# Conceptual Modelling of Service-Oriented Software Systems

Klaus-Dieter Schewe*,[a], Karoly Bósa[a], Andreea Buga[a], Sorana Tania Nemeş[a]

[a] Christian-Doppler Laboratory for Client-Centric Cloud Computing, Hagenberg, Austria

Abstract. *Conceptual modelling has originated from the areas of software engineering, databases and knowledge representation, and Heinrich C. Mayr, to whom this article is dedicated, has been involved in this area from the very beginnings. While in these areas a high degree of maturity has been achieved, conceptual modelling still lacks this maturity in other areas such as service-oriented systems despite the demand from novel application areas such as cloud computing. In this article we discuss the axiomatic BDCM$^2$ framework capturing **b**ehaviour, **d**escription, **c**ontracting, **m**onitoring and **m**ediation. We argue that the framework gives an abstract answer to the ontological question what service-oriented systems are. On these grounds we address the intrinsically connected modelling question how to capture cloud-enabled service-oriented systems. We outline a conceptual modelling approach that is grounded in a distributed middleware coordinating the client access to multiple clouds through a concept of mediator. For this we exploit abstract machines with interconnected layers for normal operation, monitoring and adaptation. We illustrate the model by the use case of a robotic care system showing that the general model can be fruitfully exploited for failure alerts, failure anticipation and prevention, and safety hazards detection, which links the research to recent interests of Heinrich in conceptual modelling for ambient assistance systems.*

Keywords. Conceptual Modelling • Service-Oriented System • Ambient Assistance

## 1 Introduction

The field of conceptual modelling has originated from the areas of software engineering, databases and knowledge representation (see e. g. the early collection by Brodie 1984). The emphasis was to create a bridge connecting the informal understanding of stakeholders and the formally precise understanding of system developers. The conceptual model should be a precise blueprint of a system to be built, which at the same time is on a sufficiently high level of abstraction to support mutual comprehensibility. Conceptual modelling was successfully applied to the development of data-centric information systems. The so-called semantic data models (see Hull and King 1987 for a survey) played an important role in this success, in particular the Entity-Relationship model (extended to perfection by Thalheim 2000) was adopted by researchers, consultant, users and developers.

In the further development of the field two main directions of the research can be identified. The first direction is dedicated to novel modelling methods that extend the mature ones in ways that enable different classes of systems to be addressed. Prominent examples are web information systems by Schewe and Thalheim (2018) and business processes by Kossak et al. (2016). The other direction emphasises the application of the methods in specialised application areas such as e-commerce (see e. g. Kaschek et al. 2006) or e-learning (see e. g. Schewe et al. 2005).

Heinrich C. Mayr, to whom this article is dedicated, has been an active researcher in the area of conceptual modelling from the various beginnings.

* Corresponding author.
E-mail. kdschewe@acm.org
The research reported in this paper has been supported by the Christian-Doppler Society in the frame of the Christian-Doppler Laboratory for Client-Centric Cloud Computing.

In particular, he emphasised the generation of skeletons of conceptual database schemata by applying natural language processing. Fliedl et al. (2005) summarise main results of this research. This is also connected to research on the conceptual modelling process, in particular the involvement of stakeholders addressed by Shekhovtsov et al. (2012). While this research stays with the original emphasis of conceptual modelling in the context of the development of data-centric information systems, his research also addressed ontologies in conceptual modelling (see Hesse et al. 2004) and business processes (see Mayr et al. 2007).

His more recent research has been directed towards specific applications. Together with colleagues (see Michael and Mayr 2016) he investigated domain-specific conceptual modelling in general, and in particular the application of conceptual modelling to ambient assistance systems (see among others Al Machot et al. 2014 and Michael and Mayr 2013).

### 1.1 Service-Oriented Systems

Surprisingly, despite the long successful history of conceptual modelling, there is no commonly accepted conceptual model for service-oriented systems, not even an understanding what services are. According to (Bergholtz et al. 2015) the many approaches to service-oriented systems can be roughly classified into those taking a business-centric view and those taking a software-centric view. Examples of the former class are among others the service science framework by Ferrario et al. (2011), the so-called Unified Theory of Services (UTS) by Sampson and Froehle (2006), the approach to semantic web services by Preist (2004) or the OASIS framework by Alves et al. (2007). Examples of the latter class are among others the service-oriented architecture frameworks (SOA) by Arsanjani et al. (2008), Erl (2007) and Papazoglou and van den Heuvel (2006), service-oriented computing (SOC) by Papazoglou and van den Heuvel (2007), web services by Alonso et al. (2003) and Benatallah et al. (2006) and the corresponding W3C standards (Universal Description, Discovery and Integration (UDDI) 2009 and Simple Object Access Protocol (SOAP) 2008), and the behavioural model of Abstract State Services by Ma et al. (2008) and its extension by Ma et al. (2009a). Both lists can be extended by many other examples (see for instance the literature review by Ma et al. 2009a or by Bergholtz et al. 2015).

The research towards the fundamental question "what constitutes a service" has led to the model of Abstract State Services ($AS^2$s) (see Ma et al. 2009a), the model of service mediators based on $AS^2$s (see Ma et al. 2012 and Schewe and Wang 2012), and the service ontology model (see Ma et al. 2009b). The discussion of further properties characterising a software service (on the web) led to the $BDCM^2$ framework capturing **b**ehaviour, **d**escription, **c**ontracting, **m**onitoring and **m**ediation (see Schewe and Wang 2015). In addition, the W*H framework by Dahanayake and Thalheim (2015) puts services into the context of their usage, intention and added value, which are more relevant for the modelling of service-oriented systems than for the clarification of the question what they are.

The $BDCM^2$ framework has been successfully adopted in a conceptual model for multi-cloud interaction by Buga et al. (2017a). The emphasis is on distributed systems exploiting software services from multiple clouds. The services themselves can be modelled as abstract state services specified by Abstract State Machines (ASMs) (see Börger and Stärk 2003 for a general introduction to ASMs). The services can be integrated using the mediator model, by means of which general skeletons are provided that are instantiated by concrete services that are selected according to a service ontology comprising functional, categorical and SLA-based properties. The concrete interaction of a mediator instance with the service providing clouds is subject of a middleware system, developed by Bósa et al. (2015) (see also Bósa 2012 and Bósa et al. 2014), which handles the interaction with the clouds and supports the interaction between different systems through the clouds (see Bósa 2013 for this aspect). For the rigorous specification of this middleware the ambient extension of ASMs by Börger et al. (2012) has been exploited.

