Service-oriented Computing and Applications manuscript No.
(will be inserted by the editor)

A Formal Model for the Interoperability of Service Clouds

Hui Ma, Klaus-Dieter Schewe,
Bernhard Thalheim, Qing Wang

Received: 7 August 2010 / Revised: 6 October 2011 / Accepted: 22 December 2011

Abstract Large-scale service-oriented computing is based on the idea that services from various servers are combined into one distributed application. Referring to a collection of services on one server as a "service cloud" the problem investigated in this paper is to define formal high-level specifications of such distributed applications and to enable the location of suitable services for them. Based on the language-independent model of Abstract State Services (AS^2s), which serves as a universal integrated model for data and software as services, we extend AS^2s by high-level action schemes called "plots" as a means to specify permitted sequences of service operations. On these grounds we develop a model for service mediators, i.e. specifications of composed services in which service slots have to be filled by actual services, and investigate matching conditions for slots of mediators and services. For a services to match a slot in a mediator, a (generalised) projection of the mediator must comply with the plot of the service. Furthermore, the service must be semantically adequate, which requires the use of a service ontology.

Keywords service cloud, abstract state service, service-oriented computing, service mediation, service ontology

Hui Ma
Victoria University of Wellington, School of Engineering and Computer Science, Wellington, New Zealand, hui.ma@ecs.vuw.ac.nz

Klaus-Dieter Schewe
Software Competence Center Hagenberg & Christian-Doppler-Laboratory for Client-Centric Cloud Computing, Johannes-Kepler-University, Hagenberg, Austria,
kd.schewe@scch.at kd.schewe@cdcc.faw.jku.at

Bernhard Thalheim
Christian-Albrechts-University Kiel, Department of Computer Science, Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Qing Wang
University of Otago, Department of Information Science, Dunedin, New Zealand, qwang@infoscience.otago.ac.nz

1 Introduction

A common slogan in cloud computing claims that "everything becomes a service", which suggests to consider cloud computing as an umbrella for service orientation in the large with the world-wide web as the key medium. In fact, it would be possible to consider clouds as repositories of software services that are built on top of platforms and infrastructure services. Key differences arise with respect to ownership and usage rights. For instance, in the "software as a service" model (SaaS) everything is owned by the service provider, whereas in the "infrastructure as a service" model (IaaS) only the basic hardware infrastructure is owned by provider, but used by the client, whereas the software is owned by the client. Whether this software is again offered as a service and thus used by others is a decision of the client who could at the same time become a software service provider. Therefore, in this paper we adopt a loose use of the term "service cloud" to refer to a repository of data and software services well knowing that there is more to cloud computing than only a functional view of services.

Despite this big interest in the area, and the many ideas and systems that have been created many fundamental questions have still not been answered. For instance, a web service could be almost anything, a simple function, a data warehouse, or a fully functional Web Information System, as long as it is made available via the world-wide web. The unifying characteristic is that content, functionality and sometimes even presentation are made available for use by human users or other services. However, the commonly used notion of web service would not capture all of these.

1.1 Our Contribution

In this article we first extend the formal model of Abstract State Services (AS^2s) by formalising the notion of *plot* of a service, which specifies algebraically how a service can be used. This was left implicit in the original work on AS^2s [21]. The notion of plot is a term adopted from the movie business that has already been used for long time in the context of web information systems [26]. It actually captures the possible sequencing of service operations, which is only implicitly present in the AS^2 model. For this we exploit Kleene algebras with test, which are known to be the most expressive formalism to capture propositional process specifications. So adding plots to the AS^2 model is a little new extension.

The key contribution of this paper, however, is the introduction of *service mediators*, which mediate the collaboration of services. For this we exploit plots with open slots for services to specify intended service-based applications on a high level of abstraction. The novel idea is to specify service-oriented applications that involve yet unknown component services. We then formally define matching criteria for services that are to fill the slots. A problem in finding such matching criteria is the fact that we would like to be able to skip component operations of services and change their order.

This enhances the work on service composition, which is already a well-explored area in service computing with respect to services that are understood functionally. In the AS^2 model this corresponds to the service operations rather than the services as a whole. More precisely, what we actually need to compose are "runs" of services that are determined by the plot. What makes our contribution even more interesting is that we investigate conditions, under which particular service operations can be removed or

their order can be changed. This leads to the rather complicated matching conditions between services and slots in mediators.

Finally, slots in mediators only make sense, if the services that could match them are located somehow. Here we adapt the idea of a service ontology, which is already omnipresent in the area of the semantic web, also in our previous work in [22]. In the paper we exploit a variant of DL_LITE, but other description logics could be used in the same way.

1.2 An Application Scenario

Suppose we want to develop an integrated complete service for conference trip organisation. To make it simple we anticipate only three parts: registration for the conference, booking of accommodation and booking of the travel – for simplicity let this be only a flight. We can imagine that for all these components web-based services exist and how these are organised. A rough picture would be the following:

- For registration we would have to provide personal data, author information and possible discounts, if applicable plus payment information.
- For flight booking we have to search for flights, select the most suitable one, and provide again personal data and payment information, say credit card details.
- For accommodation booking we have to search for available hotels, select a hotel and a room, and provide again personal data and payment information.

We observe some particularities here. For flight and accommodation booking there may be several competing services. As we do not care which one should be used, we would have to access these services in parallel for the search, combine the results, and continue with one of the services only, once a selection has been made. Furthermore, all the services involved contain redundant parts for entering personal data and credit card details, but we only want to provide such data once. A solution might be to collect these data locally and to push them through to the corresponding services when required.

Next we have the choice to either adopt a bottom-up or a top-down approach to system specification. In the former case we would search for suitable services and compose them. In the latter case we try to specify the composed system with slots that are to be filled in by not yet known services. Let us concentrate on the top-down aproach.

In this case we specify a mediator consisting of local components and yet unknown services, each of which providing some service operations. In our scenario we would need one conference registration service and several services for flight and accommodation booking. The mediator would have to specify the flow of data in and out of the services. Figure 7 illustrates this idea for our application scenario. It can be seen that ideally it will be necessary that services can interact with each other, and that they employ several service operations users or other services can interact with.

The next step would be to search for services that match the “slots” in the mediator. For matching we have to fulfil functional conditions regarding the input and output, conditions that refer to the application domain (so we really get flight booking services), and conditions that align the flow of data in the mediator with those in the individual services. Figure 4 illustrates the flow of data within the individual services, so the projection of the data flow from the mediator must be compatible with this.

In addition to matching slots in the mediator with suitable services we have to make a selection, as there may be many matching services. Selection may depend on availability, performance, security, costs and other non-functional service-level agreements. These criteria are beyond the scope of our work here.

Finally, after search and selection of suitable services we instantiate the mediator, thus replace the slots by actual services. For our application scenario this is illustrated in Figure 8, in which two flight booking services, but only one hotel booking service have been assumed to be selected. This instantiation gives a sketch of a process that solves the problem we started with. For execution there is still more work to be done. For instance, intermediate results have to be stored somewhere, i.e. the process may require to be extended by services docking at a yet unknown cloud. However, the paper at hand is not concerned with refinement and final implementation of the resulting service.

In the technical sections of this paper we will use this simple scenario to illustrate our key contributions.

1.3 Related Work

As the research reported in this paper addresses service specification and description, service discovery, service composition and orchestration of service-based processes, it naturally links to commonly used technology such as the Web Services Description Language (WSDL) [11] that takes the role of describing the parameters that are needed to use a web service, service publishing with the use of a UDDI registry [23], which addresses universal description, discovery and integration, service ontologies such as WSMO, which is a full ontology language dedicated to web services [15] based on the web services modelling framework WSMF [14], and service orchestration languages such as BPEL.

This further links our research to the area of semantic web services, which aim at enabling semantic search for services exploiting the idea of the semantic web [7], in particular ontologies [16]. The work on WSDL-S extends WSDL by adding attributes, but does not yet provide a full ontology [1], whereas WSMO is a full ontology language dedicated to web services. Particular emphasis in semantic web services is given to functional descriptions as e.g. in [18] and non-functional properties as e.g. in [24].

Let us take BPEL and WSDL as representatives of the state of the art to explain the differences in our work. In a BPEL orchestration a process flow is specified, in which individual components are web services that can be run sequentially, in parallel or even iterated. However, the services as such appear as black boxes without being interleaved. This implies that any service component that appears redundantly in several component services will be used repeatedly. For instance, provision of an address or payment details may be requested more than once. Our notion of mediator tries to avoid this, and the key for this is the plot, which specifies the flow between service operations. As a consequence of taking interleaving of services and thus also communication between services into the model we also need a more sophisticated notion of service and of service composition. Every service described by WSDL can be considered as an AS^2, but the opposite does not hold.

