Service Composition with Functional Flexibility Using Nondeterministic Service Interface

Chien-Hsiang Lee, San-Yih Hwang
Department of Information Management
National Sun Yat-sen University
Kaohsiung, Taiwan 80424, syhwang@mis.nsysu.edu.tw

I-Ling Yen
Computer Science Department
University of Texas at Dallas
Richardson, United States, ilyen@utdallas.edu

Abstract—**Although previous works pertaining to Web service composition have paved the way for resolving the automatic composition issue with considering single IOPE goal, functional flexibility of service composition with complex workflow is not well-addressed. In this paper, we propose a novel service type, namely prospect service that possesses nondeterministic service interface beyond concrete and abstract services used by current service composition methods, to improve functional flexibility of service composition. Furthermore, we define a meta-model including prospect service and adaptable workflow template to provide comprehensive constructs for modeling flexible and adaptable workflow templates while developing an automatic instantiation method to instantiate concrete workflows with different functionalities from a template. Several experiments are also conducted to evaluate the performance of the automated method under different contexts.**

Keywords-Web service composition; service pattern; meta-model

I. INTRODUCTION

Web service compositions leverage existing Web services to quickly form new services in order to fit new requirements and improve the agility of businesses. There are CASE tools that help developers build service compositions, such as SAP NetWeaver. However, in practice, business process analysts still manually plan the workflow and then consult IT staff to deploy workflow-based Web service compositions, which demands considerable efforts.

To reduce the effort of service composition, AI planning and logical reasoning methods [2, 7, 9-11] have been adopted to automatically compose services into a new service composition that satisfies the desired inputs and outputs. However, scalability issue and limited I/O requirements confine them to narrow applications.

On the other hand, workflow templates approach has been investigated in the literature to help build complex workflow in a service composition. Generally workflow templates are in a form of abstract workflows, where services in the workflow can be concrete or abstract. An abstract service has well defined I/O data types, precondition, and effects, yet is not executable. It should be grounded to concrete services before it is executable.

Some works [3, 4] 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. Although these workflows are adaptable in QoS, they still lack 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.

To increase the reusability of workflow templates, it is desirable to also introduce functional flexibility. In other words, it is desirable that the IOPE definition of a workflow template is nondeterministic so that it can be instantiated into different abstract workflows with fixed IOPEs and subsequently grounded into different concrete workflows. Here we call such workflow templates that not only offer flexibility in achieving different QoS goals but also are flexible in their functionality *the service patterns*.

Often times, there are workflows that possess similar structure but are different in their functionalities. With service pattern model, it is possible to express them by one service pattern. More importantly, providing a service pattern offers the possibility of constructing a wide range of different abstract workflows that cannot be enumerated by individual workflow templates. With the popularity of SaaS, many providers are now building platforms to allow users to compose services to fulfill the desired tasks of the customers. To overcome the difficulties in service composition, the service patterns will offer great reduction in the time and efforts required for the service composition task. It also provides high configurability that is the most important feature of SaaS.

Service patterns that are flexible in functionality raise new issues that have not been considered in the literature. Some workflow template (or service pattern) related works consider flexible control flows to increase the flexibility of the templates in order to achieve different functionalities, such as optional/alternative feature choices [1, 8]. However, although flexible control flow in service patterns increases its functional flexibility, it also creates the data type definition problem that has not been seriously considered. For example, consider a workflow with two alternative paths. If the output data types derived from the two paths are different, then the service, say *x*, receiving the output from one of these two potential paths may need to have alternate input data type definitions. Existing literature does not consider this potential issue. They may implicitly assume that two alternative paths have to have the same output data types.

