQty Out
Precondition: null
Effect: null
?TypeOfEd = TProdInv
?TypeOfJedOut = TSalesQty

Fig. 11 Candidate data type deviation for an example prospect service

further filter out incompatible assignments due to the violation of message alignment verified in the next stage, which will be described in the next subsection.

In addition to the above rules, we also defined other rules that consider the facts converted from the goal, such as the values of alternative selections and skippable toggles specified by developer. Those rules assert new facts that represent the employment of some prospect services according to developers' choice and may trigger the first rule.

5.3 Alignment rules in the second stage

In the second stage, we validate message alignments by checking whether data types of connected parameters are compatible. We define three rules for this task:

rule 3: testing connected parameters of each message alignment for matched type,
rule 4: checking a sequence of message flow facts for consistent message alignments, and
rule 5: removing false type assignment facts.

A prospect service may have several type assignment facts for its nondeterministic I/O data types because there could be several potentially matched services with different data types. Rule 3 considers the message flow facts obtained from message alignments of service pattern and the type assignment facts asserted in the first stage. If data types of source and destination parameters in message flow fact are compatible, the rule will create a valid message flow fact that asserts the target message flow connecting two cooperative service instances.

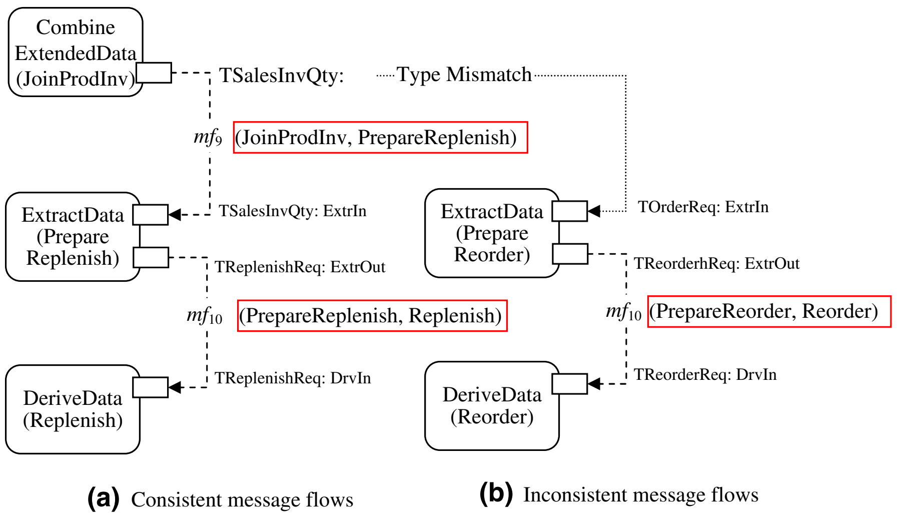
Combine ExtendedData (JoinProdInv)
TSalesInvQty:
Type Mismatch
mf9
(JoinProdInv, PrepareReplenish)
ExtractData (Prepare Replenish)
TSalesInvQty: ExtrIn
TReplenishReq: ExtrOut
mf10
(PrepareReplenish, Replenish)
TReplenishReq: DrvIn
DeriveData (Replenish)
ExtractData (Prepare Reorder)
TOrderReq: ExtrIn
TReorderhReq: ExtrOut
mf10
(PrepareReorder, Reorder)
TReorderReq: DrvIn
DeriveData (Reorder)
(a) Consistent message flows
(b) Inconsistent message flows

