Closing the User-Centric Service Coordination Cycle

Hans Weigand[1], Paul Johannesson[2], Birger Andersson[2], Maria Bergholtz[2], Jeewanie Jayasinghe Arachchige[1]

[1] Tilburg University, P.O.Box 90153,
5000 LE Tilburg, The Netherlands
H.Weigand@uvt.nl, J.JayasingheArachchig@uvt.nl
[2] Royal Institute of Technology
Department of Computer and Systems Sciences, Sweden
pajo,ba,maria@dsv.su.se

In the future vision of an Internet of Services, users take an active role in service selection and composition. In this context, web services are mostly interfaces to real services and can be classified as coordination services with respect to the latter. To enable users to perform service composition, the effect of the coordination services must be described in such a way that users are not only able to discover services but also to detect and prevent possible conflicts in their composition. To meet these requirements, a service description language for coordination services is proposed based on the REA business ontology.

Keywords: Internet of Services, service design, REA , service description

1. Introduction

In spite of considerable progress that has been made in the area of Service Oriented Computing, the impact on society has still been limited. There is not yet such a thing as an Internet of Services that would allow users to integrate the services they want to use easily and seamlessly. It has been acknowledged that users must play a more active role in service composition, if only because of the long tail of specific and heterogeneous services around [1] that simply cannot be handled all by the IT departments. Enterprise mashups may provide an instrument to realize this service co-creation effort of users and developers [7]. In this paradigm, software resources such as (REST or SOAP) *web services* are embedded in *widgets* that provide simple user interaction mechanisms to these resources; these (visual) widgets are combined by the user himself to create *mashups*.

However, users are not interested in composing web services as such. To them, these are merely interfaces to "real" services such as traveling, meeting support, child care, entertainment or car maintenance. Users have a need to plan and coordinate the services they use (cf. [2]).

Fig. 1 depicts the envisioned user-centric service coordination cycle: users compose mashups and interact with the widgets in them to access web services. The web service typically supports the coordination with a service provider who offers a

real-world service as part of a service bundle. The service affects a resource that concerns the user (the resource could be the user himself, for instance in the case of a hotel reservation). That web services themselves may be composite software entities is left out of this figure as being less relevant to the user, but is of course relevant to the software developer.

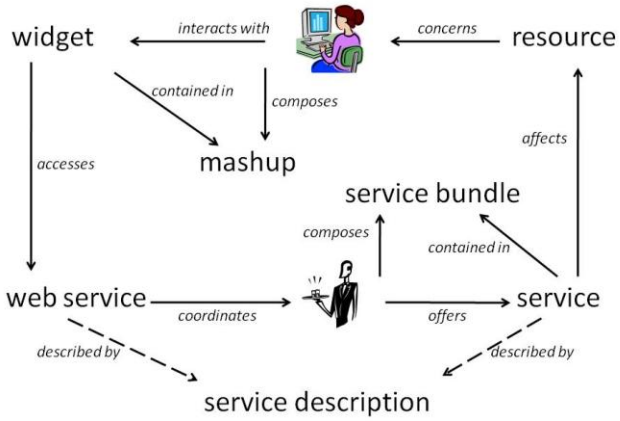

Fig. 1 User-centric service coordination cycle

Both web services and services need a description, but what should be in this description? In composing web services, a major challenge is to reconcile incompatible data representations. In composing services in the real world, a major challenge is to meet the constraints imposed by the fact that resources are scarce, can only be in one place at a time and often cannot be shared. For that reason, [13] argues convincingly that "asset-driven" service modeling will be a central concern in developing an Internet of Services and claims that "novel methodologies and tools are needed to support the modeling of the key assets of services". In our view, this modeling should support at least conflict prevention and conflict detection.

Let *s* be a service that a user U intends to consume and let M be the set of resources and actors involved in the execution of *s*. Each *m* in M has a time-based context A(E,C) where E is a set of events planned for *m* and C a set of constraints on E. The goal of *conflict prevention* is to ensure that when *s* is added to the planning of U, all context constraints are still met, for all *m* in M. Typical events that stem from the planning of *s* are the start of the service execution and its ending. The goal of *conflict detection* is to check context constraints when an event *e* is added. Typical events are contingencies such as a flight being delayed. We can assume that in a future Internet of Services and Internet of Things, most of these events are generated without active user involvement. If *s* is a composite service, then the check should be done on all the services involved individually and jointly.

