Service Composition through Agent-Based Coalition Formation

Ingo Müller, Ryszard Kowalczyk
Swinburne University of Technology
Faculty of Information and Communication Technologies
Hawthorn, 3122 Victoria, Australia
{imueller, rkowalczyk}@ict.swin.edu.au

Abstract

Service-oriented architectures, such as Web services and Grid services, have been recognised as advantageous architectural styles for future enterprise and scientific applications. However, on top of already available middleware layers, many problems regarding the management of service compositions have been identified as open issues. This paper introduces the concept of an agent-based framework for dynamic and adaptive service composition. This approach consists of a multi-agent system which comprises software agents requesting and providing composite services based on semantically enriched Web services. The composition process is performed by software agents engaging in a coalition formation process to collectively provide composite services that meet complex user requests.

1. Introduction

Service-oriented architectures, such as Web services and Grid services, have been recognised as advantageous architectural styles for future enterprise and scientific applications. However, on top of already available middleware layers, many problems regarding the management of compositions of multiple services have been identified as difficult.

Basic service-oriented standards [7] enable the access of core business functionality and also allow the creation of static compositions of services in terms of one software entity accessing successively multiple Web services. Those approaches are static, not flexible, and not reliable due to their hard-coded tightly-coupled nature.

Researchers in the Semantic Web domain aim at overcoming the given drawbacks by dynamically discovering and composing Web services based on a generic workflow description at runtime [20]. A broker agent discovers available Web services and selects the most suitable Web service based on matchmaking of a client request with service advertisements (e.g. OWL-S profile) in a locally stored repository. Semantic Web approaches are more flexible and add fault-tolerance by abstraction for discovering appropriate services (rather by service type than by name). However, they suffer from their centralised nature. Broker based architectures presume necessary knowledge about services to be available at one central spot in the system and they introduce a single point of failure since all interactions run through the broker entity. Furthermore, they do not include negotiation of quality of service (QoS) parameters, which we expect to make the selection process even more flexible. The latter problem is addressed by current efforts such as the Adaptive Services Grid (ASG) project [2]. However, decision making and knowledge is still managed in a centralised manner.

We want to go beyond the prevailing centralised character and incorporate QoS negotiations in order to improve flexibility and fault-tolerance of the service composition process. Therefore, we are working on novel techniques for dynamic and adaptive service composition by applying agent technology. Software agents truly offer a complementary approach for the development of distributed systems with a strong focus on intelligent and dynamic behaviour, autonomous adaptation, and goal-based interaction. In particular, software agents request and provide services based on semantically enriched Web services. The composition process is performed by software agents engaging in a coalition formation process to collectively offer composite services that meet complex user requests.

This paper reports on ongoing work on the underlying software framework of our approach. It introduces a two layer architecture and specifies the design and design goals of an incorporated multi-agent system in Sec. 2. After that, we outline in Sec. 3 our concept for service composition based on agent-based coalition formation whereas Sec. 4 reports on experiences with our first prototype. Sec. 5 discusses related work before the paper is finally summarized and an outlook is given in the last section.
2. Overall Architecture

We consider service-orientation as the enabling technology for uniform and standardised access to core business functionalities. This major concern is reflected in our choice of a two-layered architecture, which strictly separates between actual business functionality and corresponding management functionality.

Basic Web services enriched with semantic information form the base of the bottom layer (service layer). On the other hand the top layer (composition layer) contains a multi-agent system being in charge of the provision of necessary management functionality in order to achieve dynamic and adaptive service composition. Dynamic composition means the creation of composite services at runtime. Adaptive composition describes the composition process to be aware of exceptions or in general to provide fault tolerance. Since we focus in this paper on the realisation of the top layer we describe in the remainder of this section the architecture of a multi-agent system forming the foundation of our service composition approach as depicted in Fig. 1.

Our multi-agent system only comprises two different agent types; user agents and service agents. A user agent interacts with its user in order to obtain a plan which details about the desired composite service and its expected outcome. Furthermore, the user agent can be configured to engage on behalf of its user in negotiations of functional or QoS parameters. One user agent refers to exactly one user.

A service agent represents a service entity. It provides management functionality necessary to enable a service instance to participate in the service composition process. Therefore, a service agent monitors the status of its associated service entity, engages in the coalition formation process, and possesses negotiation abilities in order to optimise benefits. One service agent always represents a single service instance. However, service providers might deploy several service agents for a single service.