The monitoring aspect with respect to security and SLA satisfaction has been addressed by Lampesberger and Rady (2015). Buga et al. (2017b) show how such cloud-enabled distributed systems can be used for modelling a robotic care system showing that the general model can be fruitfully exploited for failure alerts, failure anticipation and prevention, and safety hazards detection. This links the research to recent interests of Heinrich in conceptual modelling for ambient assistance systems.

### 1.2 Organisation of the Article

In Section 2 we address the ontological question: what are services and what are service-oriented systems. Here we refer to the BDCM$^2$ framework and its underpinnings in the behavioural theory of distributed adaptive systems. That is, we emphasise an axiomatic characterisation, though we will abstain from going too much into theoretical depth. Section 3 is then dedicated to the intrinsically connected modelling question: how can service-oriented systems be modelled conceptually. Here our research emphasises Abstract State Machines with various extensions as well as service ontologies as the key building blocks. In view of the many non-technical additions that are needed in design and development of very complex software systems (as discussed by Dahanayake and Thalheim 2015 for services and deeply investigated by Schewe and Thalheim (2018) for web information systems that are largely related) we acknowledge that our presentation here does not cover the complete picture. To illustrate the model we briefly outline the application to the robotic care system use case. Finally, in Section 4 we discuss perspectives for future research in this area, which are an open invitation to Heinrich and his colleagues to remain active and to further contribute to the area of conceptual modelling.

## 2 The Ontological Question: What *Are* Services and Service-Oriented Systems?

The BDCM$^2$ framework addresses the question how to obtain a language independent definition of the notions of service and service-oriented system. We will rephrase the definition in an axiomatic way, but for the sake of brevity we have to leave out many technical details, for which we refer to the literature.

### 2.1 Functional Behaviour

A *behavioural theory* provides first a language-independent clarification of a class of systems by means of a set of axioms, which is complemented in a second step by an abstract machine model, for which it is then proven in a third step that the machine model captures exactly the systems stipulated by the axioms. For the case of services, Ma et al. (2009a) started with the introduction of Abstract State Service (AS$^2$) defining the functional behaviour of a service. Roughly speaking, an AS$^2$ provides a process that can be used by someone else knowing only what the process implementing the service is supposed to do. The user does neither own the service nor is he able to manipulate it.

Each service will provide some form of workflow that will access data resources. Therefore, the AS$^2$ model refers to an underlying database (using this term in a very general sense), which defines an internal layer. For services there must be an additional external layer made out of views, which export the data that can then be used by users or programs. We complete this picture by adding operations on both the conceptual and the external layer, the former one being handled as database transactions, whereas the latter ones provide the means with which users can interact with a database.

**Axiom 1.** A service comprises an internal (database) layer, which 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 axioms of a database transformation over $\mathcal{S}$.

**Axiom 2.** A service comprises an external (view) layer comprising a finite set $\mathcal{V}$ of (extended) views.

- Each view $v \in \mathcal{V}$ is associated with a database transformation such that for each state $S \in \mathcal{S}$ there are views $v_1, \ldots, v_k \in \mathcal{V}$ with finite runs $S_0^j, \ldots, S_{n_j}^j$ of $v_j$ $(j = 1, \ldots, k)$, starting with $S_0^j = S_d$ and terminating with $S_{n_j}^j = S_d \cup V_j$.
- Each view $v \in \mathcal{V}$ is further associated with a finite set $O_v$ 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 $(S, S') \in \tau$.
- 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 $v_i$.

We omit further technical details of the model of abstract state services (for these see Ma et al. 2009a or Schewe and Wang 2015). Both axioms above refer to database transformations, which have been axiomatically defined by Schewe and Wang (2010) with a further sharpening of the underlying behavioural theory by Ferrarotti et al. (2016). Database transformation can be axiomatically characterised by four further axioms, which we will sketch next (for details we have to refer to the lengthy treatment in the literature).

**Axiom 3.** A database transformation comprises a set $\mathcal{S}$ of states, a subset $\mathcal{I} \subseteq \mathcal{S}$ of initial states, and a state transition function $\varsigma \subseteq \mathcal{S} \times \mathcal{S}$ (sequential time).

**Axiom 4.** States of a database transformation are meta-finite states over a signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \Sigma_b$ comprising database functions, algorithmic functions and bridge functions. States are closed are isomorphisms (abstract state).

**Axiom 5.** A database transformation comprises a background structure, which captures at least truth values, constructors for records and finite multisets, and eventually further constructors for the building blocks of the database model as well as operators associated with these constructors for values and terms (background).

**Axiom 6.** There is a finite set $W$ of multiset comprehensions terms built over $\Sigma$ and the background structure such that for any two states $S_1, S_2$ that coincide on them the corresponding update sets $\Delta(S_1)$ and $\Delta(S_2)$ (defined by the state transition function) are equal (bounded exploration).

## 2.2 Description of Services

We may assume that abstract state services as defined above are available through service repositories, for which we adopt the notion of "cloud". In a service-oriented system several such services are combined including third-party services. For the latter ones it has to be known, which functionality is provided, despite the fact that the internal layer (and thus the implementation) is hidden. So the question arises how meaningful services can be located, for which a description of the available services is needed. The key idea is that given a coarse description of the service needed, it should be possible to search for such a service. This is exactly what ontologies are meant to capture, a description of the (semantics of) services. Therefore, we adopt the common idea of a service ontology, which is already omnipresent in the area of the semantic web, also in our own work (see e. g. Ma et al. 2012, 2009b).

This is usually grounded in some more or less expressive description logic, e.g. the web ontology language OWL (see Grau et al. 2008). While it is not possible to precisely describe an abstract state service in a way that it can be automatically matched to a service request, it is commonly accepted that a service ontology should capture three aspects (see e. g. Fensel et al. 2007 and the references in there), which leads to the following axiom:

**Axiom 7.** Each service is equipped with a description in a service ontology, which comprises

- a *functional* description covering the specification of input- and output types as well as

pre- and post-conditions telling in technical terms, what the service will do;

- a *categorical* description capturing interrelated keywords telling what the service operation does by using common terminology of the application area;
- a *quality of service* (QoS) description capturing non-functional properties such as availability, response time, cost, etc.

A functional description alone would be insufficient. For instance, a flight booking service may functionally be almost identical to a train booking system. Therefore, an additional categorical description is indispensable. The terminology of the application domain 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.

The QoS description is not needed for service discovery, but for selection among alternatives. However, the non-functional properties cover also the description of how a service is meant to be used, what are the obligations and rights of the participating agents, at least of the service provider and the service user, how conflicts are to be handled, etc. These non-functional aspects should be considered as being intrinsically part of a service. (Schewe and Wang 2015) discuss them in detail.

