8th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
Suceava, Romania, May 25 – 27, 2006

ACTING AGENTS FOR SOC

Ioan Alfred LETIA[1], Anca CHIORAN[2]
Technical University of Cluj-Napoca
Department of Computer Science
Baritiu 28, RO-3400 Cluj-Napoca, Romania
{letia, anca}@cs-gw.utcluj.ro

Abstract. *We present a SOA based on MAS principle. Modelling service composition from both choreography and orchestration point of view has some drawbacks in the context of a dynamic and open environment. Global composition requires all the information for composition to be available and understandable. Existence of ontology is a key requirement for interoperability, so we propose a SOC approach based on ontology of activities. Each simple service is described in terms of this ontology and the composite service too. But the synthesis process of composite service is moved from a central entity to each agent involved in SOA. Therefore, agents work with local views of the composite service and interact to decide which services and in which order have to be enacted for the composite one.*
Keywords: *web services, agent collaboration, process specification language, situation calculus*

1. Introduction

Web services, interoperation and integration are for quite some time the major subjects of modeling and software engineering as a response to the requirements of the business world. The benefits of using web services are far from being complete in the absence of their composition. Although not brand new, with principles taken from MAS, SOAs (Service Oriented Architectures) address many of the service integration issues. Service-Oriented Computing (SOC) is the emerging paradigm for distributed computing and e-business processing that utilizes services as fundamental elements to enable building agile networks of collaborating business applications distributed within and across organizational boundaries [10].
The ability to build on conventional information technology in a standardized manner requires strong conceptualizations of the knowledge domains, in other words semantic descriptions. SOA, that supports reasoning for discovery, composition, invocation, integration and knowledge management is not the only relevant topic for semantic enabled software engineering. Model driven architecture addresses the reasoning for consistency checking and transformation; Autonomous Computing emphasizes the reasoning for self-management of software systems; Requirements Engineering use ontologies as more expressive domain models. Together they offer principles for building new systems from the existing one.
The composition aspects discussed in this work are inspired from the relation between services and agents. We propose a SOA based on agents and a domain ontology in order to address some challenges of SOC like dynamic selection, fail recovery and utility computing in the context of service integration. The services are modeled as processes, as in many other works. The central issue of our proposal is the activity ontology that captures behavioral aspects of services domain. This ontology underlies the collaboration of agents associated to services in their attempt to solve a query that implies one ore more web services.
The present paper is organized as follows: Section 2 discusses some models for web services and introduces the PSL language and situation calculus; Section 3 introduces the ontology, methods for reasoning on it and simple web service description. The agent's collaboration for synthesis of composite service is highlighted in Section 4 and Section 5 draws some conclusions.

2. Web service model

Business processes are used for both orchestration and choreography of services. The

orchestration analyse the interaction from the service perspective, and the choreography from a global one. But even when we look at the processes from the choreography perspective, the services are not autonomous in deciding the collaborations; these decisions are taken by the global business process.

Drawbacks of the business processes are the static description of them. That processes exist apriori, they are not built dynamically from some rules; in this context, their capability to adjust to the changing environment is limited. In order to allow free interaction of services, methods from agent planning and reasoning should be applied for services or the join between the two paradigms - web services and agents - should be used.

The WSDL description offers only descriptive information about the way a service can be invoked. But no information about when to invoke that service is contained by this description. The functionality of the service and its flow must be part of the service interface.

There are two important issues: the first is how a service must describe what it can do, and the second one is how the other services can find out and understand this. For the first one, describing a service as a business process seems to be a suitable approach. For the other one, there must be protocols for complex conversation between services in order to know each other behaviours and to work in a collaborative manner for achieving a goal.

There are more approaches for these issues. In [8]it is suggested a combination and extension of two web services languages, WS-Conversation Language(WSCL) and WS-Agreement in order to overcome the simple request/response communication between services. The paper proposes extending the languages with speech-acts that allow many complex types of interaction.

The Roman model [2], [3] includes transitions based not only on internal activities but also on message exchange [5]. The e-service model includes an exported behaviour for each service represented as a set of (possibly) infinite sequence of actions that the e-service participates in.

Representing services as Mealy machines has a large acceptance in the composition area. The Conversational Model from [5] includes the idea of exchange sequence of messages of given types according to a predefined set of channels.

