# Design of Composable Services

George Feuerlicht[1,2]

[1] Department of Information Technology,
University of Economics, Prague, W. Churchill Sq. 4, Prague, Czech Republic
[2] Faculty of Engineering and Information Technology,
University of Technology, Sydney
P.O. Box 123 Broadway, Sydney, NSW 2007, Australia
jiri@it.uts.edu.au

**Abstract.** Service composition methods range from industry standard approaches based on Web Services and BPEL to Semantic Web approaches that rely on AI techniques to automate service discovery and composition. Service composition research mostly focuses on the dynamic (workflow) aspects of compositions. In this paper we consider the static component of service composition and discuss the importance of compatibility of service interfaces in ensuring the composability of services. Using a flight booking scenario example we show that reducing the granularity of services by decomposition into service operations with normalized interfaces produces compatible interfaces that facilitate service assembly. We then show how relational algebra can be used to represent service operations and provide a framework for service assembly.

**Keywords:** Service composition, service assembly, service reuse.

## 1 Introduction

In general, the specification of service compositions consists of two parts: the static part that involves the definition of services, including the service operations and their interfaces, and the dynamic part that defines the associated process workflow. It can be advantageous to treat the design of the static part of service compositions separately, as the service operations can then be reused in the context of various process specifications [1]. We refer to this static part of service composition as service assembly in this paper. In this context, the design of the inbound and outbound message structures is of paramount importance as it determines the compatibility of service interfaces, and consequently the composability of services into higher level business functions. A key determinant of service composability and reuse is service granularity, i.e. the scope of functionality that individual services implement. Increasing the scope of the functionality implemented by a given service reduces the potential for reuse and therefore makes the assembly of services more problematic. In previous publications we have described a methodological framework for the design of services based on the data properties of interface parameters that aims at achieving optimal level of service granularity [2, 3]. In this paper we extend this framework and consider the problem of service assembly. Unlike some authors, for example [4-8] who consider service composition a run-time problem and apply semantic

G. Feuerlicht and W. Lamersdorf (Eds.): ICSOC 2008, LNCS 5472, pp. 15–27, 2009.

techniques to run-time resolution of compatibility conflicts, we regard service assembly a design-time concern and focus on specifying compatible services interfaces. Furthermore, our analysis assumes that we are dealing with domain-wide services based on an industry standard specification, avoiding incompatibilities arising from services designed by individual service providers. Service assembly is a recursive process that produces high-level (coarse-grained) services as compositions of elementary (fine-grained) services. A key objective of service design is to ensure that services are both composable and reusable so that higher level services can be implemented as assemblies of mutually independent elementary services. Service compositions are typically implemented using languages such as BPEL (Business Process Execution Language) to form complete business processes, and we adopt the BPEL composition model as the basis for our analysis [9]. BPEL compositions involve implementing higher level business functions using previously defined Web Services accessible via partner links and externalizing the resulting functionality of the composite service via a WSDL interface. This process can be repeated recursively, so that complex high-level business functions can be implemented by aggregation of lower level services [10]. The BPEL model is a message-based paradigm and the communication between Web Services involves mapping the results of service invocations between the outbound and inbound messages of service interfaces (i.e. the signatures of Web Service operations). Local and global BPEL variables are used to store and manipulate the intermediate results of service invocations. It follows that composability of services is dependent on the compatibility of interfaces of service operations involved in the composition.

We have argued elsewhere that service interfaces can be treated as data parameters, and that data engineering principles apply to the design of services [2]. In this paper we extend this work to considerations of service composability. More specifically, we argue that composability of services depends on compatibility of service interfaces within the service assembly, and show that relational algebra formalism can be used to represent the static part of service compositions. In the next section (sections 2) we explore service composability in more detail and discuss the relationship between service reuse and composability. We then illustrate the process of decomposing coarse-grained services into elementary (fine-grained) services with normalized interfaces that facilitate service composition using a flight booking scenario (section 3). In the next section (section 4) we show how relational algebra formalism can be used to represent service operations and to describe service assembly. In the final section (section 5) we summarize the main contributions of this paper, discuss related work and outline further research.