Hence, we apply software agents as active representatives not only for the client side but also for the server side, the Web services. These agents play a key role in the service composition process because of their active nature in contrast to their passive counterparts in broker-based approaches.

The third key component is a blackboard [10]. The blackboard is an information space providing user agents with the opportunity to leave activity plans (requests for composite services) and to gain information about proposals of service agent coalitions. On the other hand it provides service agents with the opportunity to read activity plans and to leave and obtain service advertisements in order to let service agents engage in direct interactions during coalition formation, proposal generation, and proposal negotiation. In addition, it validates agent coalitions, performs integrity checks, and provides trust-related information.

2.1. Design Goals

The overall architecture is the visible result of a design process, which is driven by considering several design issues. Here we outline major design goals that have been addressed.

2.1.1. The multi-agent system should automatically adjust to new services or services types. It is extensible and adaptive. This goal is achieved by referring all agents, services, and the blackboard of a particular application domain to a global domain ontology. In this context, a domain ontology describes concepts, data, and relations between them within one application domain such as the domain of travel planning or e-shopping, etc. The ontology contains service type descriptions that specify mandatory input, output, and QoS parameters as well as optional parameters. We assume that at least one concrete service instance exists per ontology-defined service type. Each service instance keeps its individual service description, which must match the mandatory elements of the domain ontology but can add optional elements. Optional elements, however, must be reported to the global ontology for the sake of completeness.

In addition, the interface between user agent and user must be adaptable as well. Later we extend our approach with a query language enabling users to input their needs. Again, the user input is based on keywords defined with the service descriptions in the domain ontology and can be validated by a user agent at runtime. Therefore, user interfaces do not have to be adjusted and recompiled for every new service or service type offered in the system.
2.1.2. Apply decentralised decision-making and distributed knowledge. Most other approaches in the context of agent-based service composition apply Broker or Mediator patterns and thus create a central point of control and make use of centralised decision-making. Moreover, these central components must have global or even complete knowledge of available services of their environment. However, this assumption cannot be held upright in an open environment such as the Web because of its dynamic nature. For that reason we establish agents with only limited local knowledge. Agents gain necessary information through the blackboard allowing them to engage in service composition processes based on distributed locally performed decision-making.

2.1.3. Employ intelligent agents. When we make use of the term software agent we refer to what is described in [21] as “a weak notion of agency”. We want to go beyond simple reactive agents and therefore apply the properties of autonomy, social ability, reactivity, and pro-activeness to all agent types incorporated in our multi-agent system. Both agent types of our approach are deliberate agents possessing distinct agent capabilities. A user agent is able to communicate with its user and to derive plans from that interaction. Furthermore, it can engage in negotiations on behalf of its user. In the opposite, a service agent is self-contained; it knows how to provide a service and what parameters it takes to be performed successfully. Moreover, service agents possess monitoring capabilities.

Information about the current state of an associated service entity can be used to improve the agent’s reasoning capabilities. Service agents negotiate with other agents in the system, particularly with coalition members, in order to gain benefit. They engage in coalition formation process which is executed based on inter-agent interactions.

2.1.4. Modular design. Our framework is designed to enable easy extension of core functionalities of our approach, in particular with the two types of software agents. For that reason, both agents implement their internal state based on an abstract finite state machine. The finite state machine specifies the data items an agent possesses and the actions that manipulate this data, including the interaction protocols that may trigger actions. The abstract finite state machine can be extended either to refine an existing behavioural model (specifying interaction protocols) of an agent or to completely develop a new behavioural model of an agent. Furthermore, our software agent design supports exchangeability of components, e.g. for determining utility or decision making needed in negotiations, for decision making during coalition formation, or for determining the satisfaction level of an agent during proposal generation. Our modular design supports successive extension of our framework with more complex implementations and also the comparison of different implementations, e.g. for coalition formation, proposal generation, or negotiation strategies.

3. Composition Process

This section outlines our concept for agent-based service composition process. Our approach is based on agent interaction protocols and focuses on distributed knowledge and a very much decentralised decision-making. The composition process comprises of four phases organised in sequential order. In each of the phases several actions are performed. Each phase produces an outcome that is used as input of the succeeding phase. The phases which are discussed in the following are the

- Plan Generation Phase,
- Coalition Formation Phase,
- Negotiation Phase, and
- Contracting Phase.

Hence, we solely focus on service composition aspects of the service orchestration process. Therefore, we just consider necessary activities and data structures until a complete and stable composition of multiple services has been achieved. Further issues such as the enactment and monitoring of composite services are out of the scope of our work.