## 2.3 Service Mediation

Service mediation addresses collaboration of services, which is another necessary aspect of services. While the functional behaviour of a service is entirely in the hands of a service provider and the service description addresses how a service is offered by the provider to the potential users, service mediation covers how users can make use of a service. For instance, it is commonly known that online sales services are often used as information repositories without the slightest intention to buy something. In other words, while the functionality of a service is defined by the provider and conditions of use are subject of the QoS part of the service ontology, it is exclusively up to the user to exploit the service for his own purposes. Therefore, service mediation is an important aspect of a theory of services.

First, in order to make the workflow within a service explicit we require the notion of *plot*, which actually captures the possible sequencing of service operations. For this Ma et al. (2012) exploit Kleene algebras with tests (KAT), 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 extension of the behavioural model, which in addition impacts on the functional description in the service ontology.

Formally, the service operations give rise to *elementary processes* of the form

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

in which $op$ is the name of a service operation, $\vec{z}$ denotes input for $op$ selected from the view $v$ with $op \in Op_v$, $\vec{y}$ denotes additional input from the user, and $\varphi$ and $\psi$ are first-order formulae denoting pre- and postconditions, respectively. The pre- and postconditions can be omitted; also simple formulae $\chi(\vec{x})$ interpreted as tests checking their validity constitute elementary processes. With this (Ma et al. 2012) define the set of *process expressions* of an $AS^2$ as 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 $\mathcal{P}$. However, this definition is tightly linked to KATs; it can be generalised in a language-independent way emphasising any kind of parallel process (as stipulated by Axioms 3-6) that are composed out of the given elementary processes.

**Axiom 8.** Each service is equipped with a *plot*, which is a process expression over the service operations of the service.

Second, service mediators exploit plots with open slots for services to specify intended service-based applications on a high level of abstraction. The idea is to specify service-oriented applications that involve yet unknown component services. On these grounds matching criteria for services are formally defined that are to fill the slots. A problem in finding such matching criteria is the fact that it should be possible 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 actually needs to be composed are "runs" of services that are determined by the plot including conditions, under which particular service operations can be removed or their order can be changed. This leads to rather complicated matching conditions between services and slots in mediators as emphasised by Ma et al. (2012) and further refined by Schewe and Wang (2012).

In order to capture this idea we relax the definition of a plot in such a way that service operations do not have to come from the same service. Thus, in elementary processes we use prefixes to indicate the corresponding $AS^2$, so we obtain $\varphi(\vec{x})\ X : op[\vec{z}](\vec{y})\ \psi(\vec{x},\vec{y},\vec{z})$, in which $X$ denotes a *service slot*, i. e. a placeholder for an actual service.

**Axiom 9.** A service-oriented system consists of service mediators, which are process expressions with service slots. Each service operation in a mediator is associated with input- and output-types, pre- and postconditions, and a concept in a service terminology. An instance of a mediator is defined by services matching the slots.

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 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. 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. 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 $AS^2$. The order of service operations may differ and certain service operations may become redundant, which has to be taken care of as well. (Ma et al. 2012) discuss matching criteria in detail.

## 2.4 Service Contracts

The aspects of behaviour, description and mediation alone do not yet capture everything that would characterise services. Zeithaml et al. (1985) emphasise general properties such as intangibility, inseparability, heterogeneity and perishability of services, which in the community have been controversially debated and rejected as being neither necessary nor sufficient. Indeed, intangibility refers to the fact that a service is owned by its provider, while a service user can exploit the service for his own purpose, but there is never a transfer of ownership nor does the user even know how the service is realised. For instance, in the often used example of a snow removal service it is up to the provider to decide whether shovels, brooms or snow ploughs are used, which the service client is

not interested in at all, as long as the agreed result is guaranteed. Intangibility is de facto captured by the behavioural model; in particular, in the $AS^2$ model a hidden internal layer is separated from the visible layer exposed to the user, which includes the associated plot. Similarly, perishability is no property of services at all. Every software system is prone to perishability, if the hardware and system software it is grounded in disappears. To the contrary, it should be part of the usage agreement between a service provider and a user that services do not perish within the agreed period of service usage. That is, availability agreements are part of the service as well as the consequences, if the agreement is violated. Inseparability has to be treated also with care. Of course, from the point of view of the provider the service is offered as a whole, and there is an agreement about this with each service user. However, as already discussed in connection with mediation, it is up to the user to exploit the services in his own way and for his own purposes, regardless what the provider intended. In this sense, a user may well cut out of a service the parts that are really needed, even though for the provider it still appears that the service was used as a whole. Finally, heterogeneity is not a characteristic at all, as it could only refer to the collection of all services, in which case it is a triviality. If it were to refer to a single service, there is no reason for the claim that heterogeneity of the components involved should always be the case.

Schewe and Wang (2015) discuss in detail alternative ideas by Bergholtz et al. (2015), in the UTS by Sampson and Froehle (2006), in the REA ontology, and in the classification of rights following Hohfeld. While we cannot repeat this intensive discussion here, we emphasise the conclusion that this would still draw an incomplete picture regarding non-functional aspects, as there are more rules than those capturing rights and obligations. In general, there should be a whole list of service-level agreements, which altogether define a contract between the provider of a service and a service user. Rady (2012) has defined fragments of an ontology capturing SLAs and implemented a tool extracting contracts from such an SLA ontology (see Rady 2014). For the time being the SLA ontology, which in our opinion should be part of the service ontology, only addresses availability and performance aspects, though more than 20 additional types of SLAs have already been discussed in the literature. SLAs concerning availability, performance, etc. but also security and privacy regulations have nothing to do with rights. For instance, availability on one hand concerns conditions of usage, e.g., if the service is only available on workdays within a specified time period, and on the other hand a commitment and obligation by the provider. Security and privacy may be also handled as commitments, but it should preferably also include the means with which security is supposed to be achieved, in which case the SLA includes a statement about the functional behaviour of the service or its environment. In summary, for each service there must exist a service contract capturing all SLAs, and the SLAs may be expressed by obligations and rights in a deontic action logic, refer to functional aspects, or simply cover factual data. The decisive feature, however, is that a model of services must comprise the ***c****ontracting* aspect.

**Axiom 10.** For each service used in a service-oriented system there must exist a contract between service provider and service user covering all relevant SLAs for the service. The conditions in the contract must be consistent with the quality of service description in the service ontology.

## 2.5 Monitoring and Adaptation

(Schewe and Wang 2015) emphasise that a contract that cannot be validated is rather useless, both technically and legally. Therefore, for every service it should be possible to check not only its behaviour, but also whether the SLAs are fulfilled. That is, there must exist a monitoring software for this purpose – this could again be a service, but not necessarily. Lampesberger and Rady (2015) make a promising step in the direction of SLA monitoring in the context of cloud computing.

