Generative Composition of Web Services*

Rajesh Thiagarajan, Wolfgang Mayer, and Markus Stumptner

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

Abstract. We present a consistency-based service composition approach, where composition problems are modelled in a generic manner using a generative constraint-based formalism. We show that in our framework concise formal specification of service configuration problems is possible. Preliminary results indicate that our approach is scalable and competitive with other state-of-the-art composition systems.

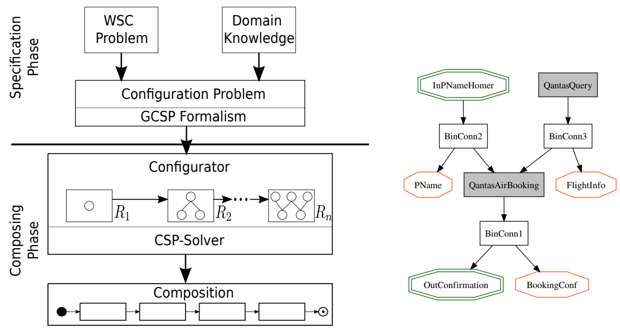
1 Introduction

Although service-oriented architectures facilitate dynamic adaptation and reuse of software, manual selection and composition of services still require considerable effort for complex scenarios. Several composition approaches [1–7] have been proposed, but many are based on ad-hoc algorithms, lack a precise representation of a service's capabilities or require prediction of the number of required services. Dynamic composition scenarios, however, require a formalism that can flexibly choose the structure of a composition and that incorporates a variety of search heuristics to compute solutions efficiently based on conceptual information as well as attributes of concrete service instances.

Constraint-based systems have been shown to provide expressive formalisms and efficient inference procedures to efficiently assemble large-scale component-oriented systems [8]. Here, *Generative Constraint Satisfaction Problems* (GCSPs) overcome the limitations of classic CSPs with fixed structure by incrementally introducing additional variables and constraints when components are added to satisfy a constraint. The fundamental building block of the generative model are the so-called *generic constraints* that specify constraints between variables on a meta level. To adapt GCSPs for service configuration, it is necessary to extend the formalism to capture aspects such as data flow and complex data structures:

- We introduce connection components that act as connectors between services. The explicit representation of connectors provides a uniform interface contract between services and also serves as a means to explicitly model the provider-consumer relationship between services. A connection component also holds a representation of the data values that is passed along the connection.
- We introduce explicit components to capture the semantics of complex data objects. This facilitates the uniform handling of service and data components

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



(a) Composition Process (b) Partial Solution

Fig. 1: Generative Composition of Web Services

in the configuration, and has the additional benefit that generic constraints can be used to impose global invariants on data structures.

In our extended framework, 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 goal. The compositions computed in our framework can be synthesised into a form suitable to execute the composed service such as BPEL and OWL-S. We show that concise formal specification of service configuration problems is possible. In our framework, GCSPs are synthesised into classic CSPs allowing efficient CSP algorithms to potentially help tune implementations of reasoning tasks. Our formalism is particularly useful in scenarios where the problem size is unknown. Preliminary evaluation shows that GCSPs can quickly find solutions for large non-trivial problems.

2 Service Composition by Configuration

Configuration is the process of assembling larger systems from individual components by extending an initial partial solution representing a goal until all the requirements are satisfied. A configuration problem is defined over a so-called *configuration domain* [9] which represents the types of components to be configured, their properties and a set of generic constraints that have to be satisfied.

Definition 1 (Configuration Domain) *A* configuration domain *is a tuple* $\langle \mathcal{T}, \sqsubseteq, \mathcal{C}, \mathcal{P}, \mathcal{D}, \Gamma \rangle$, *where*

- $\mathcal{T}$ *is a finite set of service types and data object types.*
- *The partial order* $\sqsubseteq$ *denotes the subtype relationship between the types in* $\mathcal{T}$. *We write* $\tau' \sqsubseteq \tau$ *to imply that* τ' *is a subtype of type* τ; τ' *inherits all properties and constraints of* τ.
- $\mathcal{C}$ *is an infinite set of component variables representing instances of services and data objects.*

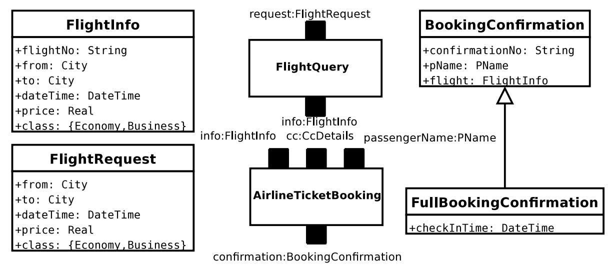


Fig. 2: Service Composition by Configuration

- $\mathcal{P}$ *is the set of property variables. We use the notation* $c.p$ *to denote the CSP variable representing property* p *of the service instance represented by the component variable* c*.*
- $\mathcal{D}$ *is the set of domains of the component and property variables. This set includes the sets* $\mathcal{C}$*, numbers, strings, and may include other domain-specific sets if required.*
- Γ *is a finite set of generic constraints including activation and resource constraints.*

Variables in $\mathcal{C}$ are initially considered *inactive*, but may be *activated* by a generic constraint. This can be seen as "expansion" operation, where the CSP is extended with additional variables (and possibly constraints). Only the active variables participate in the solution process, while the inactive variables remain hidden until activated. Hence, a valid configuration is an assignment of values to all *active* variables in $\mathcal{C}$ such that all constraints are satisfied.