In order to make conflict prevention and conflict detection possible at all, web services must provide more information than input and output requirements such as we find in a WSDL document. What we need is a generic language to describe services, the resources they use as well as planned and actual events on the type level.

Web services can use this language to represent the preconditions and effects of the real services they connect to as well as their own semantics. A mashup environment can collect and combine this information, integrate it with other sources such as the user's agenda (that should be represented in the same format) in order to provide the user with the conflict prevention and conflict detection functionality described above. On the basis of the service description and after instantiating the formulae with actual data, the user immediately knows the effect of a successful service invocation.

In this paper, we propose to ground the service description language in the REA ontology [9] where we concentrate on coordination services as being of most interest to the user. An advantage of REA is that it has a very small set of basic concepts, and therefore is relatively easy to understand.

To arrive at rigorous and relevant research results, we use Peffers' design science phases [12]. The *problem identification and motivation* has been stated. Our *solution objective* is to develop a coordination service description language based on REA (without addressing a particular syntactic style, e.g. OCL or OWL). In section 2, we work out how REA represents services and the coordination of services. On the basis of that we show in section 3 how service descriptions can be developed that enable the required conflict detection (*design and development*). This is applied to the well-known hotel reservation case (*demonstration*).

2. Coordination Services in REA

2.1 REA and Capacity Planning

The Resource-Event-Agent (REA) ontology was first formulated in [9] and has been developed further, e.g. in [14,4,8]. The following is a short overview of the core concepts of the REA ontology based on [16].

A *resource* is any object that is under the control of an agent and regarded as valuable by some agent. This includes goods and services. The value can be monetary or of an intangible nature, such as status, health state, and security. Resources are modified or exchanged in processes. A *conversion process* uses some input resources to produce new or modify existing resources, like in manufacturing. An *exchange process* occurs as two agents exchange (provide, receive) resources. To acquire a resource an agent has to give up some other resource. An *agent* is an individual or organization capable of having control over economic resources, and transferring or receiving the control to or from other agents [5]. Agents participate in events from inside (the primary perspective of the model) or outside.

The constituents of processes are called *economic events*. An economic event is carried out by an agent and affects a resource. The notion of stockflow is used to specify in what way an economic event affects a resource. REA identifies five stockflows: produce, use, consume, give and take, where the first three occur in conversion processes and the latter two in exchange processes. REA recognizes two kinds of duality between events: conversion duality and exchange duality.

Events can be assigned to a *location*. Sometimes the acronym REAL is used for REA plus location [11].

Using the REA model, we can define the notions of capacity and availability. We take the perspective of the resource manager *a* (e.g. hotel manager) who has received or reserved certain resources from another agent *x* (e.g. hotel owner). He can commit resources of a certain resource type to another agent *x* for a certain date. In that case, there is a specify relationship between the reservation and the resource type. The commitment/reservation has a cardinality indicating the number of resources reserved. The actual allocation of resources (instances) to a certain reservation is usually done later. If we assume the Capacity is stable over time, the following definitions suffice:

Capacity(a,t) = card(R)
R= {r: resource | typify(r,t) ∧ (∃x:agent received(a,x,r) ∨
∃s:reservation (give(x,s) ∧ take(a,s) ∧ specify(s,t) ∧ reserve(s,r)) }
Reserved(a,t,d) = ∑ c: card(s,c), s∈RS(a,t,d) where
RS(a,t,d) = {s: reservation | ∃x,a:agent give(a,s) ∧ take(x,s) ∧ specify(s,t)
∧ date(s,d) }
Available(a,t,d) = Capacity(a,t) - Reserved(a,t,d)

The capacity for a resource type *t* is what the agent has received or that is made available to him (and that is of the resource type *t*. To calculate the availability at some date/time *d*, we first sum up the commitments, and detract this number from the capacity.

