A Formal Model for Service Mediators

Klaus-Dieter Schewe[1] and Qing Wang[2]

[1] Software Competence Center Hagenberg, Hagenberg, Austria
kd.schewe@scch.at
[2] University of Otago, Dunedin, New Zealand
qing.wang@otago.ac.nz

Abstract. In this paper we present a model of *service mediators*, which are high-level specifications of service-based applications. These mediators provide slots that are to be filled by actual services. Suitable services have to match the specification of the slots according to functional and categorical characteristics. Services and mediators are based on the ASM-based model of Abstract State Services.

1 Introduction

A lot of research is currently investigated into service-oriented architectures (SOA) (see e.g. [9,12,16]), service-oriented computing (SOC) [14], web services (see e.g. [1,2,3,6,11]), and cloud computing [4,10,24,25], which are all centred around related problems. In an effort to consolidate and integrate current research activities, the Service-Oriented Computing Research Roadmap [20] has been proposed. Service foundations, service composition, service management and monitoring, and service-oriented engineering have been identified as core SOC research themes.

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. Nonetheless there is an agreement that content, functionality and sometimes even presentation should be made available for use by human users or other services, which ressembles the view of a pool of resources in the meme media architecture [23]. The general idea is that media resources are extracted from any accessible source, wrapped and thereby brought into the generic form of a meme media object, and stored in a meme pool, from which they can be retrieved, re-edited, recombined, and redistributed.

Our research aims at laying the foundations of a theory of service-oriented systems. In particular, we try to answer the following fundamental questions:

- How must a general model for services look like capturing the basic idea and all facets of possible instantiations, and how can we specify such services?
- How can we search for services that are available on the web?
- How do we extract from such services the components that are useful for the intended application, and how do we recombine them?
- How can we optimise service selection using functional and non-functional (aka "quality of service") criteria?

J. Trujillo et al. (Eds.): ER 2010 Workshops, LNCS 6413, pp. 76–85, 2010.

In [18] we addressed the first of these problems by developing the formal model of Abstract State Services (AS^2s) as a general, formal model for services. It is based on Abstract State Machines (ASMs) [8], which have already proven their usefulness in many areas. AS^2s abstract from and generalise our research on Web Information Systems [21], and integrate the customised ASM thesis for database transformations [22]. The model of AS^2s captures in particular web services (see e.g. [1]).

In [19] we extended our work towards the second problem. We proposed a formalisation of clouds as service federations, which in addition provide an ontological description of the offered services. The ontology must 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. It can be formalised by description logics [5]. In a sense, this is the idea of the "semantic web", which enables at least the semi-automatic selection of services. Thus, AS^2s that are extended in this way by an ontological description of services capture the idea of "semantic web services" (see e.g. [17]).

In this paper we further extend our work addressing the third of the problems above. More precisely, we formalise the notion of *plot* of a service, which specifies algebraically how a service can be used. We then turn the idea around using plots with open slots for services to specify on a high level of abstraction intended service-based applications. We call such specifications *service mediators*, as the mediate the collaboration of participating services. We then have to formally define matching criteria for services that are to fill the slots. In accordance with the proposed ontological description of services we investigate matching conditions based on functional and categorical characteristics. For plots we adopt Kleene algebras with tests instead of looking into much more sophisticated process algebras [7], which would all be far too complicated to have a chance to obtain a decidable matching condition.

In the remainder of the paper we first elaborate on AS^2s and associated plots in Section 2. Note that in our original work on AS^2s in [18] plots were at best included implicitly. In Section 3 we introduce service mediators with service slots, and formally discuss matching between such slots and actual services. We conclude wth a brief summary and outlook.

2 Service Plots

Abstract State Services are composed of two layers: a database 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. The sequencing of several service operations in order to execute a particular task is only left implicit in the AS^2 model. In this section we make it explicit by algebraic expressions called *plots*.

2.1 Abstract State Services

Starting with the database layer and following the general approach of Abstract State Machines [13] 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 [22].

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 $j = 0, \ldots, k-1$.

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 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 database 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.

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) [22].