Further to these differences to related work our view of a service cloud as a pool of resources is also used in the meme media architecture [32], which is based on research that already started in the second half of the 1980s. Naturally, this links our work

to research on service-oriented architectures (SOA) (see e.g. [10,20]), service-oriented computing (SOC), and web services (see e.g. [2,4,6,11]). In an effort to consolidate and integrate research activities, the Service-Oriented Computing Research Roadmap [25] has been proposed. Service foundations, service composition, service management and monitoring, and service-oriented engineering have been identified as core SOC research themes as exemplified by WSDL [11], the OASIS web services standard [4], the UDDI standard for universal description, discovery and integration [23] and the SOAP protocol for simple object access [30]. Web service integration has become a highly relevant research topic [6] including transaction processing (e.g. WS-Transaction [12] and WS-Coordination [13]).

We developed the model of Abstract State Services (AS^2s) in [21] as a universal, formal model for services. It is based on Abstract State Machines (ASMs) [9], which have already been shown their usefulness in many areas, e.g. modelling web services [3]. AS^2s abstract from and generalise our research on Web Information Systems [26], integrate the customised ASM thesis for database transformations [28], and base service composition on general ideas about component-based systems engineering [27]. Thus, AS^2s provide an integrated model for data and software as a service.

Remark 1 Actually, Gurevich's seminal work on Abstract State Machines in [17,8] is first a language-independent clarification of the notion of (sequential) algorithm by means of a few intuitive postulates. Then he proves that algorithms defined this way are captured by (sequential) ASMs. As there are many formal languages that are equivalent to ASMs, they could have been used instead, though the proof may have become much harder. A decisive feature of ASMs is the exploitation of first-order Tarski structures for states (as common in mathematics), which permits specifications of algorithms on any level of abstraction, i.e. without tedious elaboration of encodings. Thus, "following the ASM approach" means to start from a language-independent definition of services by means of the AS^2 model. As shown in our previous work, AS^2s are of course captured by a variant of ASMs, but concrete specifications could exploit any other formalism that is equivalent.

A service ontology should contain at least a functional description of services by means of types, and pre- and postconditions, plus a categorical description of the application area by keywords, which can be formalised by description logics [5,33]. The choice of DL_Lite is only to exemplify the key ideas. Service mediation has also been addressed in the context of semantic web services, e.g. [31], but our model goes further. It also extends our first tentative ideas in [29].

1.4 Outline

In Section 2 we first present a brief overview of the ideas of the AS^2 model. Then we formally introduce the notion of a plot of a service. Section 3 is dedicated to our notion of a service cloud, which consists of several services plus a service ontology describing them for the sake of enabling search. Section 4 contains the core contribution of this paper, the notion of mediator plus its instantiation by services matching the mediator's slots. We conclude with a brief summary and outlook on future work in Section 5.

2 A Formal Model for Services

In this section we first give a brief summary of the formal model of *Abstract State Services* (AS^{2}s) [21]. We then extend this model to make the possible flows of service operations explicit. For this we exploit high-level action schemes, also called *plots*, which are common in Web Information Systems [26]. Plots can be expressed algebraically by Kleene algebras with tests [19], which gives a nice handle to use them for specifications of applications that exploit multiple services.

2.1 Abstract State Services

Abstract State Services are composed of two layers: a data(base) layer and a view layer on top of it. Both layers combine static and dynamic aspects. The assumption of an underlying database is no restriction, as it is hidden anyway, and data services will be formalised by views, which in the extreme case could be empty to capture pure functional services.

Starting with the database layer and following the general approach of Abstract State Machines (ASMs) [17] we may consider each database computation as a sequence of abstract states, each of which represents the database (instance) at a certain point in time plus maybe additional data that is necessary for the computation, e.g. transaction tables, log files, etc. In order to capture the semantics of transactions we distinguish between a wide-step transition relation and small step transition relations. A transition in the former one marks the execution of a transaction, so the wide-step transition relation defines infinite sequences of transactions. Without loss of generality we can assume a serial execution, while of course interleaving is used for the implementation. Then each transaction itself corresponds to a finite sequence of states resulting from a small step transition relation, which should then be subject to the postulates for database transformations [28], in particular states comprise a finite database part and a possibly infinite algorithmic part.

Definition 1 A *database system* DBS consists of a set $\mathcal{S}$ of states, together with a subset $\mathcal{I} \subseteq \mathcal{S}$ of initial states, a wide-step transition relation $\tau \subseteq \mathcal{S} \times \mathcal{S}$, and a set $\mathcal{T}$ of transactions, each of which is associated with a small-step transition relation $\tau_t \subseteq \mathcal{S} \times \mathcal{S}$ ($t \in \mathcal{T}$) satisfying the postulates of a database transformation over $\mathcal{S}$.

A *run* of a database system DBS is an infinite sequence $S_0, S_1, \ldots$ of states $S_i \in \mathcal{S}$ starting with an initial state $S_0 \in \mathcal{I}$ such that for all $i \in \mathbb{N}$ $(S_i, S_{i+1}) \in \tau$ holds, and there is a transaction $t_i \in \mathcal{T}$ with a finite run $S_i = S_i^0, \ldots, S_i^k = S_{i+1}$ such that $(S_i^j, S_i^{j+1}) \in \tau_{t_i}$ holds for all $j = 0, \ldots, k-1$.

Figure 1 illustrates the notion of a database system with a wide-step transition relation τ and runs of two transactions t_i by means of the small-step transition relations τ_{t_i}.

Views in general are expressed by queries, i.e. read-only database transformations. Therefore, we can assume that a view on a database state $S_i \in \mathcal{S}$ is given by a finite run $S_i = S_0^v, \ldots, S_\ell^v$ of some database transformation v with $S_i \subseteq S_\ell^v$ – traditionally, we would consider $S_\ell^v - S_i$ as the view. We can use this to extend a database system by views.

In doing so we let each state $S \in \mathcal{S}$ to be composed as a union $S_d \cup V_1 \cup \cdots \cup V_k$ such that each $S_d \cup V_j$ is a view on S_d. As a consequence, each wide-step state transition

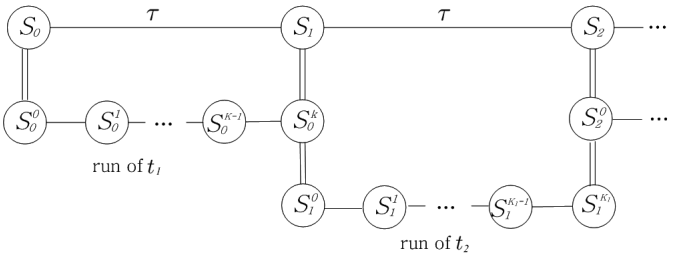

Fig. 1 Illustration of the notion of database system from Definition 1

becomes a parallel composition of a transaction and an operation that "switches views on and off". This leads to the definition of an Abstract State Service (AS^2).

Definition 2 An *Abstract State Service* (AS^2) consists of a database system DBS, in which each state $S \in \mathcal{S}$ is a finite composition $S_d \cup V_1 \cup \cdots \cup V_k$, and a finite set $\mathcal{V}$ of (extended) views. Each view $v \in \mathcal{V}$ is associated with a database transformation q_v such that for each state $S \in \mathcal{S}$ there are views $v_1, \ldots, v_k \in \mathcal{V}$ with finite runs $S_d = S_0^j, \ldots, S_{n_j}^j = S_d \cup V_j$ of v_j $(j = 1, \ldots, k)$. Each view $v \in \mathcal{V}$ is further associated with a finite set $\mathcal{O}_v$ of (service) operations $o_1, \ldots, o_n$ such that for each $i \in \{1, \ldots, n\}$ and each $S \in \mathcal{S}$ there is a unique state $S' \in \mathcal{S}$ with $(S, S') \in \tau$. Furthermore, if $S = S_d \cup V_1 \cup \cdots \cup V_k$ with V_i defined by v_i and o is an operation associated with v_k, then $S' = S'_d \cup V'_1 \cup \cdots \cup V'_m$ with $m \geq k - 1$, and V'_i for $1 \leq i \leq k - 1$ is still defined by v_i.

In a nutshell, in an AS^2 we have view-extended states, and each service operation associated with a view induces a transaction on the database, and may change or delete the view it is associated with, and even activate other views. These service operations are actually what is exported from the database system to be used by other systems or directly by users. Figure 2 illustrates this notion of state in an AS^2 and the effect of applying a service operation o, which induces a transaction t.

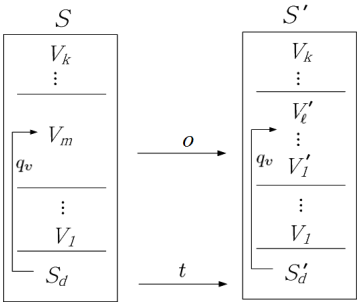

Fig. 2 Illustration of the notion of state in Definition 2

Note that for each view v the defining query, i.e. the database transformation q_v, can be considered itself a service operation. This simply reflects the fact that data

that is made available on the web can be extracted and stored or processed elsewhere. In particular, we have the extreme cases of a *pure data service*, in which no service operations would be associated with a view v, i.e. $\mathcal{O}_v = \emptyset$, and a *pure functional service*, in which the view v is empty.

