A Generative Framework for Service Process Composition*

Rajesh Thiagarajan, Wolfgang Mayer, and Markus Stumptner

Advanced Computing Research Centre, University of South Australia
{cisrkt,mayer,mst}@cs.unisa.edu.au

Abstract. In our prior work we showed the benefits of formulating service composition as a Generative Constraint Satisfaction Problem (GCSP), where available services and composition problems are modeled in a generic manner and are instantiated on-the-fly during the solving process, in dynamic composition scenarios. In this paper, we (1) outline the salient features of our framework, (2) present extensions to our framework in the form of process-level invariants, and (3) evaluate the effectiveness of our framework in difficult scenarios, where a number of similar and potentially unsuitable services have to be explored during composition.

1 Introduction

A vast number of proposals that exploit formal specifications of individual services to automatically select and compose individual services into executable service processes have been brought forward [1,2,3]. While most frameworks can successfully address basic composition tasks, many are based on ad-hoc algorithms or lack a precise representation of a service's capabilities. Therefore, the problem of configuring and tailoring the software that implements a given service is often left aside.

Constraint satisfaction based configuration techniques have been proposed as an alternative to address these challenges, where both type and instance information of services and relevant data must be considered [4,5]. Existing models compose services by treating services as components and assembling them. However, standard Constraint Satisfaction Problems (CSPs) are insufficient to model configuration problems where the number of components to be configured is unknown. Existing CSP-based composition techniques handle this by pre-specifying the number or type of services to be composed. In general, such estimation is difficult to make since such problem specific knowledge is not available a priori.

We present a generative consistency-based service composition approach that addresses these challenges [6]. We extend models that have been successfully applied to model and configure complex systems [7] to the software domain. The service composition problem is posed as a configuration task, where a set of service components and their interconnections are sought in order to satisfy a given

* This work was partially supported by the Australian Research Council (ARC) under grant DP0988961.

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 358–363, 2009.

goal. Our framework is based on a declarative constraint language to express user requirements, process constraints, and service profiles on a conceptual and also on the instance level. One of the major benefits of our approach is the provision to define problems independent of the number and type of services required, thereby overcoming the disadvantage of other models.

In this paper, we briefly outline our formalism that offers uniform (generic) constructs to represent service capabilities and semantics, represent data- and control flow between services (Section 2). We also present our extensions to the original model to unambiguously correlate conversations involving service instances or different parts of a workflow in a composition (Section 3). We also present the results from our evaluation on non-trivial pessimistic scenarios, where service compositions often fail (Section 4). Experimental results indicate that even in such pessimistic settings our framework's performance is quite competitive to other composition systems.

2 GCSP-Based Service Composition

CSPs[1] have successfully been applied in various domains including general planning [8] and service composition [4,5]. Generative CSPs (GCSPs) [7] extend standard CSPs by lifting constraints and variables to a meta-level, where *generic constraints* are the primary modeling element. Generic constraints abstract the actual CSP variables into so-called *meta variables* that are instantiated into ordinary constraints over variables in a classical CSP. Generative configuration can be seen as the problem of incrementally extending a CSP network until all the generic constraints are satisfied. In our formalism, when a component (C) of a particular type is added to the configuration, related variables and constraints (instances of the generic constraints of C) are activated in the configuration process. This dynamic activation of CSP variables makes GCSPs suitable for dynamic situations where a priori prediction of the problem structure is difficult.

A GCSP is characterized by the set of available component types, their attributes and ports (to connect to other components), and a set of generic constraints. In generic constraints, meta-variables act as placeholders for component variables. Generic constraints can be seen as prototypical constraints on a meta-level that are instantiated into *ordinary* constraints over variables in the CSP. For example, assume a generic constraint $X \sqsubseteq BudgetShipper \Rightarrow X.price < 1000$ is given, stating the invariant that the value of attribute *price* for any data object of type *BudgetShipper* must be less than 1000. A generic constraint over a meta-variable is said to be consistent if and only if the constraint is satisfied for all instances over active CSP variables. We extended the generative formalism in as follows [6]:

– We introduced connection components that act as connectors between services. The explicit representation of connectors provides a uniform interface

[1] A CSP consists of a finite set of variables, set of values that each variable can be assigned to (the variable's *domain*), and a set of constraints restricting the valid assignments of values to variables.

contract between services and also serves as a means to model the provider-consumer relationship between services.
- A connection component also holds a representation of the data values that may be passed along the connection.
- To capture their semantics, we treat complex data objects as components. This facilitates the uniform handling of service and data components in the configuration, and has the additional benefit that generic constraints can be used to impose invariants on data structures throughout a configuration.
- We introduced non-local process level constraints to model data flow, control flow, and structural invariants of service processes.

