Considerations of Service Assembly based on the Analysis of Data Flows between Services

George Feuerlicht
Faculty of Information Technology,
University of Technology, Sydney,
P.O. Box 123 Broadway, Sydney, NSW 2007, Australia
jiri@it.uts.edu.au

Abstract

Service composition research mostly focuses on the dynamic (workflow) aspects of compositions. In this paper we consider the static component of service composition and focus on analyzing the data flows between services within a composition. We argue that compatibility of service interfaces is a necessary precondition for service composability, and we show that data flow analysis can be applied to the problem of service composition design to identify compatible service interfaces and to minimize data coupling between services.

1. Introduction

In general, the design of service compositions consists of two parts: a static part that involves the definition of services in WSDL, including the service operations and their interfaces, and a dynamic part that defines the associated process workflow using languages such as BPEL (WS-BPEL). It can be advantageous to treat the design of the static part of service compositions separately, to ensure that the services are composable and can be reused in the context of various process specifications [1]. BPEL compositions involve implementing higher level business functions using previously defined Web Services accessible via partner links and externalizing the resulting functionality via WSDL interface, i.e. as a composite Web Service. The BPEL model is a message-based paradigm and communication between Web Services involves mapping the results of service invocations between the outbound and inbound messages of service interfaces (i.e. signatures of Web Service operations). We adopt the BPEL composition model in this paper and discuss composition in the context of the data flows between services (i.e. the outbound and inbound messages passed between service operations). To facilitate composition of services into higher level business functions, service interfaces must be compatible, i.e. share common data parameters. At the same time the granularity of service operations must be optimized with respect to composability and reuse. In previous publications we have described a methodological framework for the design of services that uses top-down decomposition based on the data properties of interface parameters to maximize cohesion and minimize coupling of service operations [2], [3]. In this paper we focus on the problem of service assembly via the analysis of data flows between services. We regard service composition as a design-time activity with the objective to ensure that service interfaces are compatible (and therefore composable), and at the same time exhibit a high degree of mutual independence. In the next section (sections 2) we describe a travel booking scenario and use it to analyze the data flows between services, we then summarize the main contributions of this paper in section 3.

2. Analysis of data flows between services

Service-Oriented Architecture (SOA) is characterized by loosely coupled, coarse-grained services that typically encapsulate high-level business processes (e.g. air travel booking, course enrolment, etc.) and rely on the exchange of composite XML documents to accomplish business transactions. This mode of operation is widely adopted by the SOA practitioners for developing Web Services applications. For example, travel Web Services based on the Open Travel Alliance

(http://www.opentravel.org/) specification implement flight booking business process for a specific itinerary using two coarse-grained, composite XML request/response message pairs: OTA_AirAvailRQ/OTA_AirAvailRQ and OTA_AirBookRQ/OTA_AirBookRS. We use a simplified flight booking scenario loosely based on the OTA specification to illustrate our discussion of service composition. We make a number of simplifying assumptions including that the flights are one-way with no stopovers and that flights for a given FlightNumber depart every day of the week at the same time (DepartureTime). Unlike the coarse-grained message interchange pattern used by the OTA specification we break down the flight booking business function into four low granularity operations