We use the following scenario to illustrate our approach, where flights between two specific cities on a given date are to be purchased. A partial model of service types in our travel domain is shown in Figure 2, including the available service and data types, service interfaces and relationships (modelled as inheritance), and available service instances. In our example $\mathcal{T}$ includes, among others, the service types *FlightQuery* and *AirlineTicketBooking*, component type *BinConn* representing connections, and data type *BookingConfirmation*.

Generic constraints are the primary constructs for expressing the semantics of components and data types in the GCSP formalism. 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.

A generic constraint is a clause of the form $A_1 \wedge \ldots \wedge A_m \Rightarrow B_1 \vee \ldots \vee B_n$ where the A's and B's are predicates over meta-variables. A generic constraint is consistent if and only if the constraint is satisfied for all bindings of meta-variables to active CSP variables.

Activation constraints are a form of generic constraints that are used to activate additional CSP variables once new components are introduced into a configuration. Conceptually, activation constraints govern the "growing" of the constraint network. Activation constraints are of the form $C \sqsubseteq \tau \implies C.p \hat{\in} \tau'$

and denote that each component of type τ (or a sub type) has a property p of type τ'. The properties that have $\mathcal{C}$ as their domain are referred to as ports, while the remaining properties of primitive data type are called attributes. For example, the activation constraint $X \sqsubseteq \mathit{FlightQuery} \implies X.\mathit{request} \hat{\in} \mathcal{C}, X.\mathit{info} \hat{\in} \mathcal{C}$. ensures that for each constraint variable representing a service component of type *FlightQuery* (see Figure 2), the port variables $X.\mathit{request}$ and $X.\mathit{info}$ are activated with domain $\mathcal{C}$ allowing them to be connected with other components.
Data Flow is captured by distinguished *BinConn* components that act as connectors between service components in the configuration. A *BinConn* component acts as an interface contract between service components by providing a uniform interface for the components connected to each port. Here, varying matching strategies, such as the well-known subsumption match [7], can be employed. Directionality is modelled by constraints that enforce a connection to connect a data consumer to a data provider where the supplied object is compatible with the requested data type.
Service Semantics are captured by specifying attributes, IO parameters, their types, and relationships between IO parameters in form of generic constraints. For example, "A *FlightRequest* object is expected through the *request* port" is modelled as $S \sqsubseteq \mathit{FlightQuery} \ \wedge S.\mathit{request}{=}BC {\Longrightarrow} BC \sqsubseteq \mathit{BinaryConn} \ \wedge$ $BC.\mathit{tout} \sqsubseteq \mathit{FlightRequest} \ \wedge BC.\mathit{out}{=}S$.
Structural Constraints ensure service compositions are sound. In particular, constraints to prohibit the direct connection of two connection components and cyclic data flow paths are required. We define a component u to *depend* on outputs of v, denoted as $u \rightsquigarrow v$, if a path from an input port of u to a port of v exists in the CSP graph. The integrity constraint used to ensure that all CSP instances are acyclic can then be expressed as the generic constraint $C \not\rightsquigarrow C$.
Resource constraints are used to express aggregate operations over attributes of multiple components and to impose limits on the number of components in a configuration. For example, the cost constraint in our example is modelled as $\Sigma\{S.\mathit{flight.price} | S \sqsubseteq \mathit{BookingConfirmation}\} \leq 400$. Resource constraints may also trigger the addition of new components in the configuration.
Goal Requirements and User Inputs are modelled using distinguished components: available inputs are modelled as components that supply one available user input; these components are not instantiated by default, but are created by the GCSP solver on demand. Required outputs are also modelled as components; from these components, a partial configuration that represents the goal is created, which must subsequently be completed by the GCSP solver.

3 Computing Service Compositions

A **Configuration problem** for a given configuration domain $D = \langle \mathcal{T}, \sqsubseteq, \mathcal{C}, \mathcal{P}, \mathcal{D}, \Gamma \rangle$ is defined as a tuple $\langle V^I, \Gamma^I, D \rangle$, where $V^I \subseteq \mathcal{C} \cup \mathcal{P}$ is the set of initial variables, where $\mathcal{P}$ is the set of property variables activated by components in $\mathcal{C}$, and Γ^I is a set of constraint instances on V^I.

The solution process is illustrated in Figure 1a. Initially, the constraint network R consists of a partial composition $\langle V^I, \Gamma^I \rangle$ that represents the goal requirements. During the composition process, R is dynamically extended by adding new variables and constraints. After each extension, R represents a standard CSP (without generic constraints); therefore, standard algorithms can be applied to solve the CSP. The CSP R will grow

1. if a type τ is assigned to a component variable c, fresh variables representing the ports and properties of τ are introduced to satisfy activation constraints,
2. if a port variable has been chosen for assignment and no existing component is eligible, a new component is added to R, or
3. if a resource constraint cannot be satisfied using the current set of components.