To accommodate the potential of having different output data types in alternate paths, a service that receives one from several potential outputs has to have flexible input data type definition. A similar issue presents when the output of a service flows into two alternative paths. As a result, the individual services nodes in the service pattern will need to have flexible IOPEs to yield the desired flexible IOPE of a service pattern. Thus, we propose the concept of *prospect*

978-0-7695-5111-1/13 $26.00 © 2013 IEEE
DOI 10.1109/ICEBE.2013.67

services to fulfill the goal. A prospect service is a component service in a service pattern with nondeterministic functional specifications, i.e., partially defined IOPE. In the concretization process, a service pattern is first instantiated into an abstract service with specific IOPE definitions. Then, the abstract service is grounded into a concrete service.

With the power of prospect services and flexible control flows, a service pattern can achieve significant functional flexibility and, hence, enhancing reuse without sacrificing its benefit of facilitating easy service compositions. There are some issues that need to be addressed in order to support such service patterns. First, new modeling techniques and specification mechanisms are required to support the definition of prospect services and corresponding service patterns, called service pattern model in this work. Second, as discussed earlier, the alternatives in control flows in a service pattern need to be considered more carefully. Essentially, the services passing inputs into or receiving outputs from alternative control flow paths need to have nondeterministic IO data types that will be concretized during instantiation, i.e., they should be prospect services. Third, the definition of a prospect service in a service pattern may have impact on prior and/or subsequent service nodes. Consider a prospect service *x* in a service pattern. Assume that *x* has a partially defined output data type and it passes data to another service *y*. It is only meaningful for *y* to also be a prospect service with at least partially defined input data type. Otherwise, if *y*'s input data type is fixed, then it confines *x*'s output data type and there is little sense to define *x*'s output data type in a more flexible way. Thus, in order to achieve flexible functionality, the workflow may involve a series of prospect services and their IOPEs have to be properly aligned to correctly express the flexible IOPE of the service pattern.

In [6], a work-in-progress abstract, we have outlined the concept of service patterns and provided a short example to illustrate the concept. In this paper, we present our detailed solutions for service pattern specification, systematic instantiation of service patterns and automatic configuration of concrete service compositions based on service pattern. Our contributions include:

- We present a model for service pattern specification, which extends existing service models to achieve the specification of the flexible functionalities of prospect services and workflows.
- We outline a systematic instantiation process 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. Therefore, the service composition efforts can be reduced significantly.
- We integrate the rule-based reasoning technique into the service pattern model to automate the instantiation, service selection, and grounding process. A set of interference rules are defined to facilitate reasoning. This further reduces the efforts for service composition, which is not possible when using design patterns. Also, due to the use of service patterns, the problem space for reasoning is highly confined, reasoning a feasible and efficient solution in the composition process is possible.

The rest of this paper is structured as follows. In Section II, a motivating example is introduced to manifest the research problem. We then describe our proposed model and illustrate the model by using a service pattern example in Section III. Next, we describe the automatic instantiation process in Section IV and discuss our experimental results in Section V. Finally, Section VI concludes the paper.

II. A MOTIVATING EXAMPLE

Consider a common workflow in business organizations for analyzing store sales data in order to help make various business decisions. Often times, to support effective analysis, other database tables such as the product profile need to be joined to provide more information about sale items for different usages, e.g. safety stock for stock replenishment and unit cost for profit computing. From the joined table, specific data can be selected, then combined horizontally or aggregated vertically to obtain summarized results that support decision making, such as stock replenishment, profit computing, and order fulfillment. Additional considerations, such as currency and metric conversions are needed for international organizations. Also, combined data have to be transformed to comply with the input format of subsequent activities. 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 an example workflow, *SalesCube*, as shown in Figure 1.

In *SalesCube* pattern, GetStoreSales and GetProductProfile services obtain sales and product tables, which are subsequently joined into a combined sales and products table. The SelectRequiredData service then filters the table and selects data entries satisfying some given criteria (which can be a parameter of the pattern) from the table. ConvertByCurrency and ConvertByUnit are offered as skippable services to process monetary or quantitative