A formalisation of database transformations is beyond the scope of this paper. In a nutshell, the postulates require a one-step transition relation between states (sequential time postulate), states as (meta-finite) first-order structures (abstract state postulate), necessary background for database computations such as complex value constructors (background postulate), limitations to the number of accessed terms in each step (bounded exploration postulate), and the preservation of equivalent substructures in one successor state (genericity postulate) [28].

Remark 2 We should note that this definition of services by means of postulates is indeed language-independent. To be precise, AS^{2}s only capture services functionally, whereas quality aspects, domain aspects as well as agreements regarding the use of services (ownership, rights, obligations, legal aspects, etc.) are beyod the scope of the model. AS^{2}s can be specified by ASMs, but they can also specified (and implemented) by any other suitable language.

Remark 3 We should further note that the AS^2 model distinguishes between *services* and *service operations*, whereas the latter ones are quite often called already services. The reason is that services may require interaction, in which case more than one service operation is used.

2.2 High-Level Action Schemes

In order to use a service (expressed as an AS^2) a sequence of service operations has to be executed. However, the sequencing of several service operations in order to execute a particular task is only left implicit in the AS^2 model. We now make it explicit by algebraic expressions called *plots*.

According to [26] a plot is a high-level specification of an action scheme, i.e. it specifies possible sequences of service operations in order to perform a certain task. For an algebraic formalisation of plots in Web Information Systems (WISs) it was possible to exploit Kleene algebras with tests (KATs [19]). Then a plot is an algebraic expression that is composed out of elementary operations including 0, 1, and propositional atoms, binary operators $\cdot$ and $+$, and unary operators * and $^-$, the latter one being only applicable to propositions. With the axioms for KATs we obtain an equational theory that can be used to reason about plots.

Propositions and operations testing them are considered the same. Therefore, propositions can be considered as operations, and overloading of operators for operations and propositions is consistent. In particular, 0 represents `fail` or **false**, 1 represents `skip` or **true**, $p \cdot q$ represents a sequence of operations or a conjunction, if both p and q are propositions, $p + q$ represents the choice between p and q or a disjunction, if both p and q are propositions, p^* represents iteration, and $\bar{p}$ represents negation.

For our purposes here, the definition of plots for AS^{2}s requires that we leave the purely propositional ground. The service operations give rise to *elementary processes* of the form

$$\varphi(\mathbf{x})\ op[\mathbf{z}](\mathbf{y})\ \psi(\mathbf{x}, \mathbf{y}, \mathbf{z}),$$

in which *op* is the name of a service operation, $\mathbf{z}$ denotes input for *op* selected from the view v with $op \in Op_v$, $\mathbf{y}$ denotes additional input from the user, and φ and ψ are first-order formulae denoting pre- and postconditions, respectively. The pre- and postconditions can be void, i.e. **true**, in which case they can be simply omitted. Furthermore, also simple formulae $\chi(\mathbf{x})$ – again interpreted as tests checking their validity – constitute elementary processes. With this we obtain the following definition.

Definition 3 The set of *process expressions* of an AS^2 is the smallest set $\mathcal{P}$ containing all elementary processes that is closed under sequential composition $\cdot$, parallel composition $\|$, choice $+$, and iteration *. That is, whenever $p, q \in \mathcal{P}$ hold, then also pq, $p\|q$, $p + q$ and p^* are process expressions in $\mathcal{P}$.

The *plot* of an AS^2 is a process expression in $\mathcal{P}$.

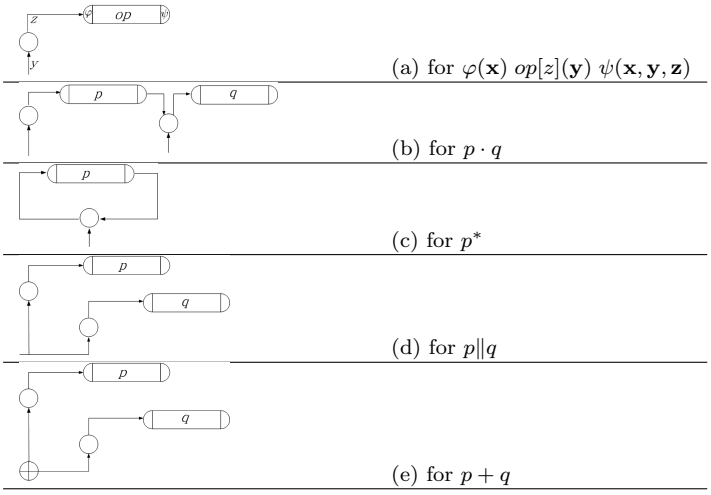

Fig. 3 Illustration of process expressions in Definition 3: Picture (a) shows an elementary process with service operation *op*, user input $\mathbf{y}$ and input $\mathbf{z}$ selected from the defining view. The defining view is indicated by the circle, but left anonymous. The two half circles capture the pre- and postcondition, respectively. The other pictures (b) – (e) show sequential composition, iteration, parallel composition and choice, respectively.

Example 1 Let us look at some very simplistic examples. For a flight booking service we may have the following (purely sequential) plot:

get_itineraries[](d) select_itinerary[i]()
personal_data[](t) confirm_flight[](y)
pay_flight[](c)

Here the parameters d, i, t, c and y represent dates, selected itinerary, traveller data, card details, and a Boolean flag for confirmation.

Similarly, the following expression represents another plot for accommodation booking:

get_hotels[](*d*) select_hotel[*h*]()
select_room[*r*]() personal_data[](*t*)
confirm_hotel[](*y*) pay_accommodation[](*c*)

Here the parameters h and r represent the selected hotel and room.

Finally, the expression personal_data[](*t*) (papers[](*p*) ∥ discount[](*d*′) payment[](*c*) represents the plot of a conference registration service.

Figure 4 shows the graphical illustrations of these plots according to the legend defined in Figure 3. □

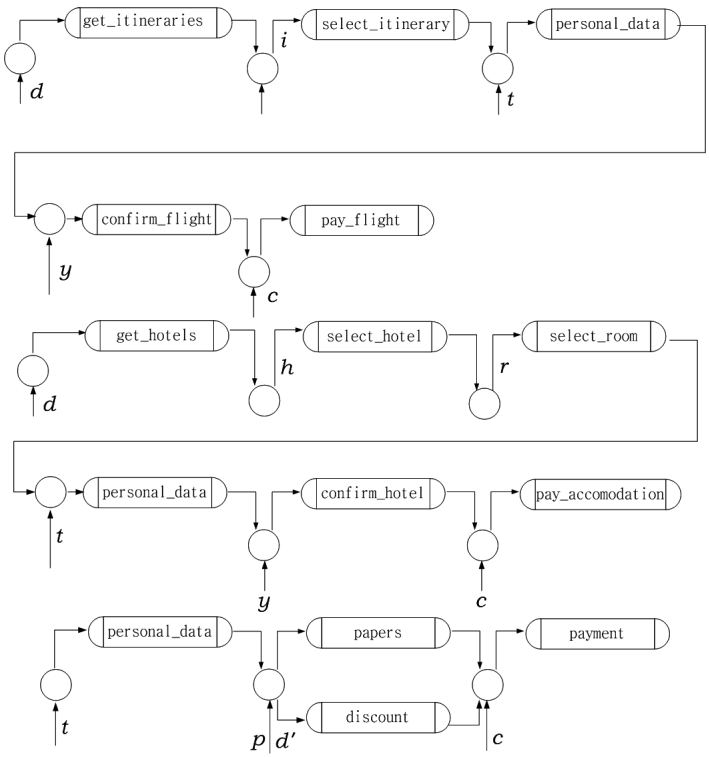

Fig. 4 Illustration of the plots for flight booking, accommodation booking, and conference registration

Note that the set of all instantiations of process expressions in $\mathcal{P}$ still defines a Kleene algebra with tests, but different to the work on Web Information Systems in [26] this algebra is not finitely generated. The sequences of service operations with instantiated parameters that are permitted by the plot define the semantics of the AS^2.

The following table summarizes the notation we used for processes:

$op[\mathbf{z}](\mathbf{y})$	service operation named op with input $\mathbf{y}$ from the user and input $\mathbf{z}$ selected from the associated view
$\varphi(\mathbf{x})$	precondition for a service operation $op[\mathbf{z}](\mathbf{y})$
$\psi(\mathbf{x},\mathbf{y},\mathbf{z})$	postcondition for a service operation $op[\mathbf{z}](\mathbf{y})$
pq	sequential composition of processes p and q
$p\|q$	parallel composition of processes p and q
p^*	iteration of process p
pq	choice between processes p and q

3 Service Repositories

In order to locate services it is crucial that the service operations including the view defining queries that are made available are provided with an adequate description, which will allow a search engine to discover (with some certainty) the required services. Such a description should comprise three parts:

- a *functional* description of input- and output types as well as pre- and post-conditions telling in technical terms, what the service operation will do,
- a *categorical* description by inter-related keywords telling what the service operation does by using common terminology of the application area, and
- a *quality of service* (QoS) description of non-functional properties such as availability, response time, cost, etc.