The constraint network R continues to grow until all generic constraints are satisfied. Once all variables have been assigned the solving process terminates and the variable assignments are returned; the components assigned to the variables in the completed network represents a valid configuration. Otherwise, a variable is chosen for assignment and constraints are propagated. If no alternative value assignment remains for a variable, the (G)CSP is inconsistent and a different choice for a preceding variable assignment is explored.

To ensure termination of our composition process, we employ an iterative strategy that limits the number of components, where the threshold is iteratively increased until a solution is found.Solving iteratively allows us to limit the effects of wrong component choice that would otherwise have resulted in indefinite expansion of the CSP.

A partial solution to the example problem is shown in Figure 1b, where the configuration grows beginning with the goal specification (*OutConfirmation*). Since the example only involves a few components, it can be solved almost instantaneously in our framework. To characterise the performance of our framework, we conducted experiments on a generalised version well-known *Producer-Shipper* problem [1]. In the adapted scenario, multiple instances of producer and shipper process exists with restrictions on their combination. For example, pa product can only be shipped by certain vendors. 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.

4 Discussion and Future Work

We presented a consistency-based service composition approach in which a service composition problem is treated as a configuration task. We showed that composition problems can be concisely represented by adopting a declarative, constraint-based meta model. By translating the service composition problem as a dynamic configuration task, we overcome the limitations of earlier work where the structure of a composition must be known a priori [10, 11]. Such a fixed scenario can be implemented by imposing constraints on the number of services and their interconnections, while cost-based optimisation and preferences [10] can be integrated through variable and value selection heuristics.

Since the GCSPs formalism can simultaneously handle constraints on different levels of abstraction, conceptual and instance specific information can be exploited for problem specification and also in the solving process. Specific information derived from service instances has already been applied to synthesise concrete executable work flows in domains where conceptual information alone is insufficient to generate precise compositions [12]. Our framework adopts similar ideas, but provides a uniform formalism in which flexible CSP solving strategies based on the concrete and conceptual information present in a partial solution can be exploited. The GCSP framework also permits hierarchical refinement of abstract solutions, where CSP solving strategies are selected dynamically driven by the specific information and structure of a partial solution rather than adopting a fixed solving strategy throughout the entire solving process, as in [3, 13, 14].

Preliminary evaluation shows that our framework is scalable to non-trivial problems and can solve compositions with hundreds of service instances efficiently. We are currently extending our framework to exploit workflow patterns to ensure interactions with complex flow, for example, synchronisation and exception handling, can be synthesised effectively. We also plan to leverage earlier work on consistency-based matchmaking into our framework to improve service selection in scenarios where only partial information is available.

References

1. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: Proc. IJCAI. (2005)
2. Trainotti, M., et al.: ASTRO: Supporting Composition and Execution of Web Services. In: Proc. ICSOC. (2005)
3. Sirin, E., Parsia, B., Wu, D., Hendler, J.A., Nau, D.S.: HTN planning for Web Service composition using SHOP2. Journal of Web Semantics **1**(4) (2004) 377–396
4. Lécué, F., Léger, A.: Semantic Web Service Composition Based on a Closed World Assumption. In: Proc. ECOWS. (2006)
5. McIlraith, S.A., Son, T.C.: Adapting Golog for composition of Semantic Web services. In: Proc. KR. (2002)
6. Gómez-Pérez, A., González-Cabero, R., Lama, M.: ODE SWS: A Framework for Designing and Composing Semantic Web Services. IEEE Intelligent Systems **19**(4) (2004) 24–31
7. Li, L., Horrocks, I.: A software framework for matchmaking based on Semantic Web technology. In: Proc. of WWW, Budapest (2003) 331–339
8. Fleischanderl, G., et al.: Configuring large-scale systems with generative constraint satisfaction. IEEE Intelligent Systems **13**(4) (1998)
9. Haselböck, A., Stumptner, M.: An integrated approach for modelling complex configuration domains. In: Proc. of the 13th Int. Conf. on Expert Systems, AI, and Natural Language. (1993)
10. Hassine, A.B., Matsubara, S., Ishida, T.: A Constraint-Based Approach to Horizontal Web Service Composition. In: Proc. ISWC. (2006)
11. Thiagarajan, R., Stumptner, M.: Service Composition With Consistency-based Matchmaking: A CSP-based Approach. In: Proc. ECOWS. (2007)
12. Carman, M., Serafini, L.: Planning For Web Services the Hard Way. In: SAINT Workshops. (2003)
13. Albert, P., Henocque, L., Kleiner, M.: Configuration Based Workflow Composition. In: Proc. of ICWS 2005. (2005)
14. Carman, M., Serafini, L., Traverso, P.: Web Service Composition as Planning. In: ICAPS Workshop on Planning for Web Services, Trento, Italy (June 2003)