## 2 Considerations of Service Reuse and Composability

Composability of services is closely related to service reuse. Many experts believe that reuse is inherent to SOA (Service-Oriented Architecture), and studies show that organizations regard reuse as the top driver for SOA adoption [11]. However, in practice reuse can be difficult to achieve and involves design-time effort to identify and design reusable services. Once services are published it becomes very difficult to improve the level of service reuse by runtime intervention, or by modifying the externalized service interfaces. It can be argued that the perception of improved reuse can be mainly attributed to the ability to derive business value from legacy applications by

externalizing existing functionality as Web Services [12]. Others have noted that the relatively low levels of service reuse can be attributed to poor design [13]. For the purposes of this analysis it is important to understand the mechanism for service reuse and how it differs from earlier approaches to software development. The mechanism for service reuse is service aggregation and we can define service reuse as the ability of a service to participate in multiple service assemblies/compositions; in this sense, service composability can be regarded as a measure of service reuse. Service composition implements complex application functionality (for example, a composite service can implement a hotel reservation, airline booking and a car rental to form a complete travel agency service) by means of recursively composing services [14]. In order to facilitate composition, the constituent services need to have characteristics which allow reuse in composing services.

Services share the basic characteristics of components such as modularization, abstraction and information hiding, and design strategies used in earlier software development approaches can be adapted to service design. However, there are significant challenges to overcome as services are typically implemented at a higher level of abstraction than components, and reuse potential is limited by the extensive use of coarse-grained, document-centric services. Another factor that makes achieving good levels of service reuse particularly challenging is that service-oriented applications tend to span organizational boundaries and the design of services frequently involves industry domain considerations. Emerging vertical domain standards such as the Open Travel Alliance (OTA) [15] specification of (XML) message formats for the travel industry domain are typically developed by committees or consortia which tend to operate on a consensus basis and pay little attention to software design. Although industry-wide standards are an essential prerequisite for e-business interoperability, standardization of message structures and business processes alone does not ensure reusability of services. The resulting message structures typically include a large number of optional elements and embedded instructions that control the processing of the documents [16]. This results in excessively complex and redundant data structures (i.e. overlapping message schemas) that include a large number of optional data elements introducing high levels of data and control coupling between service operations [17]. Consequently, Web Services based on such message formats do not exhibit good levels of reuse and composability [18]. It is often argued that standardization leads to reuse [19], but in order to achieve high levels of service reuse and composability, detailed consideration needs to be given to service properties at design-time. More specifically, services should be self-contained, have clearly defined interfaces that are compatible across the domain of interest. These requirements lead to a consideration of cohesion (i.e. maximization of service cohesion) and coupling (i.e. minimization of coupling between services) resulting in fine-grained services that are associated with improved level of service composability [20].

## 3 Identifying Composable Services

Most vertical-domain applications are characterized by coarse-grained services that typically encapsulate high-level business processes 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

to improve performance and reduce the number of messages that need to transmitted to implement a specific business function [21]. Consider, for example, travel Web Services based on the OTA specification implement flight booking business process for a specific itinerary using two request/response message pairs: OTA_AirAvailRQ/ OTA_AirAvailRS and OTA_AirBookRQ/OTA_AirBookRS. The OTA_AirAvailRQ/ OTA_AirAvailRS message pair includes the data elements of requests and responses for airline flight availability and point of sale information [15]. This situation is illustrated in Figure 1. The Availability Request message requests flight availability for a city pair on a specific date for a specific number and type of passengers, and can be narrowed to request availability for a specific airline, flight or booking class on a flight, for a specific date. The Availability Response message contains the corresponding flight availability information for a city pair on a specific date. The Availability request/response interaction is (optionally) followed by the Booking request/response message exchange. The Book Request message is a request to book a specific itinerary for one or more passengers. The message contains origin and destination city, departure date, flight number, passenger information, optional pricing information that allows the booking class availability and pricing to be re-checked as part of the booking process. If the booking is successful, the Book Response message contains the itinerary, passenger and pricing information sent in the original request, along with a booking reference number and ticketing information.