Fig. 12 The graphic presentation of mf_9 and mf_{10} message flow facts

Even though a valid message flow fact is asserted, it may become false when it is inconsistent with its preceding or succeeding valid message flow facts. For example, consider the execution of a sequence of prospect services, *CombineExtendedData*, *ExtractData*, and *DeriveData* in *SalesCube* service pattern in Fig. 6; there are two message alignments, mf_9 and mf_{10}, that connect source and destination parameters between two adjacent prospect services. Suppose that each of *ExtractData* and *DeriveData* prospect services has two potentially matched services: *PrepareReplenish* and *PrepareReorder* for *ExtractData* and *Replenish* and *Reorder* for *DeriveData*; and *CombineExtendedData* prospect service maps to *JoinProdInv* concrete service. After applying the first rule as mentioned before, for message alignment mf_{10}, there are two valid message flows: (*PrepareReplenish*, *Replenish*) and (*PrepareReorder*, *Reorder*). On the other hand, for message alignment mf_9, there is only one valid message flow, (*JoinProdInv*, *PrepareReplenish*), because the output parameter JedOut of *JoinProdInv* service is incompatible with the input parameter ExtrIn of *PrepareReorder* service. Their relationships are graphically displayed in Fig. 12, where the corresponding concrete services are shown in parentheses and, three valid message flow facts are shown in red box beside the message alignments. As can be seen, the sequence of valid message flows, (*JoinProdInv*, *PrepareReplenish*)-(*PrepareReplenish*, *Replenish*), is the only consistent message flows when considering both mf_9 and mf_{10} message alignments. Therefore, we conduct consistency checking between message flows in rule 4 by examining the consistency of destination service instance and source service instance for every message flow. The message flows validated by rule 4 are asserted as valid message flows whereas the other flows are claimed as false message flows. Finally, rule 5 removes all of type assignment facts involving false message flows.

5.4 Goal checking rule in the third stage

An assignment is feasible if and only if all specified data types satisfy restriction rules, all message alignments are validated, or every prospect service has some matched concrete service. Therefore, in the final stage, we define a rule by which all facts deduced in previous stages are checked for a feasible assignment. The antecedent of this rule includes the validity checking of all of message alignments, alternative selections, and skippable toggle and the assurance that all of prospect services are concretized with valid data types.

In case all of the conditions fit, the assignment for instantiation parameters is pronounced to be feasible. However, the complete assignment of instantiation parameters cannot be directly extracted from matched concrete services because there may be several matched concrete services for a prospect service, but only the matched instances conforming to the message alignments is a correct binding. Therefore, we develop the extraction function to print out a set of complete assignments for all instantiation parameters. Indeed, there may exist several complete assignments and how to choose a "good" set of assignments is beyond the scope of this paper.

6 Model evaluation

We conducted an empirical study to evaluate the effectiveness of the proposed service pattern model. Subjects are selected to participate in the composition processes of new composite services. The scenarios are based on land management applications used by city governments in Taiwan, including land registration, land rights transferring, and land division. Composition time and accuracy are two dependent variables in the evaluation process. The service pattern model is compared with two other methods: *creation from scratch* and *creation using similar service composition*. The details of the experimental design and the results are described in the following subsections.

6.1 Experiment design

We conducted a two-factor experiment with two dependent variables per subject, and the two variables were experimental problems and methods for creating composite services. Three service composition problems with different difficulty levels were designed. The methods include manual composition from scratch, M1, manual composition by adapting similar service composition, M2, and using service pattern, M3. To solve a given problem using M1, each subject was provided with description of candidate Web services and was asked to manually compose these services to solve the problem. For M2, in addition to the description of candidate Web services, each subject was given the service composition of a similar application to help him/her manually compose a process for the experimental problem. Finally, for M3, each subject was offered a service pattern from which a composition can be instantiated.

We recruited 18 subjects who had more than 4 years of industry experience in information systems design. All subjects had not worked on land management domain, and no particular subjects would benefit from prior knowledge. After one hour training of BPMN and our proposed model, subjects were asked to design three compositions for three experimental problems. In addition, these experimental problems were shown to have (statistically significantly different levels of difficulties by one-way ANOVA on the subjects' ratings. To measure the effect, we recorded the time spent by the subjects in designing the composite service for the experimental problems. Also, a domain expert was invited to evaluate the accuracy of the subjects' service compositions in the range of 0–100.