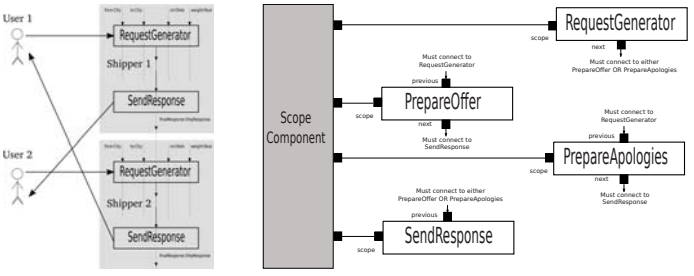
In our approach, a service composition problem is posed a as a configuration task expressed by an initial set of components and constraints that must be satisfied. During the configuration process, the CSP is dynamically extended by adding new variables and constraints. After each extension, the configuration is represented by a standard CSP constraint network (without generic constraints); therefore, standard algorithms can be applied to solve the CSP. An iterative deepening strategy that limits the number of components that may be introduced in the configuration prohibits the configuration from expanding indefinitely. Once that limit has been reached, the algorithm backtracks and attempts to find an alternative solution. If that fails, the limit is increased and the search is restarted. Detailed elaboration on the incremental configuration algorithm is given in [6].

3 Workflow Scope

The scope of a workflow or a sub-workflow defines the tasks it encapsulates in its behavioral process specification. For example, the Shipping process from the *Producer-Shipper* composition problem [9] encapsulates tasks *RequestGenerator* and *SendResponse* to process a request and acknowledge it, respectively. Existing specifications are insufficient if multiple instances of a sub-workflow exist within a composition. For example consider the process model in Figure 1a, where two users interact with two instances of the Shipping workflow.[2] The users would like to place a shipping order using the available shipping process, but are oblivious of each other. From a composition point-of-view, components in each sub-process are interchangeable. For example, the offer requested by *User 1* to *Shipper 1* may actually be sent to *User 2* (as in Figure 1a). Hence, means to ensure messages are directed to correct recipient must be provided.

To address this problem, we introduce explicit workflow *scope components* in our generic framework. Scope components, in addition to encapsulating a process specification, also differentiate between multiple instances of the same workflow by maintaining a *session ID* that is unique for each occurrence of the workflow. Figure 1b shows the scope component that encapsulates the Shipping process in our example. Formally, scope components and their connections to their process elements are also defined using generic constraints like other components

[2] We consider user interaction as an explicit part of the composition.



(a) Multiple Instances Problem

(b) Workflow Scope

Fig. 1. Process Specifications

in our framework. Connections between scope component and the process elements are also defined as generic constraints. While each workflow instance is identified by a unique ID attribute. Additionally, each constituent component in a composition is further distinguished by its unique component ID. Generic constraints make use of the scope components and IDs to ensure that the resulting service composition is well-formed by restricting the control and data flow between components.

4 Experimental Evaluation

We conducted an evaluation on a generalized version of the well-known *Producer-Shipper* problem [9], where a product ordering and shipping process must be composed from individual producers and shipper services, considering the capabilities and restrictions of individual services. Our largest problem with 28 parallel producer-shipper processes (1400 services) can be solved in roughly 3 minutes; a result quite competitive with other approaches [6]. The Producer-Shipper process is quite atypical in that is does not include complex chains of services or non-trivial transformations of data exchanged between services. We conducted further experiments to assess the performance of our framework in a scenario where items must be processed by a sequence of services in order to meet the goal requirements.

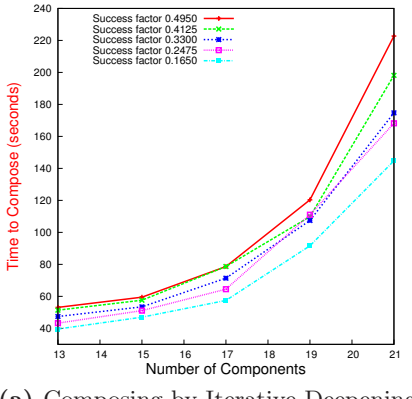
Assume the supply chain of a pie factory (at a fixed location) is to be configured using services. The factory requires supplies of flour and sugar to produce pies. Our model includes wheat and sugarcane farming *services* located in various Indian cities. In addition to the factory and farmers, the problem domain includes flour and sugar mills located in a number of South-East Asian destinations. The model also includes shipping services that ship products between locations. The aim is to compute a composition utilizing available services (farmers, mills, and shippers) in order to facilitate production operations in the pie factory while minimizing handling of the products (minimal number of shippers).