While monitoring is a decisive aspect of service-oriented systems, the focus on SLA monitoring is too narrow.

Lampesberger (2016) already emphasises anomaly detection as another aspect, for which monitoring is required, in particular, if services exploit public clouds or are in any other way exposed to undesired external interference. This naturally extends to failures and any other critical situation as emphasised by Buga et al. (2017a).

**Axiom 11.** Every instance of a service-oriented system constitutes a distributed adaptive system, in which all transactions and service operations of individual services are reflective, i. e.. they satisfy the axioms for reflective database transformations.

Note that this axiom includes the possibility to provide specialised services for monitoring (as e. g. for anomaly detection) and for adaptation, e. g. replacing services in an instantiation as discussed by Buga et al. (2017a). For the extension, that reflective behaviour has to be supported; we refer to the work by Schewe et al. (2017), who discuss the axioms for distributed adaptive systems in detail.

For our brief exposition here we restrict to mention that only axioms 3-6 need to be slightly amended. In a nutshell, reflection can be captured by storing the specification of the system behaviour, e.g. the signature and ASM rule as part of the system's states. Then terms over the signature $\Sigma$ can also be used as values, and the interpretation of some terms in a state may result in terms. For the bounded exploration axiom we then need a stronger notion for the coincidence of terms that are extended in this way. For strong coincidence we request (as before) that the interpretation of the terms in $W$ is the same for two states, but in addition the interpretation of terms resulting from this first interpretation also yields equal results. Axiom 6 can then be rephrased as follows:

**Axiom 6′.** There is a finite set $W$ of multiset comprehensions terms built over $\Sigma$ and the background structure such that for any two states $S_1$, $S_2$ that strongly coincide on them the corresponding update sets $\Delta(S_1)$ and $\Delta(S_2)$ are equal (reflective bounded exploration).

## 3 The Modelling Question: How to *Capture* Service-Oriented Systems?

Our axiomatic definition of service-oriented systems is language-independent, so for the actual task to model service-oriented systems we require adequate languages by means of which we can obtain models satisfying the axioms. Actually, in behavioural theories we go further asking for the capture of the class of systems of interest, which requires to formally prove that all systems as stipulated by the axioms are faithfully represented by the modelling language. For conceptual modelling the focus is in addition on the capture on a high level of abstraction, by means of which the comprehensibility requirement can be fulfilled.

With respect to the axioms describing functional behaviour, mediation and monitoring the concepts of abstract state services, mediators and reflection together, i. e. all axioms except Axioms 7 and 10, give rise to distributed adaptive systems. In a longer sequence of theoretical research on foundations of such systems a behavioural theory was discovered. In a nutshell, the final result is that distributed adaptive systems are captured by concurrent reflective ASMs. Here we abstain from a presentation (for technical details see Schewe et al. 2017 and the references in there). Instead we concentrate on a cloud-based middleware architecture that enables distributed adaptive service-oriented systems. While the middleware itself is specified and verified using concurrent reflective ASMs, applications can exploit the $AS^2$ model for modelling of services, mediators for creating service-oriented systems, and an associated service ontology.

### 3.1 Cloud-Enabled Service-Oriented Systems

In the previous section we emphasised that services should be taken from a service repository. We deliberately use the term *cloud* to refer to such

a repository of services as emphasised by Ma et al. (2012). From the definition of mediators it is clear that an instantiated mediator is a high-level specification of a distributed application that runs several services at the same time. Refining and implementing such a specification requires several add-ons. 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, for which we employ the *client-cloud interaction middleware* (CCIM) model defined by Bósa et al. (2014).

**Client-Cloud Interaction Middleware**

The CCIM has been specified using ambient ASMs in order to describe formal models of distributed systems incorporating mobile components in two abstraction layers. While the algorithms of executable components are specified in terms of ASMs, their communication topology, locality and mobility are described with the terms of ambient calculus. As each ambient ASM specification can be translated into a pure ASM specification as shown by Börger et al. (2012). The approach provides a universal way to handle client-cloud interaction independent from particularities of certain cloud services or end-devices, while the instantiation by means of particular ambient results in specifications for particular settings. Thus, the architecture is highly flexible with respect to additional end-devices or cloud services, which would just require the definition of a particular ambient. The architecture of the CCIM integrates all novel software solutions such as Service Plot-Based Access Management, Client-to-Client Interaction (CTCI) Feature, Identity and Access Management (IdMM), Content Adaptivity, SLA Management and Security Monitoring Component into a compound single software component.

In the general architecture (see e. g. Buga et al. 2017a) the middleware is replicated by several components, each connected to one or more service clouds, but each cloud is connected to exactly one middleware component. Thus, there are three modes of interaction: (1) interaction of users with a middleware component, (2) interaction of a middleware component with a service in one of its clouds, and (3) interaction among several middleware components. The challenge is to keep users oblivious about the interaction among middleware components to locate individual services and to manage the transfer of results among the participating services. This challenge is addressed by the propagation of service requests among the middleware components. That is, when a middleware component receives a request for access to a particular service from a client or another middleware component, it will route the request to the middleware component owning the service, i. e. being the component that connects to the cloud, on which the service resides. Thus a service requests always comprises also routing information.

In addition to the routing of requests to access individual services each middleware component will exploit the features of the mediator model and analyse how to execute a particular mediator by extracting services that it can handle itself and those parts that have to be forwarded to other components. This is captured by an ambient-ASM specification of the distributed middleware emphasising the normal execution model. The normal execution mode requires the abstract machine, i. e. the ambientASM specification, the service interfaces, the request handler that links to the users and other middleware components, and the communication handler that handles the interaction among middleware components.

The CCIM provides a cloud service infrastructure that permits a transparent and uniform way for clients to interact with multiple clouds. It permits to access and combine the available functions of cloud services, which may belong to various owners, and it leaves the full control over the usage of

their services in the hands of the service owners. If a registered cloud user intends to subscribe to a particular service, a subscription request is sent to the cloud, which may forward it to such a special client corresponding to the service owner. This client responds with the service plot, which defines how the service can be used by the user and determines the permitted combination of service operations.