6.2 Data analysis and discussion

Figures 13 and 14 show the average composition time and the average accuracy of the three methods across different experimental problems respectively, where P3 is more difficult than P2, which in turn is more difficult than P1. It can be seen that the proposed method M3 has the lowest mean composition time and highest mean accuracy across all the three problems. From Fig. 13, we can see that the composition time using M3 does not change significantly across problems of different difficulty levels. This is attributable to the service pattern that provides a good design practice and alternative features to the subjects. It can reduce the effort of designing workflow and checking design validity. On the other hand, for M1, the composition time grows when the difficulty level of problem increases. The composition time for using M2 also remains approximately the same for different problems. This shows that composition by adapting a similar workflow is also effective. However, the average composition time using M2 is higher than using M1. Now we consider composition accuracy as shown in Fig. 14. M3 again achieves a similar accuracy level for all problems. It implies that service pattern can help achieve stable design quality. Whereas, the design quality of M2 could be affected by the suitability of the similar workflow selected for the problem and its accuracy is unstable. We also performed the two-factor ANOVA to test the composition time and composition accuracy in groups after assuring that the variance of data in groups is homogeneous via Levene's test. From the ANOVA tables of composition time and accuracy, the effects of problem factor and problem*method interaction are not significant and method effect is significant at .05 level. Their *F* and *p* values are shown in Table 2.

We then conduct LSD post hoc tests for two dependent variables with the method factor and the result is shown in Table 3. For composition time, M3 significantly outperforms M1 at .01 level, yet it only outperforms M2 at .2 significance level. For accuracy, M3 still significantly outperforms M1 at .01 level, and it is better than M2 only at .1 significance level. In addition to the obviously smaller sample size, our interview with the participants reveals that most people are used to adapt existing service composition for new requirement but they are slightly undertrained about service pattern model with only one hour of training. If they had received more practices about service pattern, both the composition time and the accuracy could have been improved further.

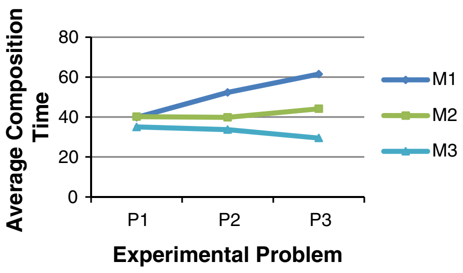

Fig. 13 Average composition time comparison over different methods

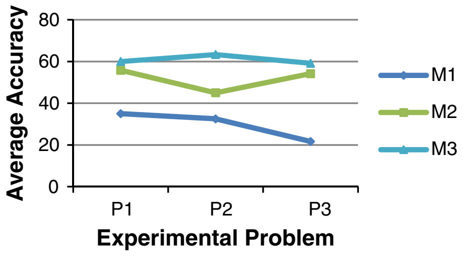

Fig. 14 Average accuracy comparison over different methods

Table 2 Simplified ANOVA tables of duration and accuracy

	F	*p*
Factors of composition time (α = 0.05)		
Problem	0.487	.618
Method	3.685	.033*
Problem × method	0.705	.593
Factors of accuracy (α = 0.05)		
Problem	0.456	.637
Method	16.344	.000*
Problem × method	0.722	.581

Asterisks indicate statistical significance at $p < 0.05$

7 The performance evaluation of automated instantiation parameter assignment

To evaluate the performance of the proposed reasoning method, we conduct three experiments to reveal the execution times under different numbers of candidate Web services, message alignments, and prospect services. We use *SalesCube* service pattern and adopt the requirements of RPW in our experiments. Specifically, the assignment given by the developer involves only RPW's IOPE and workflow adapting parameters. Moreover, we observe that more message alignments or prospect services, as created by message alignment and prospect services, may consume more time. To examine the impact of message alignment

Table 3 The result of LSD post hoc comparisons for duration and accuracy with method factor