The use of such complex (coarse-grained) data structures as payloads of Web Services SOAP message improves performance, but significantly reduces reuse potential [22].

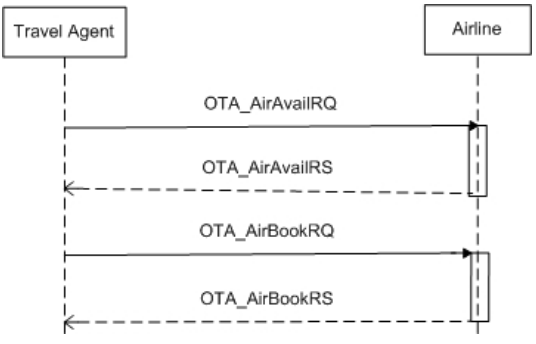



**Fig. 1.** OTA flight availability and booking message sequence

Decomposition of coarse-grained services into fine-grained (elementary) service operations improves the opportunity for reuse and is a necessary precondition for composability of services. In order to facilitate service composition, detailed consideration needs to be given to service interfaces at design-time to ensure that individual services are mutually compatible across a collection of related services. This leads to the requirement for matching of interface parameters, so that for example, the BookingReference (output) parameter of the BookFlight operation matches the BookingReference (input) parameter of the MakePayment operation. We explore this aspect of service composition in the following sections, using an airline flight booking scenario introduced in this section.

### 3.1 Decomposition of the Travel Booking Service

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). These simplifications make the example easier to follow while maintaining good correspondence to the *real-world* situation. Unlike the coarse-grained message interchange pattern used by the OTA specification (illustrated in Figure 1) this scenario breaks down the business function into fine-grained service operations that closely match the requirements of the flight booking dialogue. As argued elsewhere [13, 23], the benefits of this fine-grained design include improved cohesion, reduction in coupling and better clarity. But, also importantly, reducing granularity leads to improved flexibility, reusability and composability of services, so that for example, the payment operation (MakePayment) is now separate from the booking operation (BookFlight). As a result, it is possible to hold the booking without a payment, furthermore, the MakePayment operation can be reused in a different context, e.g. in a hotel booking service. In order to identify candidate service operations, we first model the flight booking dialogue using a sequence diagram (Figure 2), and then define the corresponding service interfaces using simplified OTA data elements. Similar to the OTA message sequence shown in Figure 1, the sequence diagram in Figure 2 describes the interaction between a travel agent and an airline. Each message pair consists of a request (RQ) message and a response (RS) message that together form the interface of the corresponding candidate service operation. We can now describe the flight booking function in more detail using a composition of 4 service operations: FlightsSchedule, CheckAvailability, BookFlights, and MakePayment as identified in the sequence diagram in Figure 2. The flight booking dialogue proceeds as shown in Figure 3. The traveler supplies the values for DepartureCity, DestinationCity, and DepartureDate as input parameters for the FlightsSchedule operation. The