2.2 Coordination services

Coordination services are defined in [16] as services supporting an exchange process (a set of events) for a good or a service. Processes like identification, negotiation, order execution and after-sales take place in both cases. We introduce the notion of coordination *object* for the object of these processes: what *is* negotiated and executed? The central coordination object is the purchase order fulfilled by the exchange event, but in complex processes there are many more. The following two reoccur quite often, especially when services are concerned: appointment and reservation. The reason for that is simply that the delivery of a service requiring resources from both the provider and customer to be present at the same time and place requires more coordination than the delivery of a good.

Using REA coordination objects can be specified in terms of commitments. Therefore, another way of characterizing coordination services is to say that these services manipulate commitments: their goal is to give, take and fulfill commitments.

We assume that for all coordination objects there is an agreement process first followed by an execution and evaluation process, that is, the coordination process per coordination object takes the form of a "Conversation for Action" [3,6]. The message exchange in these conversations is not in the scope of this paper, but what is important is the effect of these conversations, since that is directly relevant for a user composing and using a certain mashup application.

In standard REA, a reservation is a relationship between a commitment and a resource. In the following, we use the term "reservation" more specifically for a commitment that precedes the purchase order, which obliges a provider not to sell a

resource to any other agent than the customer for whom the reservation is created. From an economic point of view, the main objective of this kind of reservations is to reduce uncertainty about the business transaction – to mitigate the risks involved, such as items being out of stock or functionality not available, and to reduce the need for slack [15]. So although the reservation has some costs in the form of less operational discretion, it increases the total value for both customer and provider.

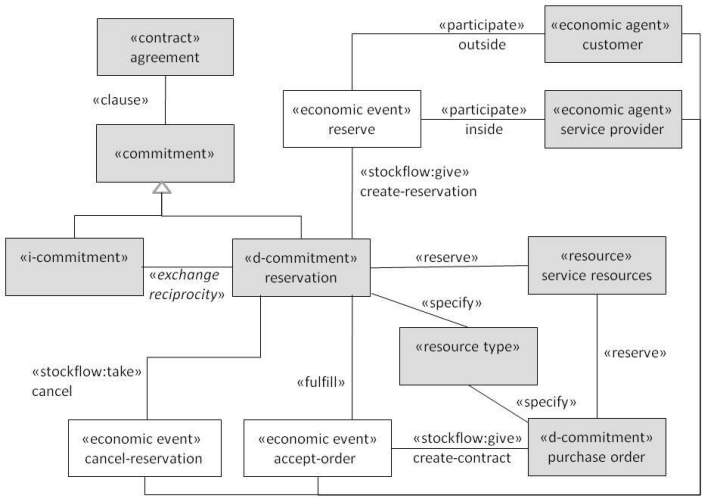

Fig. 2 REA Application Model for reservations

The REA application model in Fig. 2 contains and relates two coordination objects: reservation and purchase order. The reservation is commitment that specifies a resource type and there is a "reserve" relationship with resource, being all resources involved in the fulfillment of the commitment and set apart for that purpose. Quite often, the commitment specifies a resource type only and the allocation of the specific resource is done later. According to REA, there is exchange reciprocity between commitments. This reciprocity leads to dependencies between commitments that must be managed properly by the coordination services. The contract can be explicit or implicit. It may contain additional commitments, usually conditional ones (terms), such as a penalty for non show-up. In line with [8] we distinguish between d-commitments (decrement) and i-commitments (increment), for commitments by or to the service provider, respectively. The fulfill relationship is one between commitment and economic event. The fulfillment of the reservation is the accept-order event by which the purchase order is created. The fulfillment of the purchase order is the service exchange event. Since this could be seen as the ultimate objective of the reservation as well, we define a fulfill* relationship being the transitive closure of fulfill-relationships.

3. Service Description and Conflict Detection

3.1 Service Description Using REA

Using the REA ontology, service descriptions can be developed for coordination services either in the form of REA events REA relations. Table 1 specifies the basic predicates.