2.2 Algebraic Plots

According to [21] 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 [15]). 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^2s requires that we leave the purely propositional ground. The service operations give rise to *elementary processes* of the form

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

in which op is the name of a service operation, $\boldsymbol{z}$ denotes input for op selected from the view v with $op \in Op_v$, $\boldsymbol{y}$ 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 $\chi(\boldsymbol{x})$ – 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 $\mathcal{P}$.

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[]() $\|$ discount[](d') represents the plot of a conference registration service. □

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 [21] 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.

3 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.

3.1 Service Slots

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 $\varphi(\boldsymbol{x})\ X : op[\boldsymbol{z}](\boldsymbol{y})\ \psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, in which X denotes a *service slot*. Apart from this we leave the construction of the set of process expression as in Definition 3.

Definition 4. A *service mediator* is a process expression with service slots. Furthermore, each service operation is associated with input- and output-types, pre- and postconditions, and a concept in a service terminology.

Example 2. 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:

$$
\begin{aligned}
&L : \text{personal\_data}[](t)\ (X : \text{papers}[]() \parallel X : \text{discount}[](d') \\
&\qquad (Y : \text{get\_itineraries}[](d)\ Y : \text{select\_itinerary}[i]() \parallel \\
&\qquad\quad Z : \text{get\_hotels}[](d)\ Z : \text{select\_hotel}[h]()\ Z : \text{select\_room}[r]()) \\
&L : \text{confirm}[](y)\ (Y : \text{confirm\_flight}[](y) \parallel Z : \text{confirm\_hotel}[](y)) \\
&L : \text{pay}[](c)\ (Y : \text{pay\_flight}[](c) \parallel Z : \text{pay\_hotel}[](c))
\end{aligned}
$$

Here the three slots X, Y and Z refer to the three services for conference registration, flight booking, and accommodation booking, respectively, while the slot L refers to local operations. For confirmation and payment the input parameters y and c are simply pushed through to the two booking services. □

The work in [19] contains a precise definition of service terminologies on the grounds of description logics [5]. For this we 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 5. A *service terminology* 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*. The semantics of a terminology is defined by its models in the usual way [19]. Such a service terminology should comprise at least two parts: a *functional* description of input- and output types as well as pre- and postconditions telling in technical terms, what the service operation will do, and a *categorical* description by inter-related keywords telling what the service operation does by using common terminology of the application area (see [19] for details).

Example 3. With respect to the service operations in the plots in Example 1 the terminology has to specify that select_itinerary is a flight booking service operation. For this purpose the terminology may contain among others the following subsumption axioms:

$$\text{Booking} \sqsubseteq \text{Service\_Operation} \sqcap \exists \text{initiator.Customer} \sqcap$$
$$\exists \text{initiated\_by.Request} \sqcap \exists \text{receives.Acknowledgement}$$
$$\sqcap \exists \text{requires.Customer\_data} \sqcap \exists \text{requires.Payment} \sqcap$$
$$\text{Flight\_booking} \sqsubseteq \text{Booking} \sqcap \forall \text{initiated\_by.Flight\_request}$$

Further details can be found in [19]. □

3.2 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 6. 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. Formally, if for a condition $\varphi(\boldsymbol{x})$ appearing in a process expression p the equation $\varphi(\boldsymbol{x}) = \varphi(\boldsymbol{x})op[\boldsymbol{y}](\boldsymbol{z})$ holds, then $op[\boldsymbol{y}](\boldsymbol{z})$ is called a *phantom* of p. That is, if the condition $\varphi(\boldsymbol{x})$

holds, we may execute the operation $op[\boldsymbol{y}](\boldsymbol{z})$ (or not) without changing the effect.

Whenever $p = q$ holds in the equational theory of process expressions, and $op[\boldsymbol{y}](\boldsymbol{z})$ is a phantom of p with respect to condition $\varphi(\boldsymbol{x})$, we may replace $\varphi(\boldsymbol{x})$ by $\varphi(\boldsymbol{x})op[\boldsymbol{y}](\boldsymbol{z})$ 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 7. 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 4. Let us look again at the simple service mediator in Example 2. 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:

$$
\begin{aligned}
&L : \text{personal\_data}[](t)\ (X : \text{papers}[]() \parallel X : \text{discount}[](d')\\
&\quad (Y : \text{get\_itineraries}[](d)\ Y : \text{select\_itinerary}[i]()\ Y : \text{personal\_data}[](t) \parallel\\
&\qquad Z : \text{get\_hotels}[](d)\ Z : \text{select\_hotel}[h]()\ Z : \text{select\_room}[r]())\\
&\qquad\quad Z : \text{personal\_data}[](t)\\
&L : \text{confirm}[](y)\ (Y : \text{confirm\_flight}[](y) \parallel Z : \text{confirm\_hotel}[](y))\\
&L : \text{pay}[](c)\ (Y : \text{pay\_flight}[](c) \parallel Z : \text{pay\_hotel}[](c))
\end{aligned}
$$