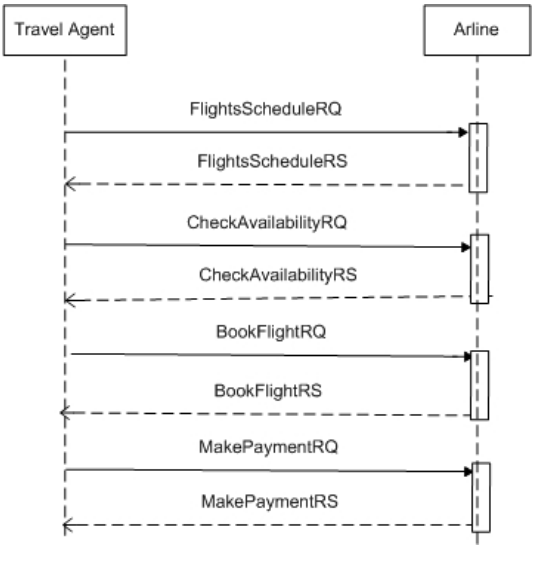



**Fig. 2.** Modified flight availability and booking message sequence

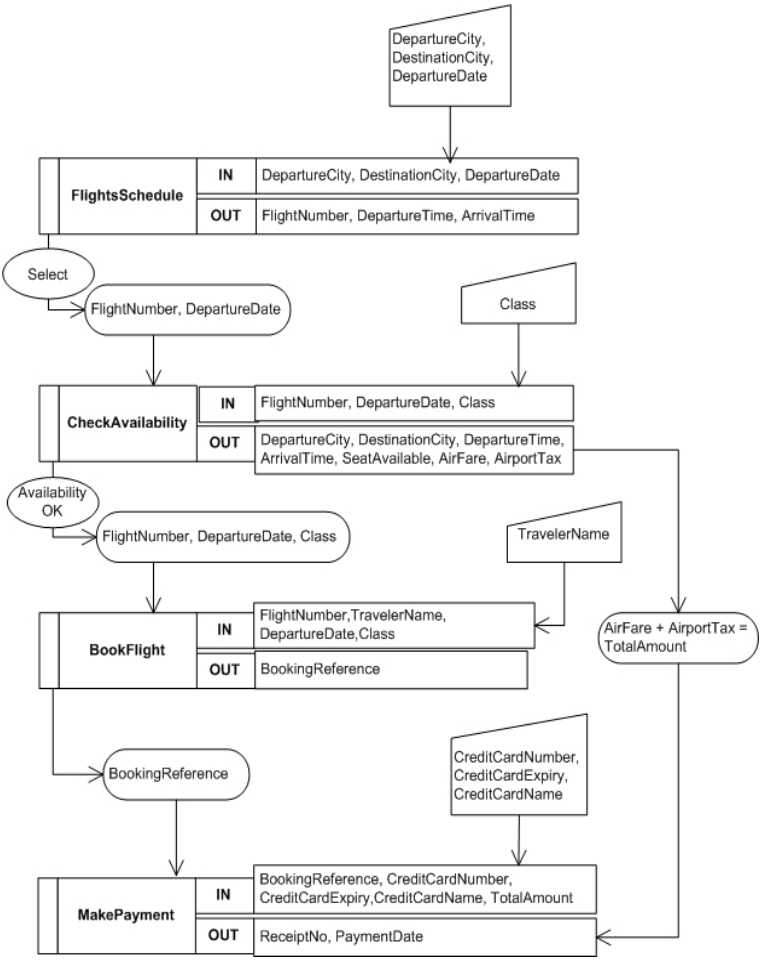



**Fig. 3.** Flight booking dialogue using fine granularity services

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.

Table 1 describes the service interfaces for the operations FlightsSchedule, CheckAvailability, BookFlights, and MakePayment showing the input and output parameters.

**Table 1.** Flight availability and booking service operations

| Operation | Input Parameters | Output Parameters |
|---|---|---|
| FlightsSchedule | DepartureCity, DestinationCity, DepartureDate | FlightNumber, DepartureTime, ArrivalTime |
| CheckAvailability | FlightNumber, DepartureDate, Class | DepartureCity, DestinationCity, DepartureTime, ArrivalTime, SeatAvailable, AirFare, AirportTax |
| BookFlight | FlightNumber,TravelerName, DepartureDate,Class | BookingReference |
| MakePayment | BookingReference, CreditCardNumber, CreditCardExpiry,CreditCardName, TotalAmount | ReceiptNo, PaymentDate |

### 3.2 Data Analysis of Service Interfaces

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 described by a common (global) database schema. Although not explicitly defined, this common schema is implicit in the industry-wide message specifications (i.e. OTA message schema specification, in this instance). 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 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 in Table 1 produces a set of 5 normalized relations that constitute the database schema associated with 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)

Given the above normalized relations, we can observe by inspecting Figure 3 that the assembly of the flight booking service takes place by passing the values of the key attributes between the service operations. For example, the composite key of the Availability relation (FlightNumber, DepartureDate, Class) forms the data flow between CheckAvailability and BookFlight operations, and 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 of the parameters would inhibit composition, e.g. removing Class from the dataflow between CheckAvailability and BookFlight operations would prevent the composition of the flight booking business function. Furthermore, the interface parameters are mutually compatible as they share common data parameters. In summary, it can be argued that the normalization of service interfaces results in service operations with high levels of cohesion, low levels of coupling and mutually compatible interfaces; properties that significantly improve service reusability and composability.