The received service plots are collected together with other available cloud functions in a personal user area by the cloud. When the subscribed user sends a service request, it is checked whether the requested service operations are permitted by any service plot. If a requested operation is permitted, then it is triggered to perform, otherwise it is blocked as long as a plot may allow to trigger it in the future. Each triggered operation request is authorized to enter into the user area of the corresponding service owner to whom the requested service operation belongs. Here a scheduler mechanism assigns to the request a one-off access to a cloud resource on which an instance of the corresponding service runs. Then the service operation request is forwarded to this resource, where the request is processed by an instance of the service whose operation was requested. Finally, the outcome of the performed operation returns to the area of the initiator user, where the outcome is either stored or send further to a given client device.

In this way, the service owners have direct influence on the service usage of particular users via the provided service plots. If a user subscribes to more than one service, he or she may have access to more than one plot. These plots are independent from each other and they can be applied concurrently. If a service owner makes available more than one service for a user, the owner has the choice either to provide independent plots for the user or to combine some functions of various services into a common service plot. This conceptual solution shows a transparent and uniform way how to provide an advanced access control mechanism for cloud services without giving up the flexibility of heterogeneous cloud access to these services.

Furthermore, due to the ambient concept the relocation of system components is trivial, and the model can be applied to different scenarios. For instance, all our novel methods including our client-cloud interaction solution can be shifted to the client side and wrapped into a middleware software which takes place between the end users and cloud in order to control the interactions of them. Note that the specified communication among the distributed system components remains the same in both scenarios.

**Monitoring and Adaptation**

In addition, the CCIM provides monitoring and assessment layers. For each service there are several dedicated monitors. For the observation of the behaviour of these services sensors are deployed across multiple clouds in order to collect environmental data that are reported to the middleware. The monitoring is part of an abstract machine, which is specified using the ASM method. It is important to consider that monitors are also components of the distributed system, so they can also exhibit failures themselves. This is taken care of by assigning a trustworthiness measure to each monitor. Monitoring components with trustworthiness below a specified threshold are removed from the network of monitors. In case of an identified critical situation the adaptation layer replaces one or several services, i. e. replacing the given mediator instantiation by a new one.

Monitors collect data from the nodes. When starting the system, each monitor is initialized by the middleware in the `Active` state, from where it submits a heartbeat request to the node it monitors. The monitor advances afterwards to the `Wait for response` state, where it checks two guards. First, it verifies if a response to its request is received. If so, it verifies if the delay of the response is acceptable. If this condition holds, the monitor moves to the `Collect data` state. If no response is received or if the response has a big delay, the monitor moves to the `Report problem` state. In the `Collect data` state the

monitor gathers low level metrics (CPU, memory and storage usage, bandwidth) and then moves to the `Retrieve information` state, where it checks local storage for past monitoring data. If the repository is available, the monitor queries it. The monitor moves to the `Assign diagnosis` state, where it interprets the available data. If it discovers a problem, it moves to the `Report problem` state, otherwise it moves to the `Log data` state, where meaningful data and operation are logged. When an issue is identified, the monitor modifies a constraint that triggers a request towards the leader of the node to inquire all its monitoring counterparts and carry out a collaborative diagnosis. After reporting the issue, the monitor moves to the `Log data` state. Here, the confidence degree of the monitor is checked, and if the monitor is still trustworthy, it starts a new monitoring cycle. Alternatively, it moves to the `Inactive` state and waits to be deployed again in the system.

Details of the CCIM model are covered in several articles, e. g. by Bósa et al. (2015), Buga et al. (2017a) and Lampesberger and Rady (2015). We omit further details here.

### 3.2 Service Ontology

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 (for details see Ma et al. 2012 and Schewe and Wang 2015).

**Terminologies**

For simplicity let us assume that $C_0$ and $R_0$ represent not further specified sets of basic concepts and roles, respectively. Then *concepts* $C$ and *roles* $R$ are defined by the following grammar:

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

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*. 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 $\neg \geq m+1.R$.

A *structure* $\mathcal{S}$ for a terminology $\mathcal{T}$ consists of a non-empty set $O$ together with subsets $\mathcal{S}(C_0) \subseteq O$ and $\mathcal{S}(R_0) \subseteq O \times O$ for all basic concepts $R_0$ and basic roles $R_0$, respectively. $O$ 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 $\mathcal{S}(C) \subseteq O$, and for each role $R$ we define a subset $\mathcal{S}(R) \subseteq O \times O$ as follows:

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

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 $\mathcal{T}$.

**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, and pre- and post-conditions. For the types we need a type system with base types and constructors. For instance, the following grammar

$$t = b \mid \mathbb{1} \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t\} \mid [t] \mid (a_1 : t_1) \oplus \cdots \oplus (a_n : t_n)$$

describes (the abstract syntax of) a type system with a trivial type $\mathbb{1}$, 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 $dom(\mathbb{1}) = \{\bot\}$. 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 $(a_1 : \mathbb{1}) \oplus \cdots \oplus (a_n : \mathbb{1})$ 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.

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. For instance, the terminology may comprise the following axioms:

$$\begin{aligned}
&\text{Service_Operation} \sqsubseteq \forall \text{pre.Condition} \sqcap \leq 1.\text{pre} \\
&\qquad \sqcap\ \exists \text{post.Condition} \sqcap \leq 1.\text{post} \\
&\text{Condition} \sqsubseteq \text{Formula} \sqcap \\
&\qquad \forall \text{uses.(Predicate} \sqcap \text{Function)} \\
&\text{Predicate} \sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \neg \geq 1.\text{out} \\
&\text{Function} \sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \\
&\qquad \exists \text{out.Type} \sqcap \leq 1.\text{out}
\end{aligned}$$

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.

**Service-Level Agreements**
SLAs have been widely discussed in the literature. By now around 20 different types of SLAs have been identified by Rady (2012), and depending on the viewpoint these SLAs have been differently classified. Following our discussion in the introduction we consider that the main purpose of SLAs is to determine as precisely as possible the rights and obligations that govern the relationship between a service providers, service users, and, if applicable, third parties. Technically, our approach consists of three parts:

- an extension of the service ontology describing the content of the SLAs,
- a contracting framework that permits a contract skeleton to be extracted from the ontology, and
- a monitoring system that can be used to check, when a violation to an SLA has occurred.

As the service ontology is realised by some description logic, the contracting framework can be realised by queries against the ontology. This has been discussed by Rady (2014) using SPARQL to extract fragments of contracts from an SLA ontology. We dispense with discussing this aspect any further in this chapter. Also, as stated in

the introduction, SLA monitoring is still in an infant state, so we will not discuss it here, but we emphasise again that SLAs that cannot be monitored are de facto useless, i. e.*monitoribility* of SLAs is a necessary property of services.