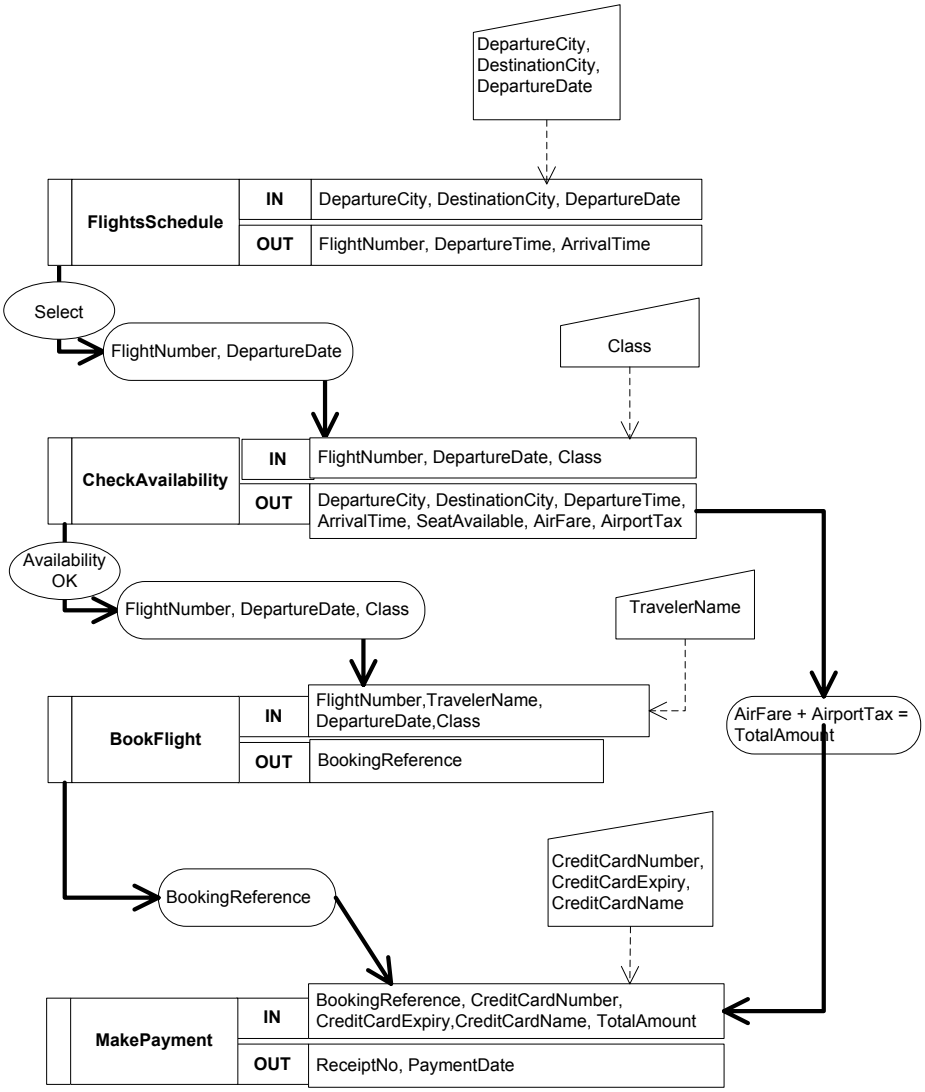

Figure 1. Flight booking operations and corresponding data flows.

FlightsSchedule, CheckAvailability, BookFlights, and MakePayment that closely match the requirements of the flight booking dialogue. The flight booking dialogue proceeds as follows: The traveler supplies the values for DepartureCity, DestinationCity, and DepartureDate as input parameters for the FlightsSchedule operation. The output of the FlightsSchedule operation produces a list of scheduled flights, i.e. corresponding values of FlightNumber, DepartureTime, and ArrivalTime. The traveler then selects a suitable flight (i.e. FlightNumber and DepartureDate) supplies the value of Class (e.g.

economy); the values of FlightNumber, DepartureDate and Class then form the input for the CheckAvailability operation. The output of the CheckAvailability operation includes information about flight availability (SeatAvailable) and pricing information (Airfare and AirportTax). Assuming that seats are available for the selected flight the traveler proceeds to book the flight using the BookFlight operation that takes the values of FlightNumber, DepartureDate, Class, and TravelerName as the input, and produces BookingReference as the output. Finally, the traveler makes a payment using the MakePayment operation supplying, credit card information (CreditCardNumber, CreditCardExpiry, CreditCardName). The MakePayment operation accepts the input parameters BookingReference and TotalAmount (sum of Airfare and AirportTax) generated by the BookFlight and SelectFlight operations, respectively, and produces ReceiptNo and PaymentDate as the output parameters.

As the payment operation (MakePayment) is separate from the booking operation (BookFlight) it is possible to hold the booking without a payment, if this is permitted by the airline. Furthermore, the MakePayment operation can be reused in a different context, e.g. in a hotel booking service.

Although the data used by the flight booking scenario is typically stored in different databases belonging to different participants in the business process (i.e. travel agent, airline, etc.), for the purposes of this analysis we assume that this data can be represented by a global database schema. We note here that we do not make any assumptions about how and where the data is stored; we simply use the underlying data structures to reason about the composability of services.

We also do not consider issues related to state maintenance, as these are orthogonal to the considerations of service composability. OTA specification also assumes that the data transmitted in XML messages (e.g. as payloads in Web Services SOAP messages) is stored persistently in the target databases and provides a number of messages to synchronize the data across the various participants (e.g. OTA_UpdateRQ/RS, OTA_DeleteRQ, etc.).

We can now proceed to analyze the underlying data structures as represented by the data elements in the interfaces of the service operations. Data analysis of the content of the interfaces of service operations FlightsSchedule, CheckAvailability, BookFlights, and MakePayment produces a set of 5 normalized relations that constitute the underlying database schema of the flight booking business function:

Flights(FlightNumber,
DepartureCity,DestinationCity,DepartureTime,ArrivalTime)

Schedule(FlightNumber,DepartureDate, AircraftType)

Availability(FlightNumber,DepartureDate,Class,
SeatAvailable,AirFare,AirportTax)

Bookings(BookingReference,
TravelerName,FlightNumber,DepartureDate,Class,Seat)

Payments(ReceiptNo,
PaymentDate,CreditCardNumber,CreditCardExpiry,
CreditCardName,BookingReference)