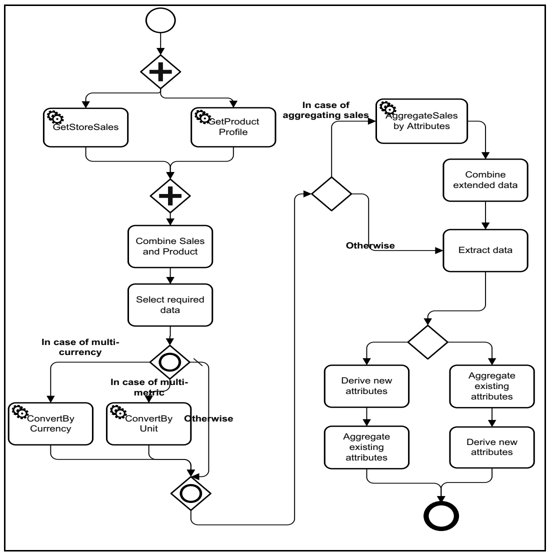

Figure 1. An example service pattern, *SalesCube*, in BPMN

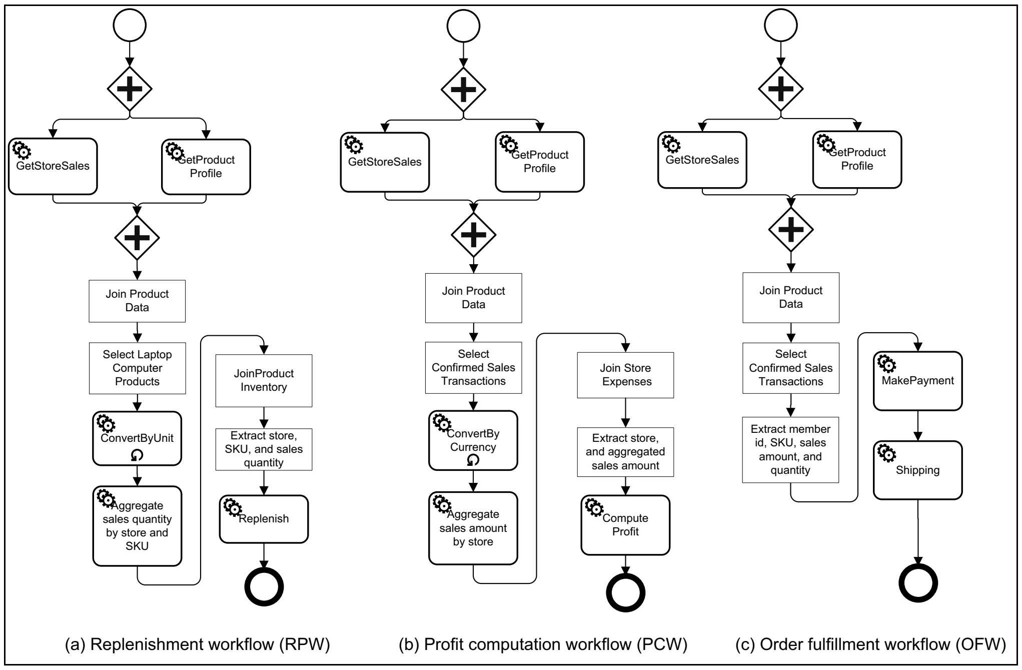
GetStoreSales
GetProduct Profile
Join Product Data
Select Laptop Computer Products
ConvertByUnit
Aggregate sales quantity by store and SKU
JoinProduct Inventory
Extract store, SKU, and sales quantity
Replenish
GetStoreSales
GetProduct Profile
Join Product Data
Select Confirmed Sales Transactions
ConvertBy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

Figure 2. Three workflow-based service compositions

columns in the table 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*, after accepting the data produced by prior services, aggregate the input attributes and derive new attributes respectively. For different purposes, these final services may be executed in different sequences as shown in exclusive choices in Figure 1.

The *SalesCube* pattern can be instantiated into several kinds of composite services with different functionalities. In Figure 2, three workflows instantiated from the *SalesCube* pattern are shown, including (a) the replenishment workflow (RPW) which derives the quantity of ordered items from their sales quantity, stock quantity, and safety stock; (b) the profit computation workflow (PCW) which calculates net profit from sales amount and store expenses in a given time period; and (c) an order fulfillment workflow (OFW) which aggregates customer's order amount, charges money for the order, and derives a delivery plan for shipping.