Dependent variables	M_I	M_J	Mean difference ($M_I - M_J$)	Std. error	Sig.
Composition time	1	2	9.83	6.819	.156
		3	18.50*	6.819	.009*
	2	1	−9.83	6.819	.156
		3	8.67	6.819	.210
	3	1	−18.50*	6.819	.009*
		2	−8.67	6.819	.210
Accuracy	1	2	−21.94*	5.592	.000*
		3	−31.11*	5.592	.000*
	2	1	21.94*	5.592	.000*
		3	−9.17	5.592	.108
	3	1	31.11*	5.592	.000*
		2	9.17	5.592	.108

Asterisks indicate significance level

and prospect service on the performance of automated assignment reasoning, we reduce the numbers of message alignments and prospect services in the second and third scenarios respectively, and compare their reasoning time with the first scenario. Therefore, two additional scenarios with different modifications are given to these experiments and they are summarized in Table 4. These experiments are conducted on a PC with an Intel Xeon 2.33 GHz CPU. The reasoning process is iterated over 100 times for each case with different numbers of candidate Web services for each prospect service, and the average execution time is reported.

In the first experiment, 3 prospect services and 17 message alignments are specified, and its results are shown in Fig. 15. It can be seen that the execution time is almost linear to the number of candidate Web services. As the execution time of reasoning process is sensitive to the number of facts, we show in Fig. 16 the number of facts in the reasoning engine across different number of candidate Web services. It shows that the growth of the numbers of facts is linear to the number of candidate Web services, which explains the results of Fig. 15.

In the second experiment, we reduce the number of message alignments to nine while maintaining the same number of prospect services in *SalesCube* service pattern. The performance comparison between the first and second scenarios is shown in Fig. 17. In Fig. 17, the execution time of the second scenario is slightly less than that of the first scenario under different number of candidate Web services. We further conduct Welch's t test, shown in Table 5, which confirms that time difference between the first and second scenarios is significant with their variances being heterogeneous in Levene's test. The reason is that fewer message alignments result in lower loading of message flow validation and consistency checking. We further analyze the performance differences of the first and second scenarios at different inference stages using the case with 350 candidate Web services. Figure 18

Table 4 The modifications and initial setting for three scenarios

Scenario	Modifications	Initial Setting
1st scenario: 17 message alignments (MFs) and 3 prospect services (PSs)	Use SalesCube service pattern	?TypeOfED = TProdInv; ?TypeOfOut = TOrder; ?SkipConverByCurrency = True; ?SkipConvertByUnit = False; ?SkipDeriveData = False; ?SkipAggregateData = True; ?ChooseFormat = 'Cube'; ?ChooseComputeOrder='DeriveFirst';
2nd scenario: 9 message alignments and 3 prospect services	Use SalesCube service pattern but remove mf_1 to mf_8 message alignments	Same as the first scenario
3rd scenario: 17 message alignments and 1 prospect services	Use SalesCube service pattern but change *CombineExtendedData* and *ExtractData* as two concrete services	Same as the first scenario

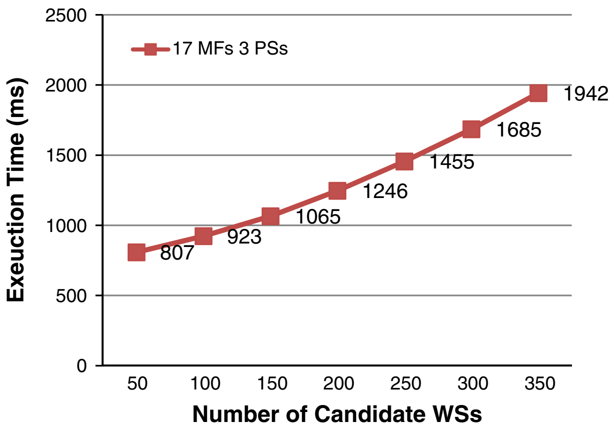

Fig. 15 The execution time under different numbers of concrete Web services for each prospect service

shows that the inference stages related with message alignment, namely MF validation and consistency checking, have wider performance gap than other stages. Especially, at consistency checking stage, the second scenario has significantly higher performance improvement as fewer message alignments considerably reduce the depth of traversing message flows. However, in all other stages, the performance differences are really minor. This explains the slight performance improvement with fewer message alignments.