Composition in the context of service communities is improved for efficiency in [9]. The paper uses the Roman model and extends it to integrate costs into the "ad hoc" delegation computation.

All of them have in common the concept of behavioural description [9] in a business process. One of the advantages of using business processes and actions is the state concept introduced by them. In the real life we usually meet long term transaction. Process state support these long transaction, unlike the web services that do not have state information [10].

In our model, web services are described by simple or complex processes. These processes are sequences of activities and subactivities. The sequence is determined by the constraints on activities. These constraints can be owned by a service - expressing the local policy of the service provider - or can be general constraints, known by all entities involved - the activity ontology.

PSL and Situation calculus

We chose to work with these two theories due to the similarities between them and their capabilities for process description. There are already standards based on PSL principles for describing services [1]. Occurrence tree of PSL are isomorphic to substructures of the situation tree from situation calculus, the primary difference being that rather than a unique initial situation, each occurrence tree has a unique initial activity occurrence. As in the situation calculus, the poss relation is introduced to allow the statement of constraints on activity occurrences within the occurrence tree. Since the occurrence trees include sequences that modellers of a domain will consider impossible, the poss relation "prunes" away branches from the occurrences tree that correspond to such impossible activity occurrences.

3. Activity ontology

Although PSL defines a neutral representation for manufacturing processes, it provides for describing the semantic domain of web service's behaviours. Behaviour specification is clearer and unambiguous using the PSL constructs [4]. Besides describing activities and subactivities of the domain, PSL axioms establish ordering and existence constraints for the sequences of the behaviour execution.

The common knowledge about the domain for the web services is axiomatised in a PSL ontology. The task on integration cannot be imagined in the absence of some common knowledge, especially if the autonomy and dynamic decisions of agents and services are aimed. Other languages proposed for describing the behaviour of web services (simple or composite) - like BPEL4WS - lack the power of expressing partial constraints [1]. PSL has support for such constraints, thus it facilitates behaviour specialisations. The use of activity ontology implies the existence of a hierarchy of activities and the capability of specialisation of the contained axioms.

The activity in the context of web service is characterised by IOPEs, like the FLOWS or OWL-s model. So the ontology will include description of activities and subactivities together with order constraints(that can be total or partial) and occurrence constraints. Besides activities, the ontology includes fluents that characterise the world state.

There are two kinds of activities: message activities - that receive and generate the messages- and behaviour activities - expressing the process flow of the service. The message activities are not the object of this work, but they will be included in future work.

IOPEs axioms

The axioms for IOPEs of each simple behaviour activity are:

- input preconditions - the knowledge needed by an activity to execute; they establish the fact that an action can not be executed before knowing its inputs. These inputs will be known after receiving a message for enactment of the service.

```
(activity sell)
(forall ?occ , ?clientid, ?prodid
   (implies
      (occurrence_of(?occ,          sell(?clientid,
  ?prodid))
      (and
        (holds (Known(client_id(?clientid)), ?occ))
        (holds        (Known(prod_id(?prodid)),
  ?occ)))))
```

For each occurrence of the *sell* activity it must hold the fluents about knowing the client id and product id.

- output conditions - the knowledge found after an activity occurrence. It is the information that the process provides to the requester. The output can be conditioned by fluents; it can be defined only for legal occurrences. An activity occurrence is legal if all the occurrence constraints are met.

```
(forall ?occ, ?clientid, ?prodid
   (implies
     (and
       (occurrence_of(?occ, shipment(?clientid,
  ?prodid))
        (legal(?occ)))
     (and
       (implies
          (served_area(?client, ?occ))
          (holds(shipment_time(?prod, ?clientid)))
        .....))))
```

For all legal occurrences of the shipment activity, the output will be the estimated time.

- effect conditions - express how the world state changed after the occurence.

The shipment activity is a long term activity, that can be interrupted and needs some time conditions. Eventually some compensation activities and exception activities should be introduced. The type for these activities can be state triggered activities.

Order constraints

The rules establishing relations between activities are expressed in terms of constraints on occurrences. Concepts from more PSL theories can be used in order to describe in a flexible manner the semantic of the activities

domain: subactivity theory, occurrence trees theory, and atomic and complex activity theory. Very important are the ordering constraints; these can impose from strict order - occurrence of activity a2 must occur immediately after activity a1 occurrence, or more less tight ordering - occurrence of activity a1 can be performed only before activity a2 occurrence. The first one implies the existence of both activity occurrences, while the second does not say anything about the need of existence of any of the two actions, and even more, between them can be performed other activities.