All three workflows share the same process in joining the tables. Then, a different selection criterion is used in each instantiation (by product category in RPW and by transaction state in PCW and OFW) to filter the table. *ConvertByCurrency* is then used in RPW and *ConvertByUnit* is used in PCW. Next, RPW and PCW aggregate sales in multi-dimensions (*AggregateSales*), and they obtain data from external sources and combine them into the data processed so far (*CombineExtendedData*). On the other hand, OFW simply uses original sales data processed so far. In the final stage, three workflows seem to have different tasks, yet they follow a common procedure: accepting partial sales data (e.g. sales amount and quantity), generating new attributes (e.g. order quantity in RPW and store profit in PCW), and aggregating original attribute (e.g. total up sales amount for each customer in MakePayment task of OFW).

III. The Service Pattern Model

Various standards, such as OWL-S and BPMN, have been developed 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 concepts "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 first define the two important service pattern elements, prospect services and adaptable workflow templates, with their corresponding instantiation parameters. In Subsection C, we formally define the service pattern.

A. Prospect Service

In OWL-S, input, output, precondition, and effects (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-defined IOPE constraints. The differences of concrete service, abstract service, and prospect service are summarized in Table I.

TABLE I. The differences among three types of services

Service Type	IOPE Spec.	Grounding Info.
Concrete service	Yes	Yes
Abstract service	Yes	No
Prospect service	Partial	No

A prospect service has nondeterministic IOPE. Its I/O messages 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. Both data type and predicate parameters are called instantiation parameters. In this paper, a variable name with prefix "?" denotes an instantiation parameter. The following is an example specification of a prospect service.

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

The prospect service, *DeriveData*, derives new attributes from existing attributes. Its input and output messages use nondeterministic data types, ?TypeOfDrvIn and ?TypeOfDrvOut, respectively. Restrictions can be specified for nondeterministic types using first-order logic, which can be a specific class in a domain ontology (in this case, all subclasses are legitimate instantiations of the data type) 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, using *modelReference* attribute proposed in SAWSDL

ModelRef(?TypeofDrvIn)=Ont#Table
ModelRef(?TypeofDrvOut)=Ont#Table

The concrete effect of *DeriveData* (the first three rows of the effect section) specifies (1) each output tuple has a corresponding input tuple and (2) each output tuple must be computed from one input tuple. Also, both the precondition and effect of *DeriveData* are also nondeterministic in service pattern and customizable during the instantiation process, having predicate parameters ?cpDeriveDataPrecond() and ?cpDeriveDataEffect(), respectively. At instantiation time, specific predicates can be given to concretize these predicate parameters and so that specific functionality for the service is determined. 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*:

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

B. Adaptable Workflow Template

We use BPMN as the basis to define the workflow within a service pattern and extend it with adaptable controls to specify flexible flow structures. Each constituent service in an adaptable workflow template can be concrete, abstract, or prospect. After instantiating a service pattern, its prospect services will be concretized into abstract or concrete services, its adaptable controls will be specialized, and the final workflow will only include conventional control flows. Here, we describe the flow control extensions for service patterns.

There are two adaptable control constructs: 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 of which is 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 specified in the workflow template is (not) skipped at instantiation time.

The control flow in a workflow defines the execution paths, but not the message exchange between component services. Generally, the output of a preceding service is fed to the input of the succeeding service. However, a service may have inputs from multiple preceding services and it is necessary to have a clear definition of data flow, i.e., source parameter and destination parameter 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 in existing models, we define a new construct, message alignment, to connect the I/O parameters of interacting services.