The QoS description is not needed for service discovery and merely useful to select among alternatives, but neither functional nor categorical description can be dispensed with.

A functional description alone would be insufficient. For instance, a flight booking service operation requires an itinerary to be selected, so the input type could be specified as $\{(flight\_no : STRING,\ day : DATE,\ departure : TIME,\ class : CHAR,\ price : DECIMAL)\}$, i.e. the input is a finite set of tuples, each of which defines a flight number, departure day and time, the booking class and the price. The output type could be similar with a status (confirmed, waitlisted, unavailable) added for each flight segment, i.e. we have the type $\{(flight\_no : STRING,\ day : DATE,\ departure : TIME,\ class : CHAR,\ price : DECIMAL,\ status : STRING)\}$. A precondition could simply be that the selected itinerary is meaningful, i.e. flight numbers exist for the corresponding date and time, and are compatible. However, no meaningful post-condition can be specified, as the output depends on the status of the (hidden) flight database. Moreover, a booking service for railway tickets would require the same types, so the functional description does not indicate exactly what kind of service is offered.

As for the categorical description, the terminology has to be specified. This defines an ontology in the widest sense, i.e. we have to provide definitions of "concepts" and relationships between them, such that each offered service becomes an instantiation of one or several concepts in the terminology. In this way we adopt the fundamental idea of the "semantic web". In the following we will outline how description logics [5] can be exploited for service description.

Remark 4 As outlined above, the key to service discovery is a description of available services. Here we follow the already well accepted approach to exploit description logics for this task. The reason for the use of description logics (since its very beginnings over

30 years ago) is that they enable the definition of concepts by necessary and sufficient conditions, and the logics are kept so simple that classification, i.e. determining subsumption relationships, is decidable. Thus, a search requires a definition of the service sought by means of a complex concept. The well-known classification algorithms for description logics then can be used to determine all instances (in the ABox) matching the complex concept. As this is standard, we do not repeat any of this in the paper.

Remark 5 In the following we adapt the idea of a service ontology using description logics, which is already omnipresent in the area of the semantic web. In particular, we exploited a variant of DL_LITE, but any other description logic could be used in the same way.

3.1 Terminologies

As outlined, the functional, categorical and QoS description of services in a cloud requires the definition of an ontology. That is, we need a *terminological knowledge* layer (aka TBox in description logics) describing concepts and roles (or relationships) among them. This usually includes a subsumption hierarchy among concepts (and maybe also roles), and cardinality constraints. In addition, there is an *assertional knowledge* layer (aka ABox in description logics) describing individuals. Thus, services in a cloud constitute the ABox of an ontology, while the cloud itself is defined by the TBox.

In principle, instead of TBox and ABox we could use the more classical notions of schema and instance, and exploit any kind of data model. A query language associated with the used data model, could then be used to find the required services. In fact, description logics only provide rather limited logics with respect to expressiveness. There are two major reasons for giving preference to description logics:

1. Description logics use two important relationships, which due to the restrictions become decidable: subsumption and instantiation. Subsumption is a binary relationship between concepts (denoted as $C_1 \sqsubseteq C_2$) guaranteeing that all instances of the subsumed concept C_1 are also instances of the subsuming concept C_2. Instantiation defines a binary relationship between instances in the ABox and concepts in the TBox asserting that an element A of the ABox is an instance of a concept C in the TBox.
   Subsumption and instantiation together allow us to discover services that are more expressive than needed, but can be projected to a service just as required.
2. Concept and role names in the TBox can be subject to similarity search by a search engine. That is, the search engine could produce services that are similar (with a certainty factor) to the ones required with respect to the categorical description, and match the functional description.

Let us now look more closely into one particular description logic in the *DL-Lite* family (see [5]). For this assume that C_0 and R_0 represent not further specified specified sets of basic concepts and roles, respectively. Then *concepts* C and *roles* R are defined by the following grammar:

$$\begin{aligned} R &= R_0 \mid R_0^- \\ A &= C_0 \mid \top \mid \geq m.R \text{ (with } m > 0) \\ C &= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \end{aligned}$$

Definition 4 A *terminology* (or TBox) is a finite set $\mathcal{T}$ of assertions of the form $C_1 \sqsubseteq C_2$ with concepts C_1 and C_2 as defined by the grammar above.

Each assertion $C_1 \sqsubseteq C_2$ in a terminology $\mathcal{T}$ is called a *subsumption axiom*. Note that Definition 4 only permits subsumption between concepts, not between roles, though it is possible to define more complex terminologies that also permit role subsumption.

As usual, we use the shortcut $C_1 \equiv C_2$ instead of $C_1 \sqsubseteq C_2 \sqsubseteq C_1$. For concepts, $\bot$ is a shortcut for $\neg\top$, and $\leq m.R$ is a shortcut for $\neg \geq m+1.R$.

Definition 5 A *structure* $\mathcal{S}$ for a terminology $\mathcal{T}$ consists of a non-empty set $\mathcal{O}$ together with subsets $\mathcal{S}(C_0) \subseteq \mathcal{O}$ and $\mathcal{S}(R_0) \subseteq \mathcal{O} \times \mathcal{O}$ for all basic concepts R_0 and basic roles R_0, respectively. $\mathcal{O}$ is called the base set of the structure.

We first extend the interpretation of basic concepts and roles and to all concepts and roles as defined by the grammar above, i.e. for each concept C we define a subset $\mathcal{S}(C) \subseteq \mathcal{O}$, and for each role R we define a subset $\mathcal{S}(R) \subseteq \mathcal{O} \times \mathcal{O}$ as follows:

$$\mathcal{S}(R_0^-) = \{(y,x) \mid (x,y) \in \mathcal{S}(R_0)\}$$
$$\mathcal{S}(\top) = \mathcal{O}$$
$$\mathcal{S}(\geq m.R) = \{x \in \mathcal{O} \mid \#\{y \mid (x,y) \in \mathcal{S}(R)\} \geq m\}$$
$$\mathcal{S}(\neg C) = \mathcal{O} - \mathcal{S}(C)$$
$$\mathcal{S}(C_1 \sqcap C_2) = \mathcal{S}(C_1) \cap \mathcal{S}(C_2)$$
$$\mathcal{S}(C_1 \sqcup C_2) = \mathcal{S}(C_1) \cup \mathcal{S}(C_2)$$
$$\mathcal{S}(\exists R.C) = \{x \in \mathcal{O} \mid (x,y) \in \mathcal{S}(R) \text{ for some } y \in \mathcal{S}(C)\}$$
$$\mathcal{S}(\forall R.C) = \{x \in \mathcal{O} \mid (x,y) \in \mathcal{S}(R) \Rightarrow y \in \mathcal{S}(C) \text{ for all } y\}$$

Definition 6 A *model* for a terminology $\mathcal{T}$ is a structure $\mathcal{S}$, such that $\mathcal{S}(C_1) \subseteq \mathcal{S}(C_2)$ holds for all assertions $C_1 \sqsubseteq C_2$ in $\mathcal{T}$. A finite model, i.e. a model with a finite base set, is also called *instance* or ABox associated with $\mathcal{T}$.

Example 2 The general part of a service ontology could be defined by a terminology as follows:

$$\text{Service} \sqsubseteq \exists \text{name.Identifier} \sqcap \leq 1.\text{name} \sqcap \exists \text{address.URL} \sqcap$$
$$\exists \text{offered\_by.Provider} \sqcap \leq 1.\text{address} \sqcap \leq 1.\text{offered\_by}$$
$$\sqcap \exists \text{defining.Query} \sqcap \leq 1.\text{defining} \sqcap \exists \text{offers.Operation}$$
$$\text{Operation} \sqsubseteq \exists \text{associated\_with.Query} \sqcap \leq 1.\text{associated\_with}$$
$$\text{Data\_Service} \equiv \text{Query} \sqcap \geq 1.\text{defining}^-$$
$$\text{Functional\_Service} \equiv \text{Operation} \sqcap \geq 1.\text{offers}^-$$
$$\text{Service\_Operation} \equiv \text{Data\_Service} \sqcup \text{Functional\_Service}$$
$$\text{Service\_Operation} \sqsubseteq \exists \text{input.Type} \sqcap \leq 1.\text{input}$$
$$\exists \text{output.Type} \sqcap \leq 1.\text{output}$$
$$\text{Type} \sqsubseteq \exists \text{name.Identifier} \sqcap \leq 1.\text{name} \sqcap \exists \text{format.Format}$$

Here we used capital first letters to indicate concept names, and lower case letters for role names. Figure 5 illustrates this TBox. □

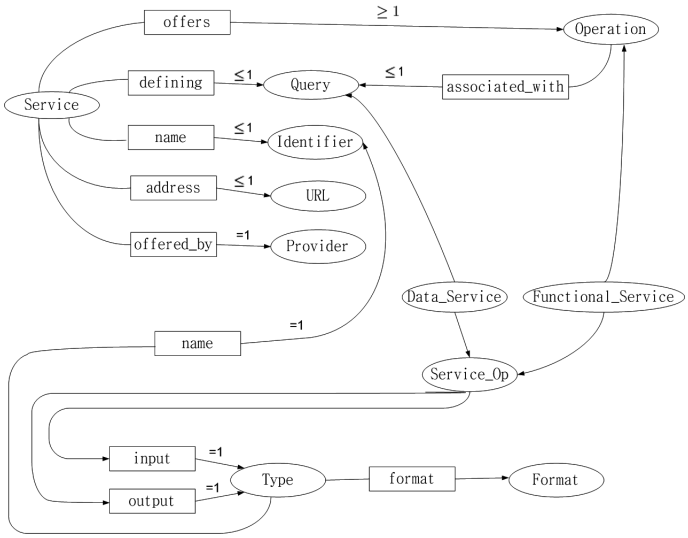

Fig. 5 Illustration of the top-level terminology for services

3.2 Functional and Categorical Description

As outlined above we expect the terminology $\mathcal{T}$ of a cloud to provide the functional, categorical and QoS description of its offered services.