## 4 Describing Service Assembly Using Relational Algebra Operations

In the previous section we have described the process of decomposition of services into elementary service operation; service assembly involves reversing this process and combining services based on interfaces data parameters. We have noted that coupling between service operations involves data parameters that correspond to the keys of the underlying relations. We can use this observation to express services using relational algebra expressions or operator trees over the underlying schema [24]. For example, the FlightSchedule operation can be expressed as:

$$\mathbf{PJ}_{\text{FlightNumber,DepartureDate,ArrivalTime}}\ \mathbf{SL}_{\text{P1}}\mathbf{JN}_{\text{FlightNumber=FlightNumber}}(\text{Schedule, Flights}),$$

where PJ, SL, and JN represent projection, selection, and join operations respectively, and P1 is a selection predicate (e.g. DepartureCity = “Sydney“ and DestinationCity = “Melbourne“ and DepartureDate= “31-May-2007“).

We can now express the operation FlightSchedule and CheckAvailability in relational algebra, for clarity substituting values into the predicates using selection specification as shown below:

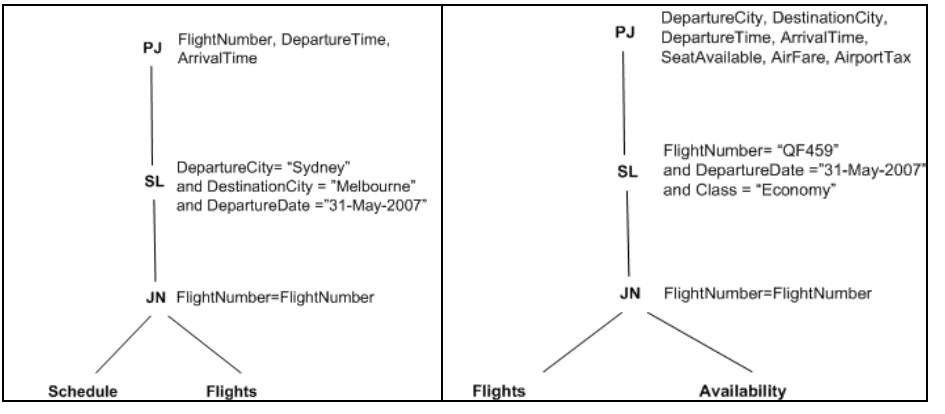
**FlightSchedule:**

$$\mathbf{PJ}_{\text{FlightNumber,DepartureDate,ArrivalTime}}\mathbf{SL}_{\text{DepartureCity=“Sydney“ and DestinationCity=“Melbourne“ and DepartureDate=“31-May-2007“}}\mathbf{JN}_{\text{FlightNumber=FlightNumber}}(\text{Schedule,Flights})$$

**CheckAvailability:**

$$\mathbf{PJ}_{\text{DepartureCity,DestinationCity,DepatureTime,ArrivalTime,SeatAvailable,Airfare,AirportTax}}$$
$$\mathbf{SL}_{\text{FighNumber=“QF459“ and DepartureDate=“31-May-2007“ and Class=“Economy“}}$$
$$\mathbf{JN}_{\text{FlightNumber=FlightNumber}}(\text{Flights,Availability})$$

Alternatively, the operations FlightSchedule and CheckAvailability can be expressed in the form of operator trees as shown in Figure 4. Figure 4(a) shows the operator tree for the FlightSchedule operation. The output parameters of the FlightSchedule operation (FlightNumber, DepartureDate, Arrival-Time) appear at the top of the operator tree, and the input parameters (DepartureCity="Sydney" and DestinationCity="Melbourne" and DepartureDate ="31-May-2007") form the predicate of the SL (select) operation. Now, assuming that the traveler selects FlightNumber = "QF459", DepartureDate ="31-May-2007" and Class = "Economy", this triplet of values forms the input for the CheckAvailability operation shown in Figure 4(b).