Different from the classification by Rady we use the following classification schema, which puts a stronger emphasis on rights and obligations:

**Terms of Usage:** Some SLAs simply define general facts about the usage of services. Among these are the pricing schema, conditions for termination and suspension, and the applicable jurisdiction. In particular, these facts do not require to be monitored. They will appear as part of the contract extracted from the ontology and can be used to check bills or determine legal actions in case of inaccuracies.

**Technical Aspects:** Some SLAs refer to technical properties of the services that are determined by the service model, i. e. they are not SLAs in the proper sense. These technical aspects cover two different areas:

***Implementation Aspects:*** For instance, *portability* refers to the property of a service to be moved from one environment to another one, which is a property of the implementation. The same applies to *interoperability*, i. e. the property that the service can be combined with others, *scalability*, i. e. that the service can be applied to various input sizes, and *modifiability*, i. e. the property that the user may tune the service.

Furthermore, properties such as *testability*, *maintainability* and *verifiability* refer to software-technical characteristics. Actually, for a service user it is much more important that a service has been adequately tested and verified, the results of these quality assurance measures are available and reproducible, and preferably the service has already been certified according to some common quality standard rather than obtaining knowledge that verification, validation and testing can be done.

***Usage Aspects:*** This is usually associated with a usability SLA. Terms such as understandability or learnability used in this context are only vaguely defined and thus cannot be used for monitoring purposes. Usability studies can nonetheless give recommendations to service users.

However, some of the technical aspects, in particular the implementation aspects, may also be regarded as defining obligations and commitments of the provider to guarantee particular features of the service, in which case the scope of the commitments made has to become part of the SLA.

**Obligations and Rights of the Provider:** The most relevant class of SLAs covers obligations of the service provider, which also capture what the provider is committed to provide. The most commonly discussed SLAs in this class comprise the following:

***Availability:*** The SLA should cover when and for how long the service is guaranteed to be available to the user. Normally, this is formulated by some form of acceptable down-time. We will discuss availability SLAs further down.

***Performance:*** The SLA defines the expected (maximum / average) response time and throughput. Same as for availability, probability distributions could be used, but this is not state-of-the-art.

***Security and Privacy:*** SLAs concerning security and privacy could define the used methods for authentication, identity management, firewall rules, secrets to be preserved, confidentiality regulations, auditing procedures, etc. In our point of view it appears advisable not only to register the obligations of the provider but also the means to be taken to ensure security and privacy.

***Reliability:*** This refers to the measures taken by the provider to ensure that message content and the service results as a whole are reliable.

***Penalty and Compensation:*** These SLA capture mainly factual data about the amount to be repaid in case an SLA cannot be satisfied.

In addition they may be SLAs capturing rights of the provider, e.g. to close down the service for maintenance or in case of imminent security threats – this could be coupled with alerting obligations – to update the service to a new version or release, to cancel a contract in parts or as a whole, to increase prices, etc.

**Obligations and Rights of the User:** Analogously, SLAs may refer to obligations of the users in particular with respect to usage and security / privacy regulations. These may also be subject to penalty and compensation regulations.

### 3.3 A Perspective for Ambient Assistance

Buga et al. (2017b) discuss requirements for a robotic care system based on the commercially available GrowMu robot(see http://www.growmeup.eu/index.php/home/growmu-robot) with various extensions. The robot is to support elderly people in everyday life activities. While the core of the care system, the robot, acts autonomously and cooperates with its client, a key feature is its connection to a cloud to process information, increase its knowledge, and to share information with other care robots.

On a structural level the care robot is able to move around in the elderly person's household. It provides means for audio-visual interaction comprising a tablet screen for additional input/output of information, a camera to monitor the household and the client in order to detect critical situations (e. g. special requests of the client, reminders or alerts, etc.), a microphone for receiving requests by the client and enabling the discovery of critical situations by analysing sound signals (e. g. a cry, the sound of a falling object, etc.), and speakers to deliver sound messages. Furthermore, the robot is equipped with sensors for temperature and humidity, and with a tray attached to its body for delivering small items (e. g. drugs the client has to take).

Abstracting from the physical machine level comprising among others the electric drive, the sensors and the input/output devices, we consider the core of the care system as a service-oriented system. Thus, on the structural level we can think of services such as Bring($\vec{x}$), Come, Report($t(\vec{x})$), Alert($\vec{x}$), Observe($t(\vec{x})$) and Tacit!, respectively. When issued the robot is to bring the small item $\vec{x}$ to the client, move to the client and wait for further instructions, record the kind of task $t(\vec{x})$, the time of issuing, the issuer, and (if applicable) any rationale for the task, deliver audio-visual information to the client and send the information to the cloud, record information about an activity $t$ (with parameters $\vec{x}$), or stop the transmission of observations to the cloud, respectively (for details see Buga et al. 2017b). All these structural services assume a proper functioning of the robot.

On an operational level the care robot is to learn the elderly person's needs and habits over time and enhance its functionality, which permits compensation for the degradation of the client's capabilities, and supports encouragement for remaining active, independent and socially involved. For this the cloud maintains an anonymised profile of the client comprising routine activities, special care needs, risks that require observation, particular interests, etc. The observations collected by the robot are subject to machine learning mechanisms that enable to learn changes to the profile. All services provided by the care robot depend on such profiles. Profiles are to be maintained by service knowledge bases in the cloud. All robot-cloud interaction will exploit a client-cloud middleware. The middleware is used to support the interaction of a robot with cloud-based services that are used in several ways such as failure alerts, failure compensation, anticipation of failure situations and failure prevention, behavioural pattern detection, safety hazards detection, and enabling of basic security mechanisms.

In order to support optimal care the cloud-based robotic care system is to support multiple interacting care robots. This addresses a learning aspect as well as a collaboration aspect. These are used in the various ways dealing with uncertainty

and privacy and enabling collaboration among robots. For instance, in the case that multiple robots take care of a group of elderly people, e.g. in an elderly home, each robot having dedicated tasks (e.g. being company during a walk, delivering post or medicine, remind a caretaker to drink, etc.), the robots have to interact with each other and decide (by means of consensus algorithms), which robot is responsible for a particular patient. These responsibilities may change over time, for which location-based information will be required from the robot, that is analysed by the monitoring layer to anticipate the upcoming needs of the patient, and the adaptation layer will realise the change of responsibilities when needed. This is extended further to capture failure situations, where other robots will have to step in to replace the functionality of a broken down "colleague".

## 4 A Proposal for Continued Research

We reported on our research on the foundations of conceptual modelling for service-oriented systems. The BDCM$^2$ framework (reported in detail by Schewe and Wang 2015) addresses in an axiomatic way the following important features of services:

***Behaviour*:** There must exist a general behavioural theory of services. For this we outlined our research on the model of Abstract State Services (AS$^2$s) by Ma et al. (2009a), which follows the line of research of the ASM thesis.