The functional description of a service operation consists of input- and output-types as already indicated in Example 2, and pre- and post-conditions. For the types we need a type system with base types and constructors. For instance, the following grammar

$$t = b \mid \top \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t\} \mid [t] \mid \\ (a_1 : t_1) \oplus \cdots \oplus (a_n : t_n)$$

describes (the abstract syntax of) a type system with a trivial type $\top$, a non-further specified collection of base types b, and four type constructors $(\cdot)$ for record types, $\{\cdot\}$ for finite set types, $[t]$ for list types, and $\oplus$ for union types. Record and union types use field labels a_i.

The semantics of such types is basically described by their domain, i.e. sets of values $dom(t)$. Usually, for a base type b such as *Cardinal*, *Decimal*, *Float*, etc. the domain is some commonly known at most countable set with a common presentation. The domain of the trivial type contains a single special value, say $dom(\top) = \{\bot\}$. For constructed types we obtain the domain in the usual way:

$$dom((a_1 : t_1, \ldots, a_n : t_n)) = \\ \{(a_1 : v_1, \ldots, a_n : v_n) \mid a_i \in dom(t_i) \text{ for } i = 1, \ldots, n\}$$
$$dom(\{t\}) = \{A \mid A \subseteq dom(t) \text{ finite}\}$$
$$dom([t]) = \{[v_1, \ldots, v_k] \mid v_i \in dom(t) \text{ for } i = 1, \ldots, k\}$$
$$dom((a_1 : t_1) \oplus \cdots \oplus (a_n : t_n)) = \bigcup_{i=1}^{n} \{(a_i : v_i) \mid v_i \in dom(t_i)\}$$

In particular, a union type $(a_1 : \top) \oplus \cdots \oplus (a_n : \top)$ has the domain $\{(a_1 : \bot), \ldots, (a_n : \bot)\}$, which can be identified with the set $\{a_1, \ldots, a_n\}$, i.e. such types are in fact enumeration types.

It is no problem to add the specification of types to the general service terminology as outlined in Example 2 thereby defining part of the functional description.

Example 3 We can extend the terminology in Example 2 by the following axioms for types:

$$\text{Type} \equiv \text{Base\_type} \sqcup \text{Trivial\_type} \sqcup \text{Composed\_type}$$
$$\text{Composed\_type} \equiv \text{Record} \sqcup \text{Set} \sqcup \text{List} \sqcup \text{Union}$$
$$\text{Record} \sqsubseteq \forall \text{component.Field}$$
$$\text{Field} \sqsubseteq \exists \text{field\_name.Identifier} \sqcap \leq 1.\text{field\_name}$$
$$\sqcap \exists \text{type.Type} \sqcap \leq 1.\text{type}$$
$$\text{Union} \sqsubseteq \forall \text{component.Field}$$
$$\text{Record} \sqcap \text{Union} \sqsubseteq \bot$$
$$\text{Set} \sqsubseteq \exists \text{component.Type} \sqcap \leq 1.\text{component}$$
$$\text{List} \sqsubseteq \exists \text{component.Type} \sqcap \leq 1.\text{component}$$
$$\text{Set} \sqcap \text{List} \sqsubseteq \bot$$

Of course, the specification of composed types impacts directly on the format, which is defined by field names and the format for the component type(s). Nevertheless, this constraint can be handled by the specification of ABox assertions. □

In addition to the types, the functional description of a service operation includes pre- and post-conditions, which are defined by (first-order) predicate formulae. These formulae may contain further functions and predicates, which are subject to further (categorical) description.

Example 4 The terminology in Examples 2 and 3 can be further extended by the following axioms:

$$\text{Service\_Operation} \sqsubseteq \forall \text{pre.Condition} \sqcap \leq 1.\text{pre}$$
$$\sqcap \exists \text{post.Condition} \sqcap \leq 1.\text{post}$$
$$\text{Condition} \sqsubseteq \text{Formula} \sqcap \forall \text{uses.(Predicate} \sqcap \text{Function)}$$
$$\text{Predicate} \sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \neg \geq 1.\text{out}$$
$$\text{Function} \sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \exists \text{out.Type} \sqcap \leq 1.\text{out}$$

This would complete the functional part of the terminology. □

As shown at the beginning of this section, the functional description is insufficient for enabling service discovery, and the QoS description is only needed as a means to support the selection among several alternatives. The core of the service description by means of the terminology of a cloud is the categorical description, which refers to the standard terminology of the application area, and relates the used notions to each other.

There are no general requirements for the categorical description, as it depends completely on the application domain. However, it will always lead to subconcepts of the concept Service_Operation plus additional concepts and roles. It will also add more details to the predicates and functions used in the pre- and post-conditions.

Example 5 Let us look at booking services as required by a conference trip application with particular emphasis on flight booking. The categorical description may consist of the following axioms:

$$\begin{gathered}
\text{Booking} \sqsubseteq \text{Service_Operation} \sqcap \exists \text{initiator.Customer} \sqcap \\
\exists \text{initiated_by.Request} \sqcap \exists \text{receives.Acknowledgement} \\
\sqcap \exists \text{requires.Customer_data} \sqcap \exists \text{requires.Payment} \sqcap \\
\exists \text{receives.(Confirmation} \sqcup \text{Declination} \sqcup \text{Amendment)} \\
\text{Request} \sqsubseteq \exists \text{object.Booking_object} \sqcap \exists \text{date.DATE} \\
\text{Flight_booking} \sqsubseteq \text{Booking} \sqcap \forall \text{initiated_by.Flight_request} \\
\text{Flight_request} \sqsubseteq \text{Request} \sqcap \forall \text{object.Flight} \\
\text{Flight} \sqsubseteq \text{Booking_object} \sqcap \exists \text{number.Flight_number} \sqcap \\
\exists \text{carrier.Airline} \sqcap \exists \text{departure.Date} \sqcap \text{duration.Duration} \\
\sqcap \exists \text{origin.Airport} \sqcap \exists \text{destination.Airport}
\end{gathered}$$

This specification is of course incomplete, but it shows how to proceed. That is, a booking is defined by a service operation that is initiated by a request from a customer, it further requires customer data and payment, and leads to an acknowledgement plus a confirmation, declination or (suggested) amendment. The request for a booking contains at least a booking object – it could contain more than one – and a date. A flight booking is a booking that is initiated by a flight request, which is a request, in which all booking object are flights. Flights themselves must have at least a flight number, an airline, a departure date, a duration, and origin and destination airports. □

3.3 Service Clouds

We are now ready to define our formal model of a service cloud as a federation of services together with a descriptive ontology.

Definition 7 A *service cloud* is a finite collection $\{\mathcal{A}_i \mid i \in I\}$ of AS^{2}s together with their plots and a service terminology $\mathcal{T}$, such that the defining queries of views and the associated service operations of these AS^{2}s define an instance of $\mathcal{T}$.

Thus, the service terminology of a service cloud can be used to search for suitable services that match the slots of a service mediator.

The terminology of a service cloud enables the discovery of services, as it can be queried. However, the success of a search depends on the compatibility of the ontologies used by the service seeker and the service provider. If there is no common understanding of the terms used in the terminology, in particular in the categorical description, the service seeker may apply a service, which does not deliver the required functionality.

Remark 6 As stated in the introduction, the use of the term "cloud" for a repository of services together with semantic descriptions only refers to the functionality of the (software) services offered by the cloud. Of course, there is more to cloud computing than this, but for the purpose of describing a formal model of how services can be integrated into new large-scale distributed applications this definition of service cloud captures the most relevant aspects. Nonetheless, the agreements regarding availability,

performance, costing, and even legal aspects could very well be made part of the quality of service description in the terminology, which has no relevance for the purposes of this paper.