The occurrence constraints can model also the context. The occurrence of the same activity can be ruled by some constraints if it is subactivity_occurrence of some activity and by other constraints in other situation.

For example, for stating the fact that for all occurrences of buy activity must exist one occurrence of sell activity and one occurrence of shipment activity, we will use the following axiom.

```
(forall ?occ
 (implies
  (occurence_of (?occ, buy))
  (and (exists (?occ1, ?occ2)
     (occurrence_of(?occ1, sell))
     (occurrence_of(?occ2, shipment))
     (subactivity_occurence(?occ1, ?occ))
     (subactivity_occurence(?occ2, ?occ))
     (min_preced(?occ1, ?occ2, buy))))
```

We can consider another activity – c*hooseThe-BestShipment*, about it can be said that none of it's occurrences can be performed after an occurrence of the shipment activity. Notice that this activity is not needed when performing the buy activity, but it can occur.

```
(forall ?occ, ?occ1, ?occ2
  (implies
    (and (occurence_of(?occ, buy))
(occurrence_of(?occ1, chooseTheBestShipment))
(subactivity_occurrence(?occ1, ?occ))
(subactivity_occurrence(?occ2, ?occ))
(not min_precedes(?occ1, ?occ2, buy)))))
```

Activity ontology to Situation Calculus

In order to catch in situation calculus the constraints expressed in PSL, the first entity that we introduce is the after-occurrence fluent - it means that after the execution of action corresponding to an activity occurrence a fluent becomes true.

$$Poss(a, S) \supset [Done(act, do(a, S) \equiv act == a]$$

An action corresponds to each PSL activity. The concept of runtime execution of behaviour specification captured by PSL occurrences is enforced in situation calculus by the situation. For different type of constraints we give some general translation to situation calculus.

Order constraints The order constraint is expressed by adding to the precondition axioms of the *before* activity the negation of the after-occurrence fluent of the *after* activity.

Subactivity constraints For the complex activities, the after-occurrence fluent is true when all the after-occurrence fluents of the subactivities are true.

Existence + order constraints The *before* activity after-occurrence fluent is one of the precondition axioms of the *after* action, but only in the context of the activity which for these two are subactivities. ";" order construct introduced by [6] expresses that two actions must be performed, and there is an order. The sequence construct is not suited due to the possibility of actions that can occur between that two.

```
Proc a1 : a2 ≡
   a1 ; while ¬ Poss(a2) do
            (π Action a). ?Poss(a) ; a
        endWhile
endProc
```

But this expression is still not good enough for expressing the subactivity concept. For example, if we use it for expressing that for buying a product the sell and shipment actions are needed we restrict the buying action to this two and maybe something between them. So another subactivity can not be defined.

Therefore, we choose to assert and retract the information about the current complex activity occurrence and the fluents for subactivities are conditioned by these information.

Processes of simple web services

The entities and relations of the activity ontology underlie the behavioural description of each simple web service. We choose for these the situation language due to the similarity of the

theories: process and situation theory.
The desired behaviour is sketched in terms of situations and possible actions. Not only activities from ontology can be part of the descriptions. The specialisation of the ontological activities can involve new actions, but these new actions should depend on the ontological activities. For example, if a service provider private policy implies choosing the shipment among some favourites partners after the selling action(described in the ontology) it can describe this myC*hoice* action even if there is no corresponding activity in the ontology. But the provider has to describe the ordering and existence constraints relating to the *selling* activity or other used activities from ontology. So the precondition axiom for *myChoice* requests the knowledge about the price of the product(this is the output of the see activity) and the after-occurrence fluent *done* for selling activity is modified.

$$Poss(myChoice(Id, Prod), S) \equiv holds(price(Id, Prod), S)$$

$$Poss(a, S) \supset [done(sell(Id, prod), do(a, S)) \equiv a = myChoice(Id, prod)]$$

New fluents can be part of the description too besides the fluents from ontology. Some provider decides to apply some discounts for loyal customers.

$$Poss(a, S) \supset [holds(discount\_price(Id, Prod), do(a, S)) \equiv a = sell(Id, Prod)\ ]$$

The partial order of PSL ontology rise the opportunity of specialisation through more order constraints. In the same context of buying process, some services can request the existence of testQuality action on some after-sell service before shipment. Both actions have corresponding ctivities in ontology, but there is defined only an order constraint, not an existence one.