The projection of this process expression to the services X, Y and Z, respectively, results exactly in the three plots in Example 1. □

4 Conclusion

In this paper we continued our research on foundations of a theory of web-based service-oriented systems. 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 provide 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 an ontological description of services.

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 and service-oriented 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.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architecture and Applications. Springer, Heidelberg (2004)
2. Altenhofen, M., Börger, E., Lemcke, J.: An abstract model for process mediation. In: Lau, K.-K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 81–95. Springer, Heidelberg (2005)
3. Alves, A., et al.: Web services business process execution language, version 2.0., OASIS Standard Committee (2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
4. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, Department for Electrical Engineering and Computer Sciences, University of California at Berkeley, US (2009)
5. Baader, F., et al. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. University Press, Cambridge (2003)
6. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. Data and Knowledge Engineering 58(3), 327–357 (2006)
7. Bergstra, J.A., Ponse, A., Smolka, S.A.: Handbook of Process Algebra. Elsevier Science B.V., Amsterdam (2001)
8. Börger, E., Stärk, R.: Abstract State Machines. Springer, Heidelberg (2003)
9. Brenner, M.R., Unmehopa, M.R.: Service-oriented architecture and web services penetration in next-generation networks. Bell Labs Technical Journal 12(2), 147–159 (2007)
10. Buyyaa, R., Yeo, C.S., Venugopala, S., Broberga, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599–616 (2009)
11. Christensen, E., et al.: Web services description language (WSDL) 1.1 (2001), `http://www.w3c.org/TR/wsdl`
12. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
13. Gurevich, J.: Sequential abstract state machines capture sequential algorithms. ACM Transactions on Computational Logic 1(1), 77–111 (2000)

14. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing 9, 75–81 (2005)
15. Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems 19(3), 427–443 (1997)
16. Kumaran, S., et al.: Using a model-driven transformational approach and service-oriented architecture for service delivery management. IBM Systems Journal 46(3), 513–530 (2007)
17. Kuropka, D., Tröger, P., Staab, S., Weske, M. (eds.): Semantic Service Provisioning. Springer, Heidelberg (2008)
18. Ma, H., Schewe, K.-D., Thalheim, B., Wang, Q.: A theory of data-intensive software services. Service Oriented Computing and Its Applications 3(4), 263–283 (2009)
19. Ma, H., Schewe, K.-D., Wang, Q.: An abstract model for service provision, search and composition. In: Kirchberg, M., et al. (eds.) Services Computing Conference - APSCC 2009, pp. 95–102. IEEE Asia Pacific (2009)
20. Papazoglou, M.P., van den Heuvel, W.-J.: Service oriented architectures: Approaches, technologies and research issues. VLDB Journal 16(3), 389–415 (2007)
21. Schewe, K.-D., Thalheim, B.: Conceptual modelling of web information systems. Data and Knowledge Engineering 54(2), 147–188 (2005)
22. Schewe, K.-D., Wang, Q.: A customised ASM thesis for database transformations (2009) (submitted for publication)
23. Tanaka, Y.: Meme Media and Meme Market Architectures. IEEE Press, Wiley-Interscience, USA (2003)
24. Yara, P., Ramachandran, R., Balasubramanian, G., Muthuswamy, K., Chandrasekar, D.: Global software development with cloud platforms. In: Software Engineering Approaches for Offshore and Outsourced Development, pp. 81–95. Springer, Heidelberg (2009)
25. Zeng, W., Zhao, Y., Ou, K., Song, W.: Research on cloud storage architecture and key technologies. In: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, pp. 1044–1048. ACM, New York (2009)