Table 1. Basic REA predicates

RELATIONS	EVENTS	TERMS
At(Agent,Location)	*Commit(Id,Agent,Agent, e(Resource Type,Time))*	*contract*
Fulfil(Event,Commitment)	*Cancel(Id,Commitment)*	*commitment*
Clause(Commitment, Contract)	*Purchase(Id,Agent,Agent, Resource)*	
Available(Agent, ResourceType,Time): Number	*Pay(Id,Agent,Agent,Money)*	
Capacity(Agent , Resource Type):Number	*Move(Id,Agent,Location)*	
PlannedCapacity(Agent, Resource Type, Time):Number	*Move(Id,Agent,Resource, Location)*	

The relations and terms have a direct counterpart in REA or have been defined in section 2. We use some shorthands for the events. *Commit* stands for create commitment, *Cancel* for withdraw commitment. *Purchase* and *Pay* stand for the standard exchange events. *Move* stands for the event of changing the location of the agent or some resource. In both Commit and commitment we make use of an embedded functor *e(x,t)* where *e* is an Event Type, *x* can be a any object (and there may be more than one argument *x*) and *t* is a time reference. Expressions of this form are called *i-events* and are used in the same way as actions in the situation calculus [10], where they can be the object of a *do*-action.

Using these predicates, we define the following list of coordination services (table 2). Note that they are services in terms of [16]: their goal is an event that affects a relevant resource. Being coordination services, they manipulate commitments. Table 2 presents the IOPEs (Input/Output/Precondition/Effect) for hotel services but in a quite general way. As such it can be applied to a flight service or theater service as well. However, the way the coordination services are bundled in web services may differ. In the typical hotel case, the Create_Contract and Check_In are one transaction: at the moment the customer shows up, according to his reservation, a contract is set up and a specific resource is allocated. In the typical flight case, the Create_Contract is performed long time before the Check_In.

3.2 Conflict Detection

As said in section 1, each resource or agent is assumed to have a time-based context A(E,C) where E is a set of planned events and C a set of constraints on E. To support

conflict detection and conflict prevention, we should be able to check whether E meets the constraints C.

Let *M* be the set of resources relevant to *U*. To determine the contents of M, the set E_u of committed events for *U* is calculated first as follows:

$E_u = \{e: \text{event} \mid \exists c:\ \text{commitment}(c) \ \wedge \text{fulfill}^*(e,c) \wedge \text{participate}(e,U))\}$

Then

$M = \{m \mid \exists e \in E_u: \text{stockflow}(e,m) \vee \text{participate}(e,m)\}$ (resources and agents involved, as far as known)

For some $m \in M$, the context E_m contains the committed events that involve *m*. Note that $U \in M$. However, not only the context of U, but the context of every $m \in M$ should not violate its constraints. The constraints in the context can be resource-specific, but a very fundamental constraint is that there can be no "agenda conflict":

Table 2. Generic coordination services

Coordination Service	Input	Output	Precondition	Effect (Goal)
Check_Availability	ResrcType R Time T User U	Bool A	A= (Available(Self,R,T)>0)	*Not a social fact*
Create_Reservation	Customer C Time T ResrcType R	Id Res	Available(Self,R,T)>0 At(Self,L)	commit(i,Self,C, e(R,T)) and i=Res and commit(j,C,Self, move(C,L,T.start))
Cancel_Reservation	Customer C Time T ResrcType R Id Res	-	commitment(i, Self,C, e(R,T)) and i=Res and not exist p: fulfill(p,i)	cancel(j,i) and forall j: commitment(j,C,Self, move(C,L,T.start)) implies cancel(j)
Create_Contract	Customer C Time T Id Res	Id PO Amount F	commitment(i,Self,C, e(R,T)) and i=Res	commit(j,Self,C,e(Rs,T)) and j=PO and typify(Rs,R) and exist contract(CT) and clause(PO,CT) and clause(Inv,CT) and commitment(Inv,C,Self, pay(F,T2)) and fulfill(PO.Res)
Check_In	Customer C Time T Id PO	Id Ri	commitment(j,C, Self, move(C,L,T.start) and j= LRes and at(C,L) and commitment(i, Self,C, e(Rs,T)) and i=PO	commit(i,Self,C,e(Ri,T)) and realize(Rs,Ri) and forall m: move(m,C,L) implies fulfill(m,LRes)
Check_Out	Customer C Id Ri	Id S	commitment(i,Self,C, e(Rs,T)) and i=PO and realize(Rs,Ri)	purchase(j,Self,C,Ri,T) and i=S and fulfill(S,PO)
Receive_Payment	Customer C Id PO	Id P	exist contract (CT) and clause(PO,C) and clause(Inv,C) and commitment(Inv,C,Self, pay(F,T2))	pay(j, C,Self, F) and j=P and fulfill(P,Inv)
Cancel_Contract	Customer C Time T Resource Rs Id PO	-	commitment(i, Self,C, e(Rs,T)) and i=PO and exist contract(C) and clause(PO,C)	cancel(j,i) and forall j: commitment(j,C,Self, Q,T') implies cancel(j)

$\forall e_1, e_2 \in E_m$ $e_1.time \cap e_2.time = \varnothing$

Another general constraint is that the resource can be at only one place at a time, and needs time for moving:

$\forall$ $e_1, e_2 \in E_m$: $e_1.time.end = e_2.time.start \Rightarrow e_1.location = e_2.location$

$\forall$ $e_1, e_2 \in E_m$: $next(e_1, e_2) \wedge e_1.location <> e_2.location$

$\Rightarrow (\exists e_i \in E_m : e_1 < e_i < e_2 \wedge e_i.type =$ «move» $\wedge$ $e_1.object = m$

$\wedge e_i.destination = e_2.location)$

where $next(e_1, e_2)$ means that e_2 is the first event after e_1.

To prevent conflicts when considering the use of a service *s*, the user first adds the commitments produced by *s* to his context (using the coordination service effect descriptions), and then executes the conflict detection process.

References

1. Anderson, C. The Long Tail: How endless choice is creating unlimited demand. Random House Business Book, London (2006).
2. Benatallah, B., Casati, F., and Toumani, F. Web Service Conversation Modeling: A Cornerstone for E-Business Automation. IEEE Internet Computing 8, 1(2004).
3. Dietz. J. Enterprise Ontology - Theory and Methodology. Springer, Berlin (2006).
4. Geerts, G., McCarthy, W.E. An Accounting Object Infrastructure For Knowledge-Based Enterprise Models. IEEE Int. Systems & Their Applications, pp. 89-94, (1999).
5. Gailly, F., Laurier, W., Poels, G. Positioning and Formalizing the REA enterprise ontology. Journal of Information Systems 22, 219-248 (2008).
6. Goldkuhl, G. Action and media in interorganizational interaction. Comm.. ACM 49, 5 pp.53-57 (2006).
7. Hoyer, V., Stanoevska-Slabeva, K. Towards a reference model for grassroots enterprise mashup environments. Proc. ECIS 2009 (2009).
8. Hruby, P. Model-Driven Design of Software Applications with Business Patterns. Springer Verlag (2006).
9. McCarthy W.E., The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review (1982).
10. McCarthy, J., Hayes, P.J. Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence, 4:463–502 (1969).
11. O'Leary, D., REAL-D: A Schema for Data Warehouses, Journal of Information Systems, Volume 13, Number 1, pp. 49-62 (1999).
12. Peffers, K., Tuunanen, T., Rothenberger, M., and Chatterjee, S. A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems, 24(3), 45-77 (2008)
13. Pistore, M., Traverso, P., Paolucci M., Wagner, M. From Software Services to a Future Internet of Services. In: G. Tselentis et al (eds), Towards the Future Internet. IOS Press, (2009).
14. UN/CEFACT Modelling Methodology (UMM) User Guide. Available at http://www.unece.org/cefact/umm/UMM_userguide_220606.pdf (2003)
15. Weigand, H., Heuvel, W.J.A.M. van den. A conceptual architecture for pragmatic web services. In M. Schoop, A. de Moor, & J. Dietz (Eds.), Proc. 1st Int. Conf. on the Pragmatic Web (pp. 53-66). Heidelberg: Springer-Verlag (2006).
16. Weigand H., Johannesson, P., Andersson, B., Bergholtz Value-based Service Modeling and Design: Toward a Unified View of Services. Proc. CAiSE'09, Springer, pp.410-424 (2009).