In the third experiment, we concretize two prospect services and leaves only one prospect service while maintaining seventeen message alignments in *SalesCube*

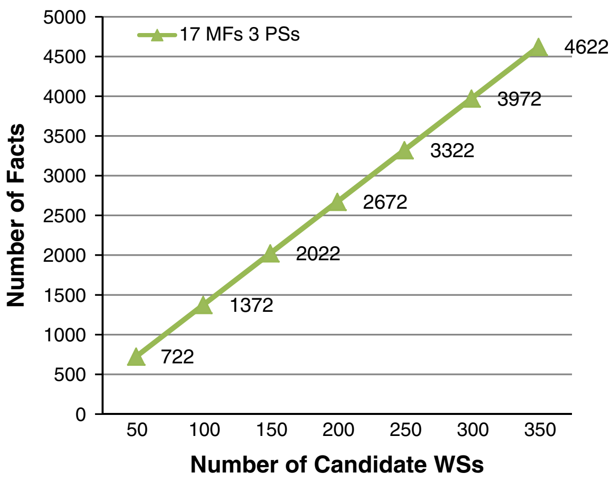

Fig. 16 The number of facts under different numbers of concrete Web services for each prospect service

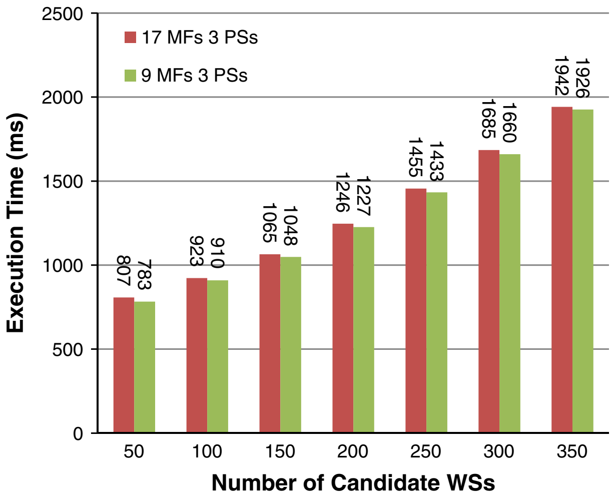

Fig. 17 The performance comparison between the first and second scenarios

service pattern. The performance comparison between the first and third scenarios is shown in Fig. 19. It can be seen that the third scenario outperforms the first scenarios as the third scenario has fewer prospect services, which result in fewer service matchmaking facts and valid message flow facts. We further conduct Welch's t test, shown in Table 6, which confirms that time difference between the first and third scenarios is significant with their variances being heterogeneous in Levene's test.

Table 5 The result of Welch's t test for the first and second scenarios

Number of candidate WSs	Degree of freedom for scenarios	Degree of freedom for error	F value	Significance
50	1	131.099	169.933	.000
100	1	162.494	59.241	.000
150	1	215.445	300.715	.000
200	1	215.912	350.406	.000
250	1	215.395	420.106	.000
300	1	213.778	513.660	.000
350	1	214.901	213.654	.000