3.1. Plan Generation Phase

The first phase mainly covers interactions between a user and his associated user agent. The user inputs a problem he wants to get solved. Thus the user specifies, according to the domain ontology, activities such as “print a document” together with mandatory (e.g. paper format, number of copies) and optional (e.g. colour print) input parameters and QoS values (e.g. price). After the user has completed his query the user agent verifies the query. The agent can request additional information from the user in case of omissions, ambiguity, or erroneous inputs.

The user input is not necessarily an ordered sequence of activities. For that reason, the user agent validates all given activities in order to obtain relations between them. These relations are detected based on similarities of the sets of input and output parameter types of different activities. Moreover, these relations can be used for deriving an ordered workflow of connected activities. If the user agent recognises relations between activities it requests the user’s approval.

Finally, the user agent creates a single machine-readable document called activity plan. An activity plan describes a composite service in a generic way. It contains an ordered workflow-like organization of activities. The description of an activity includes information about input, output, and
3.2.1. Service Advertisement.

The starting point of this action is a newly published activity plan on the blackboard. Before a service agent can advertise its service for a particular activity plan it must get to know about it. This information can be obtained in two different ways. Firstly, service agents can regularly check the blackboard for new activity plans. They send a keyword list of service names they provide to the blackboard. If one or multiple activities of newly published plans match the keyword list the blackboard sends back information about them. Secondly, service agents can register a listener with the blackboard. All blackboard listeners are called in case a new activity plan got published on the blackboard. If one or multiple activities of the new plan match the capabilities of the service a service agent represents, the listener sends information about the plan to the service agent. Hence the keyword matching is always feasible since all participants of the multi-agent system refer to the domain ontology. Moreover, each service agent owns knowledge about the services it provides.

Now that a service agent knows a new activity plan, it can advertise its service to other service agents in order to allow coalition formation. Therefore, a service agent leaves an advertisement or note at the blackboard since we assume agents not to have general knowledge about other agents. The advertisement is associated with a specific activity plan and in particular with the activities an agent can provide services for. Each advertisement is given for just a limited time (equal to the activity plan’s live time at most) and can be deleted by its owner again. In addition, service agents are allowed to leave advertisements for multiple activity plans at a time.

3.2.2. Teaming Up. After a service agent has placed its advertisement(s) on the blackboard it checks for advertisements of other agents in order to contact potential candidates. It can happen that a service agent checks for other advertisements but receives an empty result. In that case the service agent checked the blackboard before other agents were able to leave their advertisements because of the inherent asynchronous nature of a multi-agent system. Therefore, the user agent needs to check the blackboard periodically or registers a listener which informs the agent about new advertisements referring to a certain activity plan.

A service agent decides whether or not to contact another service agent based on its reasoning capabilities. In the positive case a service agent requests other service agents to form a coalition. This can be achieved in two ways. Firstly, a service agent can request if other service agents are willing to join its coalition. Secondly, a service agent can ask for participation in another coalition.

This simple approach turns out to get very complex depending on how coalition membership is achieved. In the simplest case, each service agent interacts with all available service agents. Each service agent tries to establish its own coalition and joins any coalition in parallel, until it has a complete set of service agents encountered. If so, a service agent requests approval from all candidates. This approach forces service agents to work together although they might have objections regarding particular coalition members. On the other hand more complex approaches are possible, e.g. all current coalition members vote for or against potential candidates. In addition, service agents should be able to leave a non-confirmed coalition at any time or to evict non-cooperative members.

Another aspect of the coalition formation process is the matching of input and output parameters of consecutive services. Since services can support optional parameters there

\[
<\text{activity plan}> ::= \langle\text{header}\rangle + \langle\text{body}\rangle \\
\langle\text{header}\rangle ::= \langle\text{ownerInfo}\rangle + \langle\text{timestamp}\rangle + \langle\text{deadline}\rangle \\
\langle\text{body}\rangle ::= \langle\text{id}\rangle + \{\langle\text{task}\rangle + \{\langle\text{overallQoS}\rangle\}\} \\
\langle\text{task}\rangle ::= \langle\text{id}\rangle + \langle\text{input}\rangle + \langle\text{output}\rangle + \{\langle\text{localQoS}\rangle\} \\
\langle\text{overallQoS}\rangle ::= \{\langle\text{parameter}\rangle\} \\
\langle\text{input}\rangle ::= \{\langle\text{parameter}\rangle\} \\
\langle\