Finally, the following table summarises the notation we use for terminologies:

$\top$	top concept in a terminology
$\bot$	bottom concept in a terminology
$C_1 \sqsubseteq C_2$	concept C_1 subsumes concept C_2
$C_1 \equiv C_2$	concept C_1 is extensionally equivalent to concept C_2
$C_1 \sqcap C_2$	intersection of concepts C_1 and C_2
$C_1 \sqcup C_2$	union of concepts C_1 and C_2
$\neg C$	complement of concept C
$\exists R.C$	concept that has a role R leading to concept C
$\forall R.C$	concept for which all roles R lead to concept C
$\geq m.R$	concept with at least m roles R leading to concept C
$\leq m.R$	concept with at most m roles R leading to concept C

4 Creating Large-Scale Distributed Service-Oriented Applications

Let us now address the main problem handled in this paper, the specification and instantiation of large-scale distributed systems exploiting services. This problem goes beyond service composition, even more beyond the composition of service operations. One way to create such an application would be to start from a set of known services that are composed and extended by local components. This can be assumed to be well-explored.

The other way is to start from a specification of the composed specification, which can be taken as the plot of an AS^2. However, in this plot we assume that most service operations are yet unknown; we only know a categorical description for them. For instance, some of the service operations may belong to a flight booking service, so we have to locate corresponding services using the service ontology of a service cloud and match them with the plot of the composed service. That is, besides search for services we also need a notion of matching services.

The matching problem becomes particularly interesting, when we consider that services sought may be overlapping. For instance, when combining several booking services, each of them may contain a service operation for payment as well as one for gathering personal data. It would be not a very interesting composed applications, if such overlaps were not integrated. For the booking example this would mean to have only one payment operation and gather personal data only once.

4.1 Service Mediators

With the concept of *service mediators* we want to capture the plot of a composed AS^2. In other words, we want to define a plot of an application that is yet to be constructed. The key issue is that such mediators specify service operations to be searched for, which can then be used to realise the problem at hand in a service-oriented way.

In order to capture the idea to specify service requests we relax the definition of a plot in such a way that service operations do not have to come from the same

AS^2. Thus, in elementary processes we use prefixes to indicate the corresponding AS^2, so we obtain $\varphi(\mathbf{x})\ X : op[\mathbf{z}](\mathbf{y})\ \psi(\mathbf{x},\mathbf{y},\mathbf{z})$, in which X denotes a *service slot*, i.e. a placeholder for an actual service. Apart from this we leave the construction of the set of process expression as in Definition 3 with the only difference that also ℓ-$op\langle p\rangle$ is a process expression, whenever p is one. Here $\langle\cdot\rangle$ denotes a finite multiset constructor, i.e. we consider an arbitrary number of processes running in parallel, and ℓ-op denotes a multiset operation, which aggregates the query results of the different processes in the multiset.

Definition 8 A *service mediator* is a process expression with service slots. Furthermore, each service operation is associated with input- and output-types, pre- and post-conditions, and a concept in a service terminology.

Figure 6 shows the graphical elements that are needed in addition to those in Figure 3 to illustrate mediators.

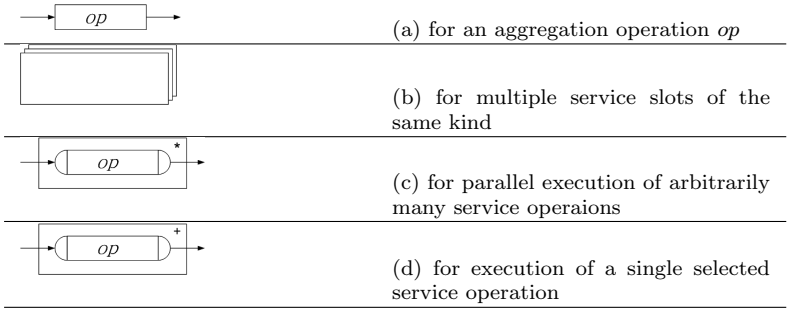

Fig. 6 Illustration of mediators in Definition 8

Example 6 Let us specify a service mediator for a conference trip application, which should combine conference registration, flight booking, and accommodation booking. Furthermore, replicative entry of customer data should be avoided, and confirmation of selection as well as payment should be unified in single local operations. This leads to the following specification:

personal_data[](t)
(X : papers[](p) ‖ X : discount[](d')
(union$\langle Y_j$: get_itineraries[](d)$\rangle$
Y_j : select_itinerary[i]() ‖
union$\langle Z_k$: get_hotels[](d)$\rangle$
Z_k : select_hotel[h]() Z_k : select_room[r]())
confirm[](y)
(Y_j : confirm_flight[](y) ‖ Z_k : confirm_hotel[](y))
pay[](c)
(Y_j : pay_flight[](c) ‖ Z_k : pay_hotel[](c) ‖ X : payment[](c))

Here the slots X, Y_j and Z_k refer to services for conference registration, flight booking, and accommodation booking, respectively, while the operations without prefix are considered to be local. For confirmation and payment the input parameters y and c

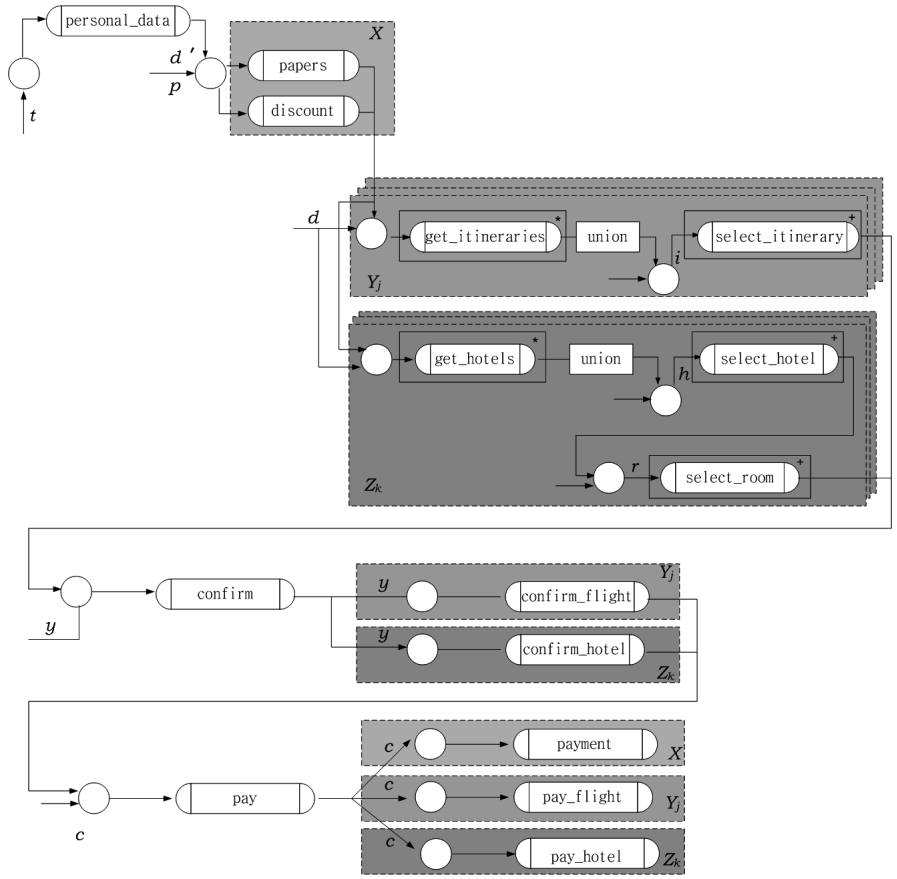

Fig. 7 Illustration of a mediator

are simply pushed through to the two booking services. This mediator is illustrated in Figure 7 using the notation from Figure 6. □

4.2 Service Matching

A service mediator specifies, which services are needed and how they are composed into a new plot of a composed AS^2. So we now need exact criteria to decide, when a service matches a service slot in a service mediator.

It seems rather obvious that in such a matching criteria for all service operations in a mediator associated with a slot X we must find matching service operations in the same AS^2, and the matching of service operations has to be based on their functional and categorical description. The guideline is that the placeholder in the mediator must be replaceable by matching service operations. Functionally, this means that the input for the service operation as defined by the mediator must be accepted by the matching service operation, while the output of the matching service operation must be suitable to continue with other operations as defined by the mediator. This implies that we

need supertypes and subtypes of the specified input- and output-types, respectively, in the mediator, as well as a weakening of the precondition and a strengthening of the postcondition. Categorically, the matching service operation must satisfy all the properties of the concept in the terminology that is associated with the placeholder operation, i.e. the concept associated with the matching service operation must be subsumed by that concept.

However, the matching of service operations is not yet sufficient. We also have to ensure that the projection of the mediator to a particular slot X results in a subplot of the plot of the matching AS^2.

Definition 9 A *subplot* of a plot p is a process expression q such that there exists another process expression r such that $p = q + r$ holds in the equational theory of process expressions.

The *projection* of a mediator m is a process expression p_X such that $p_X = \pi_X(m)$ holds in the equational theory of process expressions, where $\pi_X(m)$ results from m by replacing all placeholders $Y : o$ with $Y \neq X$ and all conditions that are irrelevant for X by 1.