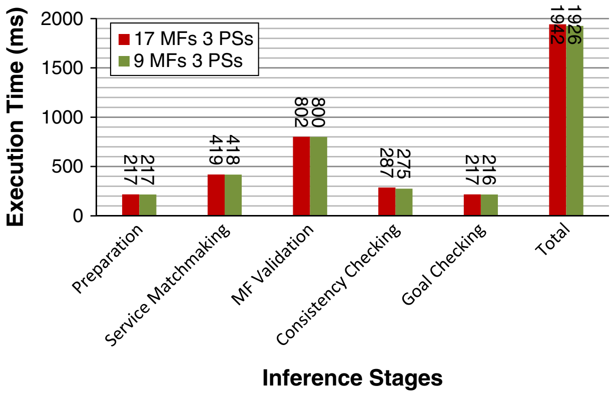

Fig. 18 The performance comparisons between the first and second scenarios at different inference stages for the case with 350 candidate Web services

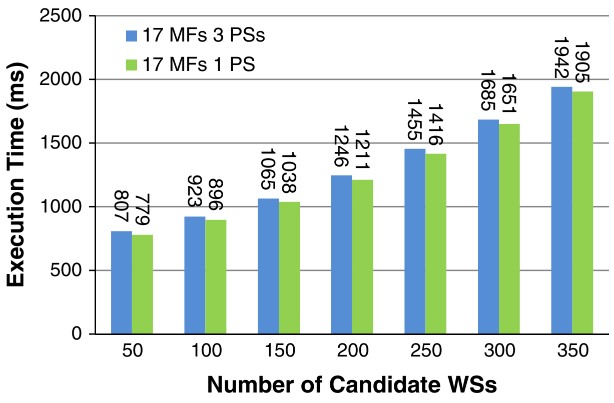

Fig. 19 The performance comparison between the first and third scenarios

Table 6 The performance comparisons between the first and second scenarios at different inference stages for the case with 350 candidate Web services

Number of candidate WSs	Degree of freedom for scenarios	Degree of freedom for error	F value	Significance
50	1	139.418	212.257	.000
100	1	215.640	568.136	.000
150	1	198.657	531.364	.000
200	1	210.839	941.348	.000
250	1	215.736	1,222.012	.000
300	1	203.642	845.442	.000
350	1	211.901	1,401.561	.000

8 Conclusion

Agility of enterprise receives much attention in recent years because businesses today have to face dynamic and changing markets in an unprecedented way. SOA technologies are critical means to improve agility of enterprise. The realization of SOA entails the reusing and composing of existing Web services. In particular, flexible Web service composition for a variety of applications enhances adaptability and flexibility of information systems in support of enterprise agility. As mentioned in Sects. 1 and 2, the existing CASE tools in IT industry and automated Web service composition methods are not sufficient in providing flexible strategies of composing Web services.

To address this issue, we proposed a meta-model, namely service pattern model, to model an adaptable workflow template capable of embedding variations for a variety of applications. In addition to prevalent variability management in SOA, the unique features of service pattern model include prospect service and adaptable workflow template. A prospect service has customizable data types for I/O parameters and/or customizable predicates for preconditions and effects in its nondeterministic specification. By concretizing its customizable part, a prospect service can be instantiated into different types of abstract services according to different requirements. Accordingly, an adaptable workflow template can be built into a variety of applications because of the functional flexibility of prospect services. The provision of message alignment and restriction rule in adaptable workflow template supports message compatibility checking between cooperative services and data type validation within services. These checking actions confirm the validation of customizable data type assignment and smooth out data exchange between cooperative services. Instantiation parameters provide a customizable mechanism by which information system developers can build different composite Web services with different functionalities from a service pattern by specifying instantiation parameters. Furthermore, we develop a set of reasoning rules capable of automatically inferring the values of instantiation parameters. These rules are divided into three groups and executed separately in three stages.

We have conducted an empirical study to evaluate the performance of service pattern model by using composition time and accuracy as performance metrics. From experimental results, we can see that the service pattern approach shortens composition time and improves composition accuracy over the manual composition method. Also, we conducted three experiments to evaluate the performance of automated reasoning process. The results show that execution time is almost linearly proportional to the number of candidate Web services and the number of prospect services and message alignments also slightly affect the performance of automated reasoning method due to the loads of message flow validation and consistency checking.