(a) (b)

**Fig. 4.** Operator tree representing the FlightSchedule (a) and CheckAvailability operation (b)

The output parameters of the CheckAvailability operation (DepartureCity, DestinationCity, DepartureTime, ArrivalTime, SeatAvailable, AirFare, AirportTax) appear at the top of the operator tree in Figure 4(b).

Having expressed service operations using relational algebra formalism we can now proceed and express service assemblies as an algebraic expression [24]. So that, for example we can combine the operations CheckAvailability and FlightSchedule to produce a composite AirAvailability operation:

**AirAvailability:PJ**$_{\text{DepartureCity,DestinationCity,DepatureTime,ArrivalTime,SeatAvailable,Airfare,AirportTax}}$
**SL**$_{\text{FighNumber="QF459" and DepartureDate="31-May-2007" and Class="Economy"}}$
**JN**$_{\text{FlightNumber=FlightNumber}}$(Flights,Availability,Schedule)

The expression for the AirAvailability operation uses the equivalence: R:$_{P1}$ **JN**$_{F}$ S:$_{P2}$ = R **JN** S: $_{P1 \text{ AND } P2 \text{ AND } F}$, where R and S are relations, P1 and P2 are selection predicates, and F is the join expression. Figure 5 show the resulting AirAvailability operation expressed as an operator tree. We now show the composite operation AirAvailability in the usual form that includes input and output data parameters, and can be mapped into a WSDL specification:

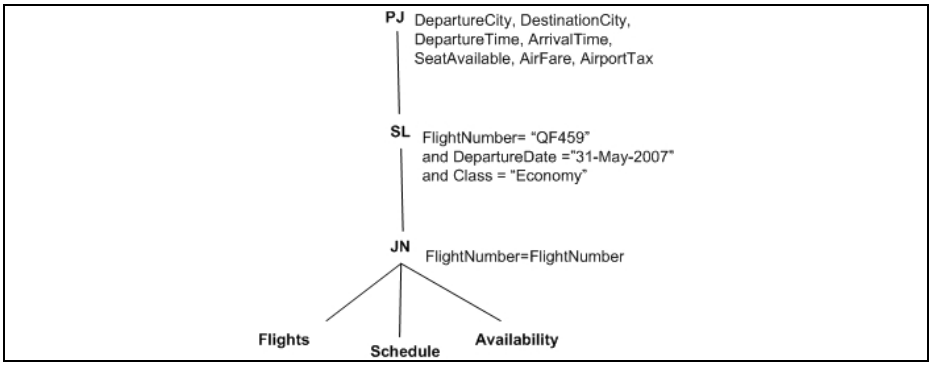



**Fig. 5.** Operator tree of a composite operation AirAvailability

**AirAvailability:**
(IN: FlightNumber, DepartureDate, Class,
OUT: DepartureCity, DestinationCity, DepatureTime, ArrivalTime,
SeatAvailable, Airfare, AirportTax)

Using this approach provides a formal framework for static service composition that allows decisions about the level of service aggregation to be based on considerations of tradeoffs between complexity of run-time dialogue (i.e. *chattiness* of services) on one hand, and software engineering properties of services such reusability, on the other hand. The designer may, for example, decide to implement the CheckAvailability and FlightSchedule service operations *internally* (i.e. within the service provider system) and externalize the composite operation AirAvailability, gaining the benefits of reuse (and composability) for internal applications, and at the same time reducing the number of messages needed to implement the flight booking dialogue. This solution is similar to the *remote façade* design pattern use to implement coarse-grained interfaces in object-oriented applications [25].