***Description*:** There must exist a description of a service that allows it to be discovered and used. For this we stressed services ontologies, e. g. the model by Ma et al. (2009b) addressing functional, categorical and quality aspects of services.

***Contracting*:** There must exist a contract between service provider and user covering all relevant SLAs for the service. Here we outlined that quality aspects of services should be extended to capture also all other aspects that could give rise to SLAs. The collection of SLAs has to be treated as a binding contract between service provider and user.

***Mediation*:** A user must use services in the contracted way, but can build service mediations to realise service-oriented systems satisfying his purposes. For this we rely on the model of service mediators developed by Ma et al. (2012).

***Monitoring and Adaptation*:** It must be possible to monitor the execution of service mediator instance in order to validate its behaviour and contracted SLAs. In case of detected critical situations or SLA violations the system configuration should be changed on-the-fly.

We then sketched the modelling of distributed, adaptive systems that are based on services supported by multiple clouds (for details see Buga et al. 2017a and the references in there). The underlying model for service-oriented systems that exploit cloud-enabled services is the mediator model, and the selection of such services is driven by a service ontology comprising functional, categorical and SLA-based characteristics. The concrete interaction with the multiple clouds is realised by a middleware architecture developed by Bósa et al. (2014). This middleware is extended by monitoring and adaptation layers that identify the need for a change of a mediator instantiation and provide an updated one. All parts of the model have been specified using Abstract State Machines including the extensions covering ambient computing (see Börger et al. 2012), concurrency (see Börger and Schewe 2016) and linguistic reflection (see Schewe et al. 2017). We further illustrated the model by a use case concerning a cloud-based robotic care system providing services for the support of an elderly client (see Buga et al. 2017b for more details).

What is more exciting for an active researcher than an open invitation to contribute to still open problems? The BDCM$^2$ framework tries to cover all aspects of software services and goes much further than related work by Bergholtz et al. (2015), Ferrario et al. (2011), Sampson and Froehle (2006), Preist (2004) or Alves et al. (2007). Nonetheless, there is still no common agreement on the ontological question what services and

service-oriented systems *are*. For instance, the proposal by Dahanayake and Thalheim (2015) contains ideas that are worth being considered as well. It would be great, if aspects that have not yet been covered adequately in the BDCM$^2$ framework (if any) were discovered and the framework fine-tuned. Let us make cloud-enabled distributed adaptive systems a prime theme for conceptual modelling and use ambient assistance as an application, where service-orientation, autonomous systems and domain-specific conceptual modelling come together.

## References

Al Machot F., Mayr H. C., Michael J. (2014) Behavior Modeling and Reasoning for Ambient Support: HCM-L Modeler. In: Ali M., Pan J., Chen S., Horng M. (eds.) Modern Advances in Applied Intelligence (IEA/AIE 2014). Lecture Notes in Computer Science Vol. 8482. Springer, pp. 388–397

Alonso G. et al. (eds.) Web Services: Concepts, Architectures and Applications. Springer-Verlag

Alves A. et al. (2007) Web Services Business Process Execution Language, version 2.0. Last Access: OASIS Standard Committee, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

Arsanjani A., Ghosh S., Allam A., Abdollah T., Ganapathy S., Holley K. (2008) SOMA: A method for developing service-oriented solutions. In: IBM Systems Journal 47(3), pp. 377–396

Benatallah B., Casati F., Toumani F. (2006) Representing, Analysing and Managing Web Service Protocols. In: Data and Knowledge Engineering 58(3), pp. 327–357

Bergholtz M., Andersson B., Johannesson P. (2015) Towards a model of services based on co-creation, abstraction and rights distribution. In: Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications and Web Services. Springer, pp. 29–44

Börger E., Schewe K.-D. (2016) Concurrent Abstract State Machines. In: Acta Informatica 53(5), pp. 469–492

Börger E., Stärk R. (2003) Abstract State Machines. Springer-Verlag, Berlin Heidelberg New York

Börger E., Cisternino A., Gervasi V. (2012) Ambient Abstract State Machines with Applications. In: J. Comp. Syst. Sci. 78(3), pp. 939–959

Bósa K. (2012) Formal Modeling of Mobile Computing Systems Based on Ambient Abstract State Machines. In: Semantics in Data and Knowledge Bases. LNCS Vol. 7693. Springer, pp. 18–49

Bósa K. (2013) An Ambient ASM Model for Client-to-Client Interaction via Cloud Computing. In: Proceedings of the 8th International Conference on Software and Data Technologies (ICSOFT). SciTePress, pp. 459–470

Bósa K., Chelemen R., Vleju M. B. (2015) A Formal Model of Client-Cloud Interaction. In: Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications. Springer, pp. 83–144

Bósa K., Holom R. M., Vleju M. B. (2014) A Formal Model of Client-Cloud Interaction. In: Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications and Web Services. Springer, pp. 83–144

Brodie M. L. (1984) On conceptual modelling – perspectives from artificial intelligence, databases and programming languages. Topics in information systems. Springer

Buga A., Nemeş S. T., Schewe K.-D. (2017a) Conceptual Modelling of Autonomous Multi-cloud Interaction with Reflective Semantics. In: Mayr H. C., Guizzardi G., Ma H., Pastor O. (eds.) Conceptual Modeling - 36th International Conference (ER 2017). Lecture Notes in Computer Science Vol. 10650. Springer, pp. 120–133

Buga A., Nemeş S. T., Schewe K.-D. (2017b) Towards Care Systems Using Model-Driven Adaptation and Monitoring of Autonomous Multi-clouds. In: de Cesare S., Frank U. (eds.) Advances in Conceptual Modeling (ER 2017 Workshops). Lecture Notes in Computer Science Vol. 10651. Springer, pp. 26–35

Dahanayake A., Thalheim B. (2015) W*H: The Conceptual Model of Services. In: Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications. Springer, pp. 145–176

Erl T. (2007) SOA: Principles of Service Design. Prentice Hall Press, Upper Saddle River, NJ, USA

Fensel D. et al. (2007) Enabling Semantic Web Services. Springer-Verlag

Ferrario R., Guarino N., Fernández-Barrera M. (2011) Towards an Ontological Foundation for Services Science: The Legal Perspective. In: Sartor G., Casanovas P., Biasiotti M., Fernández-Barrera M. (eds.) Approaches to Legal Ontologies. Law, Governance and Technology Vol. 1. Springer, Netherlands, pp. 235–258