Based on this definition it is tempting to require that the projection of a mediator should result in a subplot of a matching service. This would, however, be too simple, as order may differ and certain service operations may be redundant. We call such redundant service operations phantoms.

Definition 10 If for a condition $\varphi(\mathbf{x})$ appearing in a process expression p the equation $\varphi(\mathbf{x}) = \varphi(\mathbf{x})op[\mathbf{y}](\mathbf{z})$ holds, then $op[\mathbf{y}](\mathbf{z})$ is called a *phantom* of p.

That is, if the condition $\varphi(\mathbf{x})$ holds, we may execute the operation $op[\mathbf{y}](\mathbf{z})$ (or not) without changing the effect. Whenever $p = q$ holds in the equational theory of process expressions, and $op[\mathbf{y}](\mathbf{z})$ is a phantom of p with respect to condition $\varphi(\mathbf{x})$, we may replace $\varphi(\mathbf{x})$ by $\varphi(\mathbf{x})op[\mathbf{y}](\mathbf{z})$ in q. Each process expression resulting from such replacements is called an *enrichment of p by phantoms*.

Thus, we must consider projections of enrichments by phantoms, which leads us to the following definition.

Definition 11 An AS^2 $\mathcal{A}$ *matches* a service slot X in a service mediator m iff the following two conditions hold:

1. For each service operation $X : o$ in m there exists a service operation op provided by $\mathcal{A}$ such that
   - the input-type I_{op} of op is a supertype of the input-type I_o of o,
   - the output-type O_{op} of op is a subtype of the output-type O_o of o,
   - $pre_o \Rightarrow pre_{op}$ holds for the preconditions pre_o and pre_{op} of o and op, respectively,
   - $post_{op} \Rightarrow post_o$ holds for the postconditions $post_o$ and $post_{op}$ of o and op, respectively, and
   - the concept C_o associated with o in the service terminology subsumes the concept C_{op} associated with op.
2. There exists an enrichment m_X of m by phantoms such that building the projection of m and replacing all service operations $X : o$ by matching service operations op from $\mathcal{A}$ results in a subplot of the plot of $\mathcal{A}$.

Example 7 Let us look again at the simple service mediator in Example 6. We can assume that the local operation personal_data[](*t*) has the postcondition $person(t)$, and this is invariant under the service operations for itinerary and hotel selection. We can further assume that in both booking services the service operation personal_data[](*t*) is a phantom for $person(t)$. Thus, the mediator can enriched by phantoms, which results in:

personal_data[](*t*)
  (X : papers[](p) ‖ X : discount[](d')
    (union⟨Y_j : get_itineraries[](d)⟩
      Y_j : select_itinerary[i]() Y_j **: personal_data**[](t) ‖
    union⟨Z_k : get_hotels[](d)⟩
      Z_k : select_hotel[h]() Z_k : select_room[r]())
        Z_k **: personal_data**[](t)
confirm[](y)
  (Y_j : confirm_flight[](y) ‖ Z_k : confirm_hotel[](y))
pay[](c)
  (Y_j : pay_flight[](c) ‖ Z_k : pay_hotel[](c) ‖ X : payment[](c))

The added phantom operations are highlighted. The projection of this process expression to the services X, Y_j and Z_k, respectively, results exactly in the three plots in Example 1. □

The following table summarises the notation we use for mediators.

$X : op[\mathbf{z}](\mathbf{y})$	service operation named *op* in slot X with user-input $\mathbf{y}$ and input $\mathbf{z}$ selected from the associated view
$\langle X_j : op[\mathbf{z}](\mathbf{y})\rangle$	parallel run of arbitrarily many service operations *op* of the same kind in slots X_j
$op\langle \dots \rangle$	execution of multiset operation *op* aggregating the results of the parallel run in $\langle \dots \rangle$

4.3 Instantiation and Execution

Once matching services for all slots in a mediator have been found, we can built an instantiation of the mediator with real services, which serves as a high-level specification of a process that exploits several services.

Example 8 Consider again the mediator from Example 7. Suppose the slot X matches a conference registration service Conf_reg. Furthermore, let Fl_book and Flight be two service matching the slot Y_j, while Hotel_boo is a matching hotel booking service for Z_k. then the instantiated mediator becomes

personal_data[](*t*)
  (Conf_reg : papers[](p) ‖ Conf_reg : discount[](d')
    (union⟨Fl_book : get_itineraries[](d), Flight : get_itineraries[](d)⟩
      Fl_book : select_itinerary[i]() Fl_book : personal_data[](t) ‖ +
      Flight : select_itinerary[i]() Flight : personal_data[](t) ‖
    Hotel_boo : get_hotels[](d)
      Hotel_boo : select_hotel[h]() Hotel_boo : select_room[r]())

HOTEL_BOO : personal_data[](t)
confirm[](y)
((FL_BOOK : confirm_flight[](y) + FLIGHT : confirm_flight[](y)) ||
HOTEL_BOO : confirm_hotel[](y))
pay[](c)
((FL_BOOK : pay_flight[](c) + FLIGHT : pay_flight[](c)) ||
HOTEL_BOO : pay_hotel[](c) || CONF_REG : payment[](c))

This is illustrated in Figure 8.

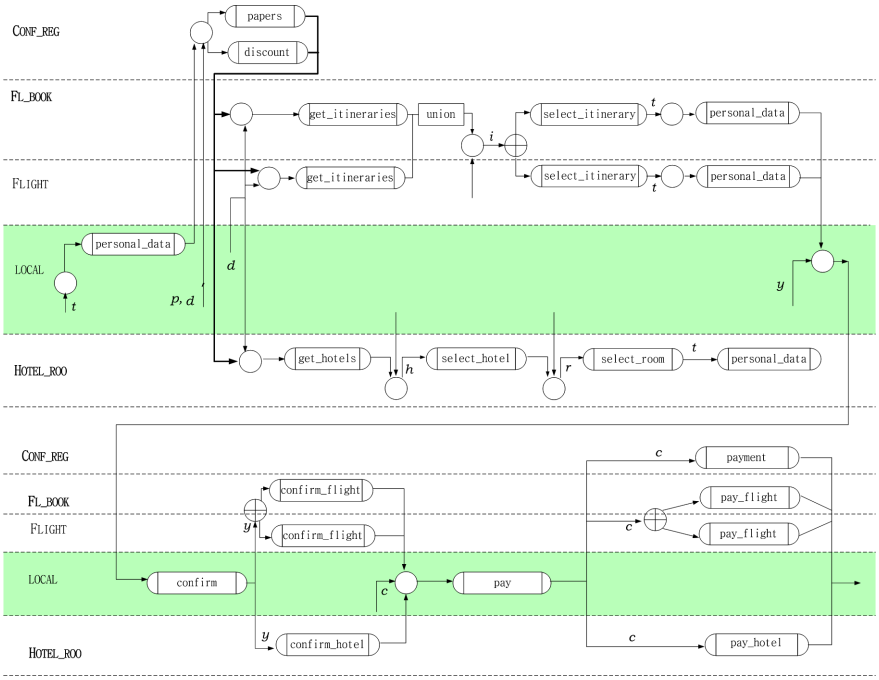

Fig. 8 Illustration of the instantiated mediator

Informally, this plot reads as follows. Start with gathering personal data t from a user of the composed service. This is a new operation performed locally. Then enter the service CONF_REG for conference registration. The first required service operation would be the entering of personal data, which can be done by passing on the already collected data, so the interaction actually continues with two service operations papers[](p) and discount[](d') for entering papers p and any potential discount d' for the conference fee. These two service operations can be accessed in parallel. With this, the interaction with service CONF_REG is already finished.

We then enter (in parallel) the services FL_BOOK, FLIGHT and HOTEL_BOO for flight and hotel booking, respectively. For the first two we first get itineraries using the service operation get_itineraries[](d), which are combined by the union operator. We then select an itineray i, which was provided either by FL_BOOK or FLIGHT. Depending on which service provided the selected itnerary, personal data are passed on to the

service without involving the user, while the other service is no longer considered. Analogously, a hotel and a room in that hotel is selected using service operations select_hotel[h]() and select_room[r]()), respectively, and again personal data are passed on.

The local operation confirm[](y) would actually have to present the selections made and request confrmation y, which is then passed on to the corresponding service operations confirm_flight[](y) and confirm_hotel[](y) of FL_BOOK (or FLIGHT) and HOTEL_BOO, respectively.

Finally, the local operation pay[](c) collects payment information and passes these on to the involved services, which then terminate. □

Remark 7 It is clear from the definition of mediators by means of KAT expressions that an instantiated mediator is only a very high-level specification of a large-scale distributed application that runs several services at the same time. This becomes further evident by Example 8. Refining and implementing such a specification would require several add-ons. First, the involved services have to be started and terminated, which usually involves a log-in and authentication process. Then data has to be passed from the mediation process to the individual services, which bypass the user interaction, i.e. a control component associated with the process is needed. Furthermore, output from several services is combined, and a selection made by a user is passed back to the originating services, while non-selection leads to service termination. This must also be handled by the control component. That is, from the high-level specification of a composed application to an executable software is still some work to be done. However, the specification shows what is needed in the implementation.