4. Agents for SOC

Web services are closely related to the agent programming paradigm. The definition of the web services architecture[11] states that "a web service is viewed as an abstract notion that must be implemented by a concrete agent".
The interactions between web services are currently limited to simple request-response exchanges [8]. For flexible and autonomous services a method for engineering expressive protocols is needed. The open problems are when a service needs another service, which service it needs and how can interact with it.
Another challenge is that the environments are rarely static and a MAS approach should deal with such dynamic nature.

Integration means joining together independent pieces of codes. These "building blocks" can be more or less publicly. The web service description is published for everyone, but in real life there are private business rules too that governs the service itself or its behaviour in a composite one.

In order to solve these problems we associate to each service an agent that intermediates the collaborating activities between services. These agent know the enactment description of the services (e.g. WSDL) and the activity ontology and change messages in order to establish the agreements that will govern the composition. The grounding activity is not discussed here, but the WSDL description allows querying a web service if the agents decide to.

Composite service

The composite service is specified in terms of activity ontology. Agents are reasoning in situation calculus on the behavioral descriptions of the served services using the activity ontology and on some special actions for agent-collaboration in order to reach a situation where the requested composite service is acquired. These special types of agent activities are delegation and foreign activity. When a service can't execute an activity it can delegate others to do it. Each service must have the possibility to perform such activity. The delegation activity asks for some kind of commitment from another service to do some common activity. The effect of the activity is a foreign activity - an occurrence activity that is part of the behavioural description of another service.

Speech-acts

We use the ideas of introducing speech-acts from [8]: *call-for-proposal*, *propose*, *inform*, *accept*. Each one has the following form s.something(r, γ). The speech-act specifies the initiator, a list of recipients and an action taken from the activity ontology. Activities from speech-acts can be executed by the served service or they can be delegated to other services.

Each speech-act has associated an action in the situation calculus, in order to include such actions in the synthesis of the composed service. We will call these actions collaboration-actions. Associated to these actions, there are fluents for asked(Action), proposed(Action) and accepted(Action). A special note is that the ability of doing an action is requested, not the execution. Also for constraints taken from ontology the *gen_poss* fluent is used.

Call-for-proposal The call-for-proposal collaboration-action is possible whenever no other action is possible (or possible all the time, but desired only in this case). The action for that an executor is searched can be taken from the needed actions (but not possible yet) from the ontology.

$$Poss(call\_for\_proposal(A)) \equiv not(asked(A)) \wedge gen\_poss(A) \wedge (not(known(A)) \vee not(poss(A)))$$

The precondition axiom expresses the fact that an agent should send a call-for-proposal only for actions that are not delegated by other agents (in order to avoid two or more step delegation for the same action).

When a call-for-proposal is received by an agent, the asked action becomes desired in that situation. If that action is possible too in the current situation, a propose collaboration-action must be executed, sending back to the initiator of the interaction a propose speech act.

Propose The propose collaboration-action is possible whether a call-for-proposal message was received or not. If this action is done as a response to a call-for-proposal and one of the simple or complex possible actions known by the service is the asked action, then the proposed action is the asked action. If the asked action is not among the known action but there are known actions that are needed for the asked one (from the ontology) then the response will include a propose for all these actions.

$$Poss(propose(A)) \equiv known(A) \wedge poss(A) \wedge gen\_poss(A) \wedge (asked(A) \vee asked(B) \wedge subactivity(A, B))$$

Accept Whenever a proposal is received, the fluent for proposed actions is changed for that actions. From the current situation a new final situation is reached, using the proposed actions and its own actions. If it can be reached a situation that meets the goal, then accept collaboration-actions are executed for each proposed actions included in this theoretical final situation.

Inform and request These two collaboration actions are needed when the choice of an action and its executor is based on some quality criteria; some more information from the possible delegated service is needed to decide whether or not to delegate (e.g. price of something). A special case of inform is the acknowledge of finishing the execution of a delegated action.

Based on the autonomy of web services, the situations of the services are each other independent. The only common information that can appear are the fluents that characterise some actions in the ontology. For these common information there are request/inform collaboration-actions.