Ferrarotti F., Schewe K.-D., Tec L., Wang Q. (2016) A New Thesis Concerning Synchronised Parallel Computing – Simplified Parallel ASM Thesis. In: Theoretical Computer Science 649, pp. 25–53

Fliedl G., Kop C., Mayr H. C. (2005) From textual scenarios to a conceptual schema. In: Data Knowl. Eng. 55(1), pp. 20–37

Grau B. C., Horrocks I., Motik B., Parsia B., Patel-Schneider P. F., Sattler U. (2008) OWL 2: The next step for OWL. In: Journal of Web Semantics 6(4), pp. 309–322

Hesse W. et al. (2004) Ontologien in der und für die Softwaretechnik. In: Rumpe B., Hesse W. (eds.) Modellierung 2004. LNI Vol. 45. GI, pp. 269–270

Hull R., King R. (1987) Semantic Database Modeling: Survey, Applications, and Research Issues. In: ACM Comput. Surv. 19(3), pp. 201–260

Kaschek R. et al. (2006) Information systems design: through adaptivity to ubiquity. In: Inf. Syst. E-Business Management 4(2), pp. 137–158

Kossak F. et al. (2016) Hagenberg Business Process Modelling Method. Springer

Lampesberger H., Rady M. (2015) Monitoring of Client-Cloud Interaction. In: Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications. Springer, pp. 177–228

Lampesberger H. (2016) Technologies for Web and cloud service interaction: a survey. In: Service Oriented Computing and Applications 10(2), pp. 71–110

Ma H., Schewe K.-D., Thalheim B., Wang Q. (2008) Abstract State Services. In: Song I.-Y. et al. (eds.) Advances in Conceptual Modeling – Challenges and Opportunities, ER 2008 Workshops. LNCS Vol. 5232. Springer-Verlag, pp. 406–415

Ma H., Schewe K.-D., Thalheim B., Wang Q. (2009a) A Theory of Data-Intensive Software Services. In: Service Oriented Computing and Its Applications 3(4), pp. 263–283

Ma H., Schewe K.-D., Thalheim B., Wang Q. (2012) A Formal Model for the Interoperability of Service Clouds. In: Service Oriented Computing and Its Applications 6(3), pp. 189–205

Ma H., Schewe K.-D., Wang Q. (2009b) An Abstract Model for Service Provision, Search and Composition. In: Kirchberg M. et al. (eds.) Services Computing Conference - APSCC 2009. IEEE Asia Pacific, pp. 95–102

Mayr H. C. et al. (2007) Business Process Modeling and Requirements Modeling. In: First International Conference on the Digital Society (ICDS 2007), p. 8

Michael J., Mayr H. C. (2013) Conceptual Modeling for Ambient Assistance. In: Ng W., Storey V. C., Trujillo J. (eds.) Conceptual Modeling - 32th International Conference (ER 2013). Lecture Notes in Computer Science Vol. 8217. Springer, pp. 403–413

Michael J., Mayr H. C. (2016) The Process of Creating a Domain Specific Modelling Method (Extended Abstract). In: Mendling J., Rinderle-Ma S. (eds.) Proceedings of the 7th International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA 2016). CEUR

Workshop Proceedings Vol. 1701. CEUR-WS.org, pp. 40–43

Universal Description, Discovery and Integration (UDDI). Last Access: http://www.uddi.org

Papazoglou M. P., van den Heuvel W.-J. (2006) Service-oriented design and development methodology. In: International Journal of Web Engineering and Technology 2(4), pp. 4120–442

Papazoglou M. P., van den Heuvel W.-J. (2007) Service Oriented Architectures: Approaches, Technologies and Research Issues. In: VLDB Journal 16(3), pp. 389–415

Preist C. (2004) A Conceptual Architecture for Semantic Web Services. In: McIlraith S. A., Plexousakis D., van Harmelen F. (eds.) The Semantic Web – ISWC 2004. Lecture Notes in Computer Science Vol. 3298. Springer, Berlin Heidelberg, pp. 395–409

Rady M. (2012) Parameters for Service Level Agreements Generation in Cloud Computing: A Client-Centric Vision. In: Castano S. et al. (eds.) Advances in Conceptual Modeling – ER 2012 Workshops. Lecture Notes in Computer Science Vol. 7518. Springer, Berlin Heidelberg, pp. 13–22

Rady M. (2014) Generating An Excerpt Of A Service Level Agreement From A Formal Definition of Non-Functional Aspects using OWL. In: Journal of Universal Computer Science 20(3), pp. 366–384 https://doi.org/10.3217/jucs-020-03-0366

Sampson S. E., Froehle C. M. (2006) Foundations and Implications of a Proposed Unified Services Theory. In: Production and Operations Management 15(2), pp. 329–343

Schewe K.-D. et al. (2005) A Conceptual View of Web-Based E-Learning Systems. In: EAIT 10(1-2), pp. 83–110

Schewe K.-D., Ferrarotti F., Tec L., Wang Q., An W. (2017) Evolving Concurrent Systems – Behavioural Theory and Logic. In: Proceedings of the Australasian Computer Science Week (ACSW 2017). ACM, Deakin University, Victoria, Australia, 77:1-77-10

Schewe K.-D., Thalheim B. (2018) Design and Development of Web Information Systems. (to appear). Springer

Schewe K.-D., Wang Q. (2010) A Customised ASM Thesis for Database Transformations. In: Acta Cybernetica 19(4), pp. 765–805

Schewe K.-D., Wang Q. (2012) Preferential Refinements of Abstract State Machines for Service Mediators. In: Muccini H., Tang A. (eds.) Proc. QSIC 2012. IEEE CPS, pp. 158–166

Schewe K.-D., Wang Q. (2015) What Constitutes a Service on the Web? In: Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications and Web Services. Springer, pp. 257–292

Shekhovtsov V. A., Mayr H. C., Kop C. (2012) Towards Conceptualizing Quality-Related Stakeholder Interactions in Software Development. In: Mayr H. C., Kop C., Liddle S. W., Ginige A. (eds.) Information Systems: Methods, Models, and Applications (UNISCON 2012). Lecture Notes in Business Information Processing Vol. 137. Springer, pp. 73–86

Simple Object Access Protocol (SOAP). Last Access: http://www.w3c.org/TR/soap

Thalheim B. (2000) Entity-Relationship Modeling: Foundations of Database Technology. Springer-Verlag

Thalheim B., Schewe K.-D., Prinz A., Buchberger B. (eds.) Correct Software in Web Applications and Web Services. Springer

Zeithaml V. A., Parasuraman A., Berry L. L. (1985) Problems and Strategies in Services Marketing. In: Journal of Marketing 49(2), pp. 33–46