A message alignment is a tuple, $<s_s, s_d, v_s, v_d>$, where s_s

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
Constraints:
Precondition:
$|\text{In\_sl}| \neq 0 \wedge \text{Day(In\_tp)} < \text{CurrentDate()} - 7$
Effect:
$|\text{Out}| > 0$
Instantiation parameters:
Workflow adapting parameters:
?SkipConverByCurrency: Boolean,
?SkipConvertByUnit: Boolean,
…,
?ChooseFormat: {'Cube','Flat'},
?ChooseComputeOrder: {'AggregateFirst','DeriveFirst'},
Data type parameters:
?TypeOfEd, ?TypeOfJedOut, ?TypeOfExtrIn, …
Restriction:
ModelRef(?TypeofEd)=Ont#Table,
ModelRef(?TypeOfJedOut)=Ont#Table,
…,
KeyAttr(?TypeOfJedIn)=KeyAttr(?TypeOfed)=
KeyAttr(?TypeOfJedOut),
$\text{Attrs(?TypeOfJedOut)} \subseteq \text{Attrs(TSalesData)} \cup \text{Attrs(?TypeOfEd)}$
Predicate parameters:
?fpSelectData(TSalesData:SelIn),
?cpDeriveDataPrecond(?TypeOfDrvIn: DrvIn) ,
…

Figure 3. Partial IOPE specification and instantiation parameters of SalesCube service pattern in the proposed meta-model

and s_d are the source and destination services, respectively, and v_s and v_d are the formal output and input variables of source and destination services, respectively.

C. Service Pattern

A service pattern is specified by an adaptable workflow template and nondeterministic 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 can be defined by the tuple, $< I, O, P, E, IP, W_T>$, 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 W_T is an adaptable workflow template.

The interface definition for the example service pattern, *SalesCube* (given in Figure 1), is shown in Figure 3. *SalesCube* takes several concrete input parameters, including the store list (In_sl), the extracting attributes (In_ea), the period of data (In_tp), the target currency (In_cu), and the aggregating attributes (In_ga), as well as a nondeterministic input *In_ed*, which has a customizable data type ?TypeOfEd. The output parameter *Out* also has a customizable data type ?TypeOfOut. Both ?TypeOfEd and ?TypeOfOut are restricted to be of "Table" type, which is defined by a Table ontology. Due to the flexible input type to *SalesCube*, the I/O of all services in the workflow have customizable types as shown in the "data type parameters" section.

The adaptable workflow template of *SalesCube* is given in Figure 4 (in BPMN). It is similar to Figure 1 except that adaptable control constructs, WS-data operators, and message alignments are incorporated into it. We generalize the data handling services as a special type of services, which can accept customized XPath expression to handle SOAP message for message transformation [5]. Also, message alignments are added in Figure 4 to facilitate

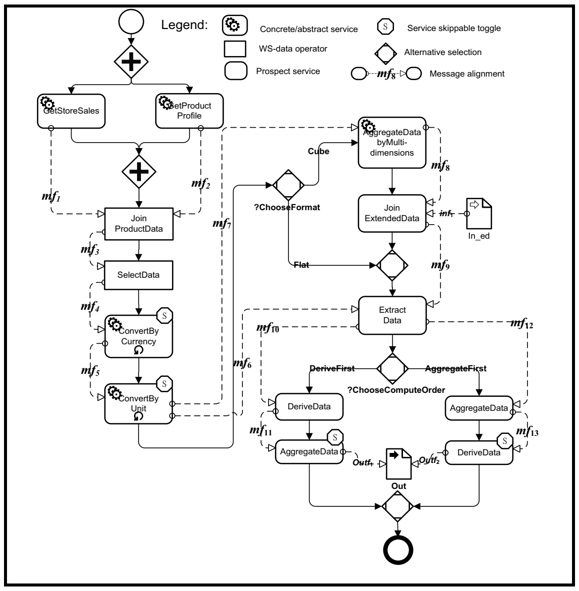

Figure 4. The adaptable workflow template of SalesCube service pattern

compatibility validation of the connected parameters after their types are concretized. Consider mf_{10} message alignment, it connects the output parameter of *ExtractData* service to the input parameter 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.