The problem structure in our scenario requires the solver to explore different alternative matching services, and for each alternative, longer chains of prerequisite services must be devised. This extensive exploration at each choice point better reflects the web service composition context, where a large number of services may appear suitable initially, but many fail to satisfy a given request. The following experiment aims to quantify effects of the changed search behavior on our framework.

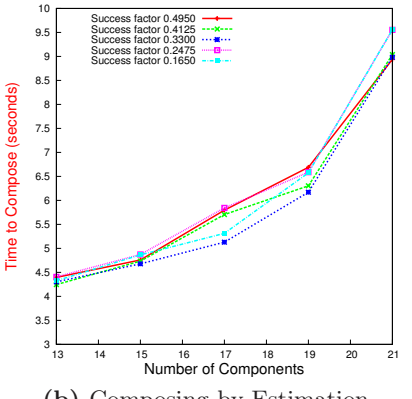
We conducted experiments on a number of variants of our model. All our models contain 1 factory (the goal), 2 mills, and 12 farming locations. The smallest model involves choosing among 25 shipping locations. We gradually increase the number of locations up to 45 locations. Our smallest variant results in a composition with 13 services, and our largest requires 21 services.

We introduce another parameter *success factor* to vary the complexity of the problem. The higher the success factor is the more likely exploring an arbitrary branch will result in a successful match. A model with success factor 1 implies that exploring any out-going branch of the factory will definitely lead us to a matching service. The Producer-Shipper scenario model has a success factor close to 1 and hence exhibits near linear performance [6]. In this experiment we only consider models with success factors below 0.5 to simulate difficult composition problems. This setting reflects the assumption that many candidates that provide similar services exist, but only a few of those will be suitable. We vary the success factors of our models by replacing suitable services with ones that share similar profiles, but do not offer the required capabilities.

The first set of experiments employed the iterative deepening strategy discussed in [6]. The limit on the number of services in each experiment was initialized to 1 and was incremented by 1 in each iterative deepening step. The results are shown in Figure 2a. Our smallest problem with 13 services can be solved in roughly 45 seconds, and our largest problem with 21 services requires roughly 3



(a) Composing by Iterative Deepening



(b) Composing by Estimation

Fig. 2. Supply Chain Problem Results

minutes. This result is comparable in terms of problem size and complexity, and is quite competitive to other composition systems [5,9].

5 Discussion

We have shown that our approach exhibits competitive performance on complex process-level composition scenarios. However, the results in Figure 2a seem counter-intuitive as the problem instances with smallest success factor are solved fastest. Intuitively one would expect the opposite, since the chance of making a wrong choice and backtracking is more frequent in models with low success factor. To analyze this observation, we conducted another set of experiments where the number of service required in a composition is approximated beforehand (no iterative deepening). The results are presented in Figure 2b. It can be observed that the time taken to solve models with high success factor is lesser than that of model with low success factor. Therefore, backtracking caused by the iterative deepening strategy is the cause for the counter-intuitive observation in Figure 2a. We are currently exploring strategies to preserve solutions to sub-problems in the configuration while backtracking. In particular, we are investigating back jumping techniques that identify candidate variables to change based on previous successful and failing attempts to solve sub-problems.

References

1. Liu, Z., Ranganathan, A., Riabov, A.: A planning-based approach for the automated configuration of the enterprise service bus. In: Proc. ICSOC (2008)
2. Lécué, F., Delteil, A., Léger, A.: Optimizing Causal Link Based Web Service Composition. In: Proc. ECAI (2008)
3. Born, M., et al.: Semantic Annotation and Composition of Business Processes with Maestro. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 772–776. Springer, Heidelberg (2008)
4. Karakoc, E., Senkul, P.: Composing semantic web services under constraints. Expert Syst. Appl. 36(8), 11021–11029 (2009)
5. Albert, P., et al.: Configuration Based Workflow Composition. In: Proc. ICWS (2005)
6. Mayer, W., Thiagarajan, R., Stumptner, M.: Service Composition As Generative Constraint Satisfaction. In: Proc. ICWS (2009)
7. Fleischanderl, G., et al.: Configuring large-scale systems with generative constraint satisfaction. IEEE Intelligent Systems 13(4) (1998)
8. Pralet, C., Verfaillie, G.: Using constraint networks on timelines to model and solve planning and scheduling problems. In: Proc. ICAPS (2008)
9. Pistore, M., et al.: Automated composition of web services by planning at the knowledge level. In: Proc. IJCAI (2005)