In practice, service pattern model and WS-data model can assist information system developers, including business analysts and system designers, in efficiently and effectively developing information systems using SOA. The development process based on our proposed service pattern model can be separated into two phases: service patterns design and service compositions construction. In the first phase, system designers construct function-flexible service patterns. At the second phase, business analysts elicit the business requirements and build concrete service composition for the requirements by employing automatic reasoning method on the service patterns. While there have been many methodologies proposed for manually developing a business process, they are not applicable to the design and instantiation of service patterns. Our ongoing work includes the design of such a methodology.

Acknowledgments This work was supported in part by the National Science Council in Taiwan under Grant NSC 101-2410-H-110-015-MY2.

References

Abu-Matar M, Gomaa H (2011) Feature based variability for service oriented architectures. In: 2011 9th working IEEE/IFIP conference on software architecture, 2011. pp 302–309

Akkiraju R, Srivastava B, Ivan AA, Goodwin R, Syeda-Mahmood T (2006) SEMAPLAN: combining planning with semantic matching to achieve Web service composition. In: IEEE international conference on Web services (ICWS 2006), 2006. pp 37–44

Amarouche I, Benslimane D, Barhamgi M, Mrissa M, Alimazighi Z (2011) Electronic health record data-as-a-services composition based on query rewriting. In: Hameurlain A, Küng J, Wagner R, Böhm C, Eder J, Plant C (eds) Transactions on large-scale data- and knowledge-centered systems IV, vol 6990. Lecture Notes in Computer Science. Springer, Berlin, pp 95–123. doi:10.1007/978-3-642-23740-9_5

Anonymous (2007) An ontology to describe the achema information of a relational database. http://www.dbs.cs.uni-duesseldorf.de/RDF/relational.owl. Accessed 18 March 2011

Barhamgi M, Benslimane D, Medjahed B (2010) A query rewriting approach for Web service composition. IEEE Trans Serv Comput 3(3):206–222. doi:10.1109/tsc.2010.4

Chen K, Xu J, Reiff-Marganiec S (2009) Markov-HTN planning approach to enhance flexibility of automatic Web service composition. In: IEEE international conference on Web services (ICWS 2009), 2009. IEEE, pp 9–16

Cummins F (2008) Building the agile enterprise: with SOA, BPM and MBM. Morgan Kaufmann, Burlington

Döhring M, Zimmermann B (2011) vBPMN: event-aware workflow variants by weaving BPMN2 and business rules. In: the 16th international conference on exploring modelling methods for systems analysis and design (EMMSAD'11), 2011/01/01 2011. Lecture Notes in Business Information Processing. Springer, Berlin, pp 332–341. doi:10.1007/978-3-642-21759-3_24