The instantiation of service pattern includes four steps: (1) The instantiation parameters are specified according to the application system requirement; (2) The workflow should be converted to the well-defined form by eliminating unselected alternative sub-flows and skipped services. Also, prospect services should be transformed into abstract services by instantiating nondeterministic data types and predicate parameters; (3) message alignments should be checked, instantiated data types are validated against the 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.

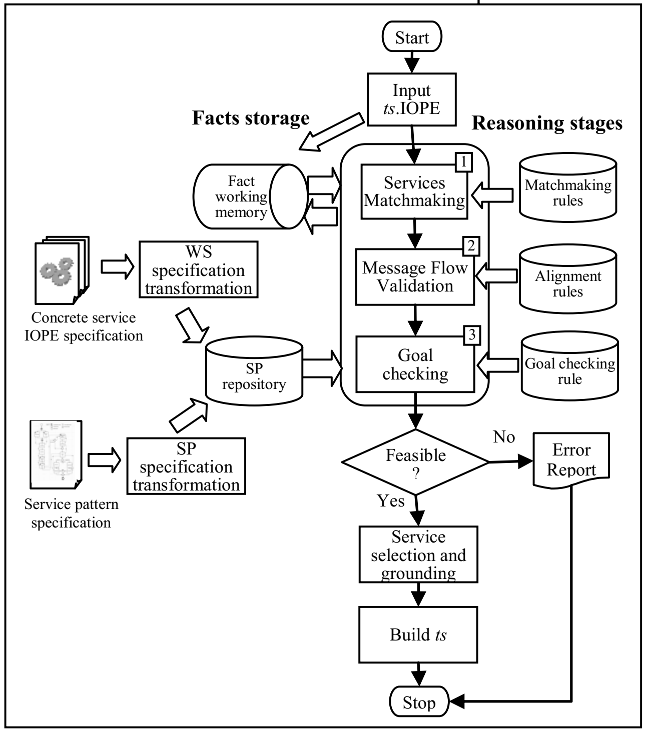

Figure 5. The process of automated instantiation

IV. Automated Instantiation

In addition to the proposed instantiation procedure, we develop an automated instantiation system by which a service pattern *sp* can be automatically instantiated into a target composite service *ts* under the given requirements to significantly reduce the composition efforts. Figure 5 shows the process of our automated instantiation system. Before the instantiation, the specification of available service patterns and IOPE specification of concrete services in the system are converted into 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 fact working memory.

We used Jess v7.1p2 as the rule engine and employ the forward-chaining method in the automated instantiation reasoning process. Because some rules have to consider the related facts as a whole but each fact can only trigger a given rule once in Jess due to performance consideration, we divide the process into three stages to gradually build the needed facts in each stage. The major purpose of the first stage is to concretize IOPE of the prospect services, and the result has to conform to restriction rules of data type parameters and their partial precondition/effect specifications. Secondly, inference rules of validating message flow verify whether data types of the parameters declared in message flow are compatible with each other or not. If it is confirmed, a valid message flow assertion will be put into the fact working memory. Finally, the condition of feasible assignment is checked to verify whether a feasible assignment exists or not.

V. Experimental Evaluation

To evaluate the performance of the automated instantiation, we conducted three experiments to reveal the execution times under different numbers of candidate Web services, message alignments, and prospect services. Therefore, we design a service pattern including five concrete services, two abstract services, three prospect services, and seventeen message flows among services as baseline case. Then, we reduce the numbers of prospect services and message flows as comparative cases, which have three prospect services and nine message flows as well as one prospect service and seventeen message flows.

We observe that more message flows or prospect services may consume more time. Figure 7 shows that the inference stages related with message alignment, namely MF validation (S2) and consistency checking (S3), have higher performance gap than other stages. Especially, at consistency checking stage, the second scenario has significantly higher performance improvement because 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.