## 4 Conclusions and Related Work

Service composition methods range from industry standard approaches based on Web Services and BPEL [26] that focus on defining the workflow of Web Services execution, to Semantic Web approaches that employ AI techniques to automate service discovery and composition [27-28]. Service composition can be regarded as a special category of the software composition problem that has been investigated in the context of object-oriented software [29] and in the general area of software composition [30]. Many researchers have applied formal methods and developed specialized composition languages to address the problem of composition [31-32]. As noted in the introduction, service composition research mostly focuses on the dynamic (workflow) aspects of compositions. We have argued in this paper that from the viewpoint of service reuse and composability, the static part that involves the definition service operations and their interfaces is of key importance. The design of the inbound and outbound message structures determines the compatibility of service interfaces, and

consequently the composability of services into higher level business functions. The main contribution of this paper is to show that composability (and reuse) of services can be facilitated by designing services with compatible service interfaces and that service assembly can then be achieved by service aggregation over the key attributes of the underlying schema. We have also shown that relational algebra formalism can be applied to the problem of representing service operations, and defining service assemblies. Service decomposition and assembly framework based on data normalization and relational algebra operations can provide a theoretical framework for combining service operations to achieve desired business functionality and at the same time maintaining high levels of service reuse.

A number of aspects of this approach deserve further investigation. Firstly, the potential of using algebraic equivalence transformations for identifying alternative composition strategies and for optimizing the level of service granularity needs further study [24]. Another potential use of the relational algebra formalism is in the area of verification of the correctness of compositions, i.e. using algebra to prove the correctness of static compositions. Finally, the examples used in the previous section (section 3) involve services that represent *query* operations, i.e. operations that return data values given a set of input parameters. Service operations that result in state change, i.e. functions that generate new data values (e.g. Bookings and Payments) cannot be directly represented by algebraic expressions and require further analysis to enable their incorporation into this framework.

## Acknowledgements

We acknowledge the support of MŠMT ČR in the context of grant GAČR 201-06-0175 “Modification of the model for information management”.

## References

1. Thöne, S., Depke, R., Engels, G.: Process-oriented, flexible composition of web services with UML. In: Olivé, À., Yoshikawa, M., Yu, E.S.K. (eds.) ER 2003. LNCS, vol. 2784, pp. 390–401. Springer, Heidelberg (2003)
2. Feuerlicht, G.: Design of Service Interfaces for e-Business Applications using Data Normalization Techniques. Journal of Information Systems and e-Business Management, 1–14 (2005) ISSN 1617-98
3. Feuerlicht, G., Meesathit, S.: Design framework for interoperable service interfaces. In: The Proceedings of 2nd International Conference on Service Oriented Computing, New York, NY, USA, November 15 - 19, 2004, pp. 299–307 (2004) ISBN 1-58113-871-7
4. Wen-Li Dong, H.Y., Zhang, Y.-B.: Testing BPEL-based Web Service Composition Using High-level Petri Nets. In: 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), Hong Kong, pp. 441–444 (2006)
5. San-Yih Hwang, E.-P.L., Lee, C.-H., Chen, C.-H.: On Composing a Reliable Composite Web Service: A Study of Dynamic Web Service Selection. In: IEEE International Conference on Web Services (ICWS 2007), pp. 184–191 (2007)