Synthesise of composed service

The synthesis of the composed service is the result of agent collaboration. One agent initiates the process and then other agents are involved based on collaboration-actions. Each agent has a local view about the composed service, knowing only the actions that it committed to do.

proc *compose(G)*
  while ¬*done(G) do*
  (πAction a). gen_poss(a) ∧poss(a) ; a
  endWhile
endproc

The choice of next action to execute is nondeterministic and it is based only on the preconditions axioms. When the number of actions increases, the complexity of choosing an action in this way is greatly affected.

In [6] cases in with an action can be executed are restricted by requiring not only that an action *a* is *Poss(a, s)*, but further that is *Desirable(a,s)*. This way, the search space for actions is constraint further. The Desirable fluent can be extracted from the PSL action ontology. For stronger constraints, a new fluent can be used *Needed(a,s)* that can express the need of executing an action in a given situation, similar to expressing a sequence of actions. Some attention have to be paid when choosing the fluents, being known the fact that a situation is not a state, it is a history. From two different states it can be reached another state throw two different actions, but if a same situation is reached from two situations then those two situations are the same and the actions are the same too.

The reliability aspect and utility for each service, together with some quality of service requirements imposed by the requester of the composite service are possible to achieve in MAS context. Based on facts learn from the recent interactions, each agent can build a reliability profile for other agents and use it in choosing the next action.

5. Conclusions and Future Work

The ideas presented in this work are still in an incipient phase. The primary goal is unifying technologies for process specification, multi-agent systems and semantic web in order to improve autonomous and reliable integration of services as SOC already emphasised as being the current challenge. Further work include refining the collaboration-actions and the planning throughout ranges of desirability for each action according to ontology constraints and QoS requirements. Further more, in ontology we will add some fail recovery action that might include compensation too.

References

[1] Battle, S. and Bernstein, A. and Boley, H. and Grosof, B. and Gruninger, M. and Hull, R and Kifer, M and Martin, D and McIlraih, S. and McGuiness, D. and Su, J. and Tabet, S. (2005) *Semantic Web Sevices Ontology ,* Semantic Web Services Initiative.

[2] Berardi, D. and Calvanese, D. and De Giacomo, G and Hull, R. and Mecella, M. (2005) *Automatic Composition of Transition-based Semantic Web Services with Messaging,* Proc. of the 31st Int. Conf. on Very Large Data Bases, Trondheim, Norway, ACM Press.

[3] Berardi, D and Calvanese, D. and De Giacomo, G. and Lenzerini, M. and Mecella, M. (2003) *Automatic composition of e-services that export their behaviour,* Proc. of International Conference on Service Oriented Computing, Trento, Italy, Springer.

[4] Bock, C and Gruninger, M. (2005) *PSL: A Semantic Domain for Flow Models,* Software and Systems Modelling Journal, 4:2, Springer.

[5] Bultan, T. and Fu, X. and Hull, R. and Su, J. (2003) *Conversation specification: a new approach to design and analysis of eservice composition,* Proc. of 12th Int. World Wide Web Conference, Budapest, Hungary.

[6] McIlraith, S. and Son, T. (2002) *Adapting Golog for Composition of Semantic Web Services,* Proc. of the Eighth International Conference on Principles and Knowledge Representation and Reasoning.

[7] Gerede, C. E. and Ibarra, O. H. and Ravikumar, B. and Su, J. (2005) *On-line and minimum cost ad hoc delegation in e-service composition,* Proc. of IEEE International Conference on Services Computing, Orlando, SUA, IEEE Computer Society.

[8] Paurobally, S. and Jennings, N. R. (2005) *Protocol engineering for web services conversations*, Journal of Engineering Applications of Artificial Intelligence, 18(2).

[9] Shen, Z. and Su, J. (2005) *Web service discovery based on behavior signatures,* Proc. of IEEE International Conference on Services Computing, Orlando, SUA, IEEE Computer Society.

[10] Singh, M.P. and Huhns, M.N. (2005) *Service-Oriented Computing: Semantics, Processes, Agents,* John Wiley and Sons, Chichester West Sussex.

[11] Booth, D and Haas, H. and McCabe, F. and Newcomer, E and Champion, M and Ferris, C and Orchard, D. (2003) *Web Services Architecture,* World Wide Web Consortium.