On the other hand, the third scenario has fewer prospect services, it produces fewer service matchmaking facts and valid message flow facts. As a result, the reasoning load of service matchmaking (S1) and message flow validation (i.e. S2 and S3) is reduced. Figure 7 provides the evidence for this result. Overall, the execution times of the third scenario at service matchmaking and MF validation are the lowest.

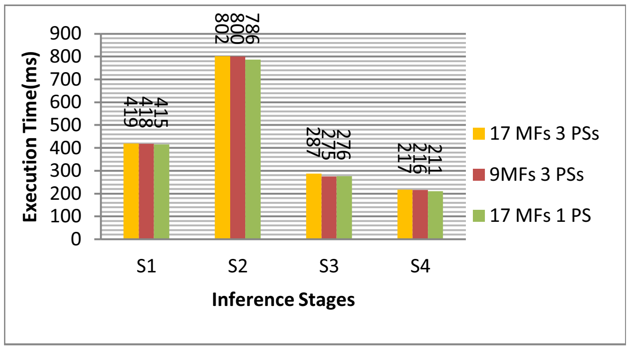

Figure 7. The performance comparisons between three scenarios at different inference stages, S1 for Service Matchmaking, S2 for MF Validation, S3 for Consistency Checking, and S4 for Goal Checking

VI. Conclusion

We have proposed a meta-model by which a flexible and adaptable workflow can be represented to instantiate a large number of service compositions with different functionalities. As a result, by our proposed model, the reusability of existing services can be improved and the efficiency of composing services can also be increased. These benefits are made possible by the unique features of the proposed model: the prospect services with nondeterministic IOPE, adapting controls, and message alignments. We subsequently proposed an automated instantiation parameter assignment method using a rule-based reasoning approach to deduce a feasible assignment of instantiation parameters and reduce the composition effort.

References

[1] M. Abu-Matar and H. Gomaa, "Feature Based Variability for Service Oriented Architectures," in *2011 9th Working IEEE/IFIP Conference on Software Architecture*, 2011, pp. 302-309.

[2] K. Chen, J. Xu, and S. Reiff-Marganiec, "Markov-HTN Planning Approach to Enhance Flexibility of Automatic Web Service Composition," in *IEEE International Conference on Web Services (ICWS 2009)*, 2009, pp. 9-16.

[3] K. Geebelen, S. Michiels, and W. Joosen, "Dynamic Reconfiguration Using Template Based Web Service Composition," in *3rd Workshop on Middleware for Service Oriented Computing*, 2008, pp. 49-54.

[4] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman, "Wings: Intelligent Workflow-Based Design of Computational Experiments," *IEEE Intelligent Systems,* vol. 26, pp. 62-72, 2011.

[5] C. H. Lee and S. Y. Hwang, "A Model for Web Services Data in Support of Web Service Composition and Optimization," in *2009 World Conference on Services - I*, 2009, pp. 384-391.

[6] C. H. Lee, S. Y. Hwang, and I. L. Yen, "A Service Pattern Model for Flexible Service Composition," in *2012 IEEE 19th International Conference on Web Services (ICWS2012)*, 2012, pp. 626-627.

[7] D. McDermott, "Estimated-regression Planning for Interactions with Web Services," in *6th International Conference on Artificial Intelligence Planning Systems*, 2002, pp. 204–211.

[8] T. Nguyen and A. Colman, "A Feature-Oriented Approach for Web Service Customization," in *2010 IEEE International Conference on Web Services (ICWS 2010)*, 2010, pp. 393-400.

[9] M. Pistore, P. Traverso, and P. Bertoli, "Automated Composition of Web Services by Planning in Asynchronous Domains," in *the 15 International Conference on Automated Planning and Scheduling*, 2005, pp. 2-11.

[10] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN Planning for Web Service Composition Using SHOP2," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 1, pp. 377-396, 2004.

[11] L. Zeng, A. Ngu, B. Benatallah, R. Podorozhny, and H. Lei, "Dynamic Composition and Optimization of Web Services," *Distributed and Parallel Databases,* vol. 24, pp. 45-72, 2008.