Remark 8 We used Kleene algebras with tests to specify mediators, and thus, also instantiations of mediators result in KAT expressions. The choice is merely motivated by the fact that KATs have the "right" expressiveness, as we only combine service operations, and only use their parameters in the specification. Furthermore, KAT expressions are very compact, and we can spare lengthy explanations of how to read a specification. We could have used ASMs instead or any other formalisms that allow us to specify processes. In particular, we could then exploit refinement in the same framework.

5 Conclusions

In this paper we continued our work on foundations of (web-based) service-oriented systems by extending our ASM-based model of Abstract State Services (AS^2s) by a service ontology, which permits searching for services. In doing so we observed that for a service seeker to be able to discover a suitable service the service specification is not sufficient nor would it be possible for a machine to understand the specification. Similarly, a functional description by means of types, and pre- and post-conditions is insufficient. This leads to the requirement that a description of services by means of a *service terminology* is necessary, which must enrich the functional service descriptions by keyword descriptions that describe the application area, and the available data and functions. This can be exploited to match the requests of a service seeker with the available services, and thus supports the semi-automatic selection of services.

Remark 9 We believe that in order to be completely sure about service selection, the final decision will be made by human experts. Nonetheless, the "semantic" description of services can be useful for making a pre-selection.

The real novel contribution of this paper, however, concerns service mediation, which we understand as the process of developing web-scale distributed applications, in which components are realised by exploiting service including interaction with them. We addressed the problem of service mediation starting from a high-level specification of an intended service-oriented application, in which "holes" are to be filled by suitable services. This led us to the formal model of *service mediators* with *service slots*. For the slots we provided matching conditions for services, which combine functional criteria by means of types, and pre- and postconditions, and categorical criteria capturing the application area. Thus, the matching conditions link to the description of services.

Furthermore, the sequencing of service operations of a matching service must comply with corresponding requirements of the mediator. Therefore, we used a concept of "plot" to define a high-level action scheme for services as well as for mediators.

With this work we add another tile to our theory, which permits the identification, search and composition of services in order to build a web-oriented application. While this is highly relevant to realise the vision of cloud computing on the web, there are still many open problems regarding allocation and optimised performance, selection among choices, and security and privacy. These open problems constitute the challenges for our continuing research.

We intend to extend our research in various directions. As the model of AS^2s is based on database transformations, we already know that all database transformations are captured (this was shown in [28]). It would thus be a natural consequence to investigate AS^2s that are bound to a particular data model, e.g. relational databases or XML, and to tailor service specifications to available languages such as XQuery.

Furthermore, solid results how well a service terminology for a particular application area can capture the service needs in that area would be desirable. The need for optimised service selection based on quality-of-service criteria constitutes another line of research to continue our present work. Finally, case studies applying the framework would be helpful to gain further insights.

In the paper we used the term of a "service cloud" to refer to a collection of services plus the necessary plots and a service terminology. While this is not a full model for clouds, we believe that it has an impact on cloud computing. First, the top of the cloud service stack is indeed the most relevant part. Infrastructure as a service becomes only interesting, if the infrastructure is used to take up software services. For platforms this is similar, and the whole stack can be understood in terms of ownership and access rights. For the simplest form of a company using infrastructure to run their own applications everything except the basic hardware infrastructure would be owned by the cloud user, but not shared with anyone else. Nonetheless, such a cloud user could offer web services, in which case we are in the picture of service clouds. Second, in this paper we only mentioned the possibility to describe non-functional properties of services in the service terminology. Theis could be elaborated to capturing many other aspects of cloud computing.

References

1. Akkiraju, R., et al. Web service semantics: WSDL-S, 2005.

http://www.w3c.org/Submission/WSDL-S.
2. Alonso, G., et al., Eds. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2003.
3. Altenhofen, M., Börger, E., and Lemcke, J. An abstract model for process mediation. In *Formal Methods and Software Engineering, 7th International Conference on Formal Engineering Methods (ICFEM 2005)* (2005), K.-K. Lau and R. Banach, Eds., vol. 3785 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 81–95.
4. Alves, A., et al. Web services business process execution language, version 2.0, 2007. OASIS Standard Committee, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.
5. Baader, F., et al., Eds. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
6. Benatallah, B., Casati, F., and Toumani, F. Representing, analysing and managing web service protocols. *Data and Knowledge Engineering 58*, 3 (2006), 327–357.
7. Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific American 285*, 5 (2001), 34–43.
8. Blass, A., and Gurevich, J. Abstract state machines capture parallel algorithms. *ACM Transactions on Computational Logic 4*, 4 (2003), 578–651.
9. Börger, E., and Stärk, R. *Abstract State Machines*. Springer-Verlag, Berlin Heidelberg New York, 2003.
10. Brenner, M. R., and Unmehopa, M. R. Service-oriented architecture and web services penetration in next-generation networks. *Bell Labs Technical Journal 12*, 2 (2007), 147–159.
11. Christensen, E., et al. Web services description language (WSDL) 1.1, 2001. http://www.w3c.org/TR/wsdl.
12. Cox, W., et al. Web services transaction (WS-Transaction), 2004. BEA Systems, IBM, Microsoft, http://dev2dev.bea.com/pub/a/2004/01/ws-transaction.html.
13. Feingold, W., and Jeyaraman, R. Web services coordination (WS-Coordination), version 1.1, 2007. OASIS Web Services Transaction WS-TX TC, http://docs.oasis-open.org/ws-tx/wstx-wscoor1.1-spec.pdf.
14. Fensel, D., and Bussler, C. The web service modeling framework WSMF. *Electronic Commerce Research and Applications 1*, 2 (2002), 113–137.
15. Fensel, D., et al. *Enabling Semantic Web Services*. Springer-Verlag, 2007.
16. Guarino, N. Formal ontology and information systems. In *Proceedings FOIS'98*. IOS Press, 1998, pp. 3–15.
17. Gurevich, J. Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic 1*, 1 (2000), 77–111.
18. Keller, U., Lausen, H., and Stollberg, M. On the semantics of functional descriptions of web services. In *Proceedings of the 3rd European Semantic Web Conference – ESWC 2006*. 2006.
19. Kozen, D. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems 19*, 3 (1997), 427–443.
20. Kumaran, S., et al. Using a model-driven transformational approach and service-oriented architecture for service delivery management. *IBM Systems Journal 46*, 3 (2007), 513–530.
21. Ma, H., Schewe, K.-D., Thalheim, B., and Wang, Q. A theory of data-intensive software services. *Service Oriented Computing and Its Applications 3*, 4 (2009), 263–283.
22. Ma, H., Schewe, K.-D., and Wang, Q. An abstract model for service provision, search and composition. In *Services Computing Conference - APSCC 2009*, M. Kirchberg et al., Eds. IEEE Asia Pacific, 2009, pp. 95–102.
23. Universal description, discovery and integration (UDDI). http://www.uddi.org.
24. O'Sullivan, J., Edmond, D., and Ter Hofstede, A. What is a service? Towards accurate description of non-functional properties. *Distributed and Parallel Databases 12*, 2-3 (2002), 117–133.
25. Papazoglou, M. P., and van den Heuvel, W.-J. Service oriented architectures: Approaches, technologies and research issues. *VLDB Journal 16*, 3 (2007), 389–415.
26. Schewe, K.-D., and Thalheim, B. Conceptual modelling of web information systems. *Data and Knowledge Engineering 54*, 2 (2005), 147–188.
27. Schewe, K.-D., and Thalheim, B. Component-driven engineering of database applications. In *Conceptual Modelling 2006 – Third Asia-Pacific Conference on Conceptual Modelling (APCCM 2006)*, M. Stumptner, S. Hartmann, and Y. Kiyoki, Eds., vol. 53 of *CRPIT*. Australian Computer Society, 2006, pp. 105–114.

28. Schewe, K.-D., and Wang, Q. A customised ASM thesis for database transformations. *Acta Cybernetica 19*, 4 (2010), 765–805.
29. Schewe, K.-D., and Wang, Q. A formal model for service mediators. In *Advances in Conceptual Modeling - Applications and Challenges (ER 2010 Workshops)*, J. Trujillo et al., Eds., vol. 6413 of *LNCS*. Springer-Verlag, 2010, pp. 76–85.
30. Simple object access protocol (SOAP). http://www.w3c.org/TR/soap.
31. Stollberg, M., Cimpian, E., Mocan, A., and Fensel, D. A semantic web mediation architecture. In *Proceedings CSWWS 2006*. 2006.
32. Tanaka, Y. *Meme Media and Meme Market Architectures*. IEEE Press, Wiley-Interscience, USA, 2003.
33. Web ontology language (OWL). http://www.w3c.org//OWL/.