Döhring M, Reijers HA, Smirnov S (2014) Configuration vs. adaptation for business process variant maintenance: an empirical study. Inf Syst 39:108–133. doi:10.1016/j.is.2013.06.002
Doshi P, Goodwin R, Akkiraju R, Verma K (2004) Dynamic workflow composition using Markov decision processes. Int J Web Serv Res 2(1):576–582
Erl T (2005) Service-oriented architecture: concepts technology and design. Prentice Hall PTR, Upper Saddle River
Erl T (2007) SOA principles of service design. Prentice Hall PTR, Upper Saddle River
Fu JC, Bastani FB, Yen IL, Hao W (2009) Using service patterns to achieve Web service composition. In: 2009 IEEE international conference on semantic computing, 2009. pp 402–407
Geebelen K, Michiels S, Joosen W (2008) Dynamic reconfiguration using template based Web service composition. In: the 3rd workshop on middleware for service oriented computing, 2008. pp 49–54
Gil Y, Ratnakar V, Kim J, Gonzalez-Calero P, Groth P, Moody J, Deelman E (2011) Wings: intelligent workflow-based design of computational experiments. IEEE Intell Syst 26(1):62–72
Gottschalk F, Van Der Aalst WMP, Jansen-Vullers MH, La Rosa M (2008) Configurable workflow models. Int J Coop Inf Syst 17(02):177–221. doi:10.1142/S0218843008001798
He Q, Yan J, Jin H, Yang Y (2008) Adaptation of Web service composition based on workflow patterns. In: Bouguettaya A, Krueger I, Margaria T (eds) 6th international conference on service-oriented computing (ICSOC 2008), 2008. Lecture Notes in Computer Science. Springer, Berlin, pp 22–37. doi:10.1007/978-3-540-89652-4_6
Hwang SY, Hsieh YH, Lee CH (2012) Data providing Web service selection using Bayesian network. In: 2012 IEEE ninth international conference on e-business engineering (ICEBE'12), 2012. pp 111–116
IBM (2011) IBM business process manager. http://www-01.ibm.com/software/integration/business-process-manager/. Accessed 21 June 2011
Kapuruge M, Jun H, Colman A (2010) Support for business process flexibility in service compositions: an evaluative survey. In: 2010 21st Australian software engineering conference (ASWEC), 6–9 April 2010. pp 97–106
Kumar A, Yao W (2012) Design and management of flexible process variants using templates and rules. Comput Ind 63(2):112–130. doi:10.1016/j.compind.2011.12.002
Lee CH, Hwang SY (2009) A model for Web services data in support of Web service composition and optimization. In: 2009 world conference on services—I, 6–10 July 2009. pp 384–391
Medjahed B, Bouguettaya A, Elmagarmid AK (2003) Composing Web services on the semantic Web. VLDB J 12(4):333–351. doi:10.1007/s00778-003-0101-5
Mietzner R, Leymann F (2008) Generation of BPEL customization processes for SaaS applications from variability descriptors. In: 2008 IEEE international conference on services computing (SCC 2008), 7–11 July 2008. pp 359–366
Nguyen T, Colman A, Han J (2011a) Modeling and managing variability in process-based service compositions. In: the 9th international conference on service-oriented computing (ICSOC), 2011a. pp 404–420
Nguyen T, Colman A, Talib MA, Han J (2011b) Managing service variability: state of the art and open issues. In: 5th workshop on variability modeling of software-intensive systems. pp 165–173
Oracle (2011) Oracle business process management suite. http://www.oracle.com/us/technologies/bpm/bpm-suite-078529.html. Accessed 21 June 2011
Pistore M, Traverso P, Bertoli P (2005) Automated composition of Web services by planning in asynchronous domains. In: the 15 international conference on automated planning and scheduling, 2005. pp 2–11
Pohl K, Böckle G, Linden F (2005) Software product line engineering, foundations, principles, and techniques. Springer, Berlin
Ponnekanti SR, Fox A (2002) SWORD: a developer toolkit for Web service composition. In: the 11th international conference on World Wide Web (WWW 2002)
Ruokonen A, Raisanen V, Siikarla M, Koskimies K, Systa T (2008) Variation needs in service-based systems. In: 2008 IEEE sixth European conference on Web services (ECOWS 2008), 12–14 Nov. 2008. pp 115–124
SAP (2011) SAP NetWeaver business process management. http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx. Accessed 17 March 2011
Sirin E, Parsia B, Wu D, Hendler J, Nau D (2004) HTN planning for Web service composition using SHOP2. Web Semant 1(4):377–396
Smirnov S, Reijers H, Weske M, Nugteren T (2012) Business process model abstraction: a definition, catalog, and survey. Distrib Parallel Databases 30(1):63–99. doi:10.1007/s10619-011-7088-5

Yang L, Dai Y, Zhang B (2009) Business-pattern-wvolution based service composition with flexibility. In: Sixth Web information systems and applications conference, 2009. IEEE, pp 132–135

Zeng L, Ngu A, Benatallah B, Podorozhny R, Lei H (2008) Dynamic composition and optimization of Web services. Distrib, and Parallel Databases 24(1):45–72