Based on this schema we implement the flight booking service composition as illustrated in Figure 1, showing the data flows between services (full lines) and the user inputs (dashed lines). Passing data values between service operations generates data flows that indicate the level of data coupling between services. It can be argued that both reuse and composability of services improves when the level of data coupling is minimized [2], so that service design should aim to produce compatible interfaces that share common data parameters, but at the same time minimize the level of data coupling. We can observe that the data flows between the service operations correspond to the primary keys of the relations that constitute the underlying global schema. For example, the composite key of the Availability relation (FlightNumber, DepartureDate, Class) forms the data flow between CheckAvailability and BookFlight operations, and that the BookingReference (i.e. the primary key of the Bookings relation) constitutes the data flow between BookFlight and MakePayment operations. This indicates that data coupling between the service operations is minimized as the elimination of any parameter would invalidate the composition, e.g. removing Class from the data flow between CheckAvailability and BookFlight operations would break the functionality of the flight booking business function. We can conclude that data flows between service operations that correspond to key attributes of the underlying schema, as is the case in Figure 1, satisfy the condition of minimal data coupling and at the same time indicate that the composition is based on compatible service interfaces.

3. Related Work and Conclusions

Service composition can be regarded as a special category of the software composition problem. In addition to extensive recent literature on service composition, the problem of composition has been investigated in the past in the context of object-oriented software [5], [6], [7], and in the general area of software composition [8]. Other researchers have applied formal methods and developed specialized composition languages to address the problem of composition [9], [10], [11], [12].

We have argued in this paper that from the viewpoint of service reuse and composability, the static part of service assembly design that involves the definition of the service interfaces (i.e. service operations and their interfaces) is of primary importance.

The main contribution of this paper is to show how data flow analysis can be applied to the problem of service composition design. We have shown that in a *well-designed* composition the data flows between service operations correspond to the keys of the relations in the underlying global schema.

5. References

[1] Thöne S., Depke R and Engels G. Process-Oriented, Flexible Composition of Web Services with UML, LNCS, 0302-9743 (Print) 1611-3349 (Online), Volume 2784/2003

[2] Feuerlicht, G., Design of Service Interfaces for e-Business Applications using Data Normalization Techniques, Journal of Information Systems and e-Business Management, Springer-Verlag GmbH, 26 July 2005, pages 1-14, ISS:1617-98

[3] Feuerlicht, G., Meesathit, S., Design Method for Interoperable Web Services, Proceedings of the 2nd International Conference on Service Oriented Computing, New York City, NY, USA, November 15-18, 2004, pages 299-307, ISBN 1-58113-871-7

[4] Juric, M. Poornachandra S, and Mathew B., Business Process Execution Language for Web Services, 2nd Edition, Packt Publishing, January 2006; ISBN 1904811817

[5] Nierstrasz, O. and Meijler, T. D. 1995. Research directions in software composition. ACM Computer. Survey. 27, 2 (Jun. 1995), 262-264.

[6] Georgakopoulos D. Ritter N. Benatallah B. Zirpins C. Feuerlicht G. Schoenherr M. Hamid R. Motahari-Nezhad (Eds.) Workshops Proceedings, Service-Oriented Computing, ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006, LNCS 4652, ISBN 978-3-540-75491-6, Book Series Lecture Notes in Computer Science, Volume 4652/2007 Springer-Verlag Berlin Heidelberg 2007

[7] Object-Oriented Software Composition, edited by Oscar Nierstrasz and Dennis Tsichritzis, is available in hardcover from Prentice Hall International. Published 1995, ISBN 0-13-220674-9.

[8] Nierstrasz, O. and Meijler, T. D. 1995. Research directions in software composition. ACM Comput. Surv. 27, 2 (Jun. 1995),
262-264. DOI=http://doi.acm.org/10.1145/210376.210389

[9] CoorSet: A Development Environment for Associatively Coordinated Components, Book Series Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ISSN0302-9743 (Print) 1611-3349 (Online), Volume 2949/2004, Coordination Models and Languages, ISBN 978-3-540-21044-3, Pages 216-231, May 13, 2004

[10] Scheben, U. 2005. Hierarchical composition of industrial components. Sci. Comput. Pro-gram. 56, 1-2 (Apr. 2005), 117-139.
DOI= http://dx.doi.org/10.1016/j.scico.2004.11.008

[11] Weerawarana, S., Curbera, F., Duftler, M. J., Epstein, D. A., and Kesselman, J. 2001. Bean markup language: a composition language for JavaBeans components. In Proceedings of the 6th Conference on USENIX Conference on Object-Oriented Technologies and Systems - Volume 6 (San Antonia, Texas, January 29 - February 02, 2001). USENIX Association, Berkeley, CA, 13-13.

[12] Holt, R. C. 1998. Structural Manipulations of Software Architecture Using Tarski Relational Algebra. In Proceedings of the Working Conference on Reverse Engineering (Wcre'98) (October 12 - 14, 1998). WCRE. IEEE Computer Society, Washington, DC, 210.