6. Keita, F., Tatsuya, S.: Dynamic service composition using santic information. In: Proceedings of the 2nd international conference on Service oriented computing. ACM, New York (2004)
7. Freddy, L., et al.: Towards the composition of stateful and independent semantic web services. In: Proceedings of the 2008 ACM symposium on Applied computing. ACM, Fortaleza (2008)
8. Meng, X., et al.: A Dynamic Semantic Association-Based Web Service Composition Method. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence. IEEE Computer Society, Los Alamitos (2006)
9. Arkin, A., et al.: Web Services Business Process Execution Language (WS-BPEL). OASIS 2, Version, http://www.oasis.org
10. Yang, J.: Service-oriented computing: Web service componentization. Communications of the ACM 46(10), 35–40 (2003)
11. Hurwitz, J., Bloor, R., Baroudi, C.: Thinking from Reuse - SOA for Renewable Business (2006) (cited December 13, 2007), http://www.hurwitz.com/PDFs/IBMThinkingfromReuse.pdf
12. Feuerlicht, G., Wijayaweera, A.: Determinants of Service Resuability. In: The Proceedings of 6th International Conference on Software Methodologies, Tools and Techniques, SoMet 2006, Rome, Italy, November 7-9 (2007) ISBN 0922-6389
13. Sillitti, A., Vernazza, T., Succi, G.: Service oriented programming: A new paradigm of software reuse. In: Gacek, C. (ed.) ICSR 2002. LNCS, vol. 2319, pp. 268–280. Springer, Heidelberg (2002)
14. Dustdar, S., Schreiner, W.A.: A survey on web services composition. International Journal of Web and Grid Services 1(1), 1–30 (2005)
15. OTA, OTA Specifications (2008) (cited May 6, 2008), http://www.opentravel.org/Specifications/Default.aspx
16. Feuerlicht, G.: Implementing Service Interfaces for e-Business Applications. In: The Proceedings of Second Workshop on e-Business (WeB 2003), Seattle, USA (December 2003)
17. Eder, J., Kappel, G., Schrefl, M.: Coupling and Cohesion in Object-Oriented Systems. In: Finin, T.W., Yesha, Y., Nicholas, C. (eds.) CIKM 1992. LNCS, vol. 752. Springer, Heidelberg (1993)
18. Feuerlicht, G., Lozina, J.: Understanding Service Reusability. In: The Proceedings of 15th International Conference Systems Integration 2007, Prague, Czech Republic, June 10-12, 2007, pp. 144–150 (2007) ISBN 978-80-245-1196-2
19. Vogel, T., Schmidt, A., Lemm, A., Österle, H.: Service and Document Based Interoperability for European eCustoms Solutions. Journal of Theoretical and Applied Electronic Commerce Research 3(3) (2008) ISSN 0718–1876
20. Papazoglou, M.P., Heuvel, W.V.D.: Service-oriented design and development methodology. International Journal of Web Engineering and Technology 2(4), 412–442 (2006)
21. Baker, S., Dobson, S.: Comparing service-oriented and distributed object architectures. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 631–645. Springer, Heidelberg (2005)
22. Feuerlicht, G.: Service aggregation using relational operations on interface parameters. In: Georgakopoulos, D., Ritter, N., Benatallah, B., Zirpins, C., Feuerlicht, G., Schoenherr, M., Motahari-Nezhad, H.R. (eds.) ICSOC 2006. LNCS, vol. 4652, pp. 95–103. Springer, Heidelberg (2007)
23. Papazoglou, M., Yang, J.: Design Methodology for Web Services and Business Processes. In: Proceedings of the 3rd VLDB-TES Workshop, Hong Kong, pp. 54–64 (August 2002)

24. Ceri, S., Pelagatti, G.: Distributed databases principles and systems. McGraw-Hill Computer Science Series. McGraw-Hill, New York (1984)
25. Fowler, M.: Patterns of Enterprise Application Architecture. The Addison-Wesley Signature Series. Addison-Wesley, Reading (2002); Pearson Education, p. 533, ISBN 13: 9780321127426
26. Kloppmann, M., et al.: Business process choreography in WebSphere: Combining the power of BPEL and J2EE. IBM Systems Journal 43(2), 270 (2004)
27. Bleul, S., Weise, T., Geihs, K.: Making a Fast Semantic Service Composition System Faster. In: The Proceedings of The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007 CEC/EEE 2007, Tokyo, pp. 517–520 (2007) ISBN 0-7695-2913-5
28. Chen, L., et al.: Towards a Knowledge-Based Approach to Semantic Service Composition. LNCS, pp. 319–334. Springer, Heidelberg (2003)
29. Nierstrasz, O., Meijler, T.D.: Research directions in software composition. ACM Computing Surveys (CSUR) 27(2), 262–264 (1995)
30. Nierstrasz, O.M., et al.: Object-oriented software composition. Prentice Hall, Englewood Cliffs (1995)
31. Kane, K., Browne, J.C.: CoorSet: A Development Environment for Associatively Coordinated Components. LNCS, pp. 216–231. Springer, Heidelberg (2004)
32. Scheben, U.: Hierarchical composition of industrial components. Science of Computer Programming 56(1-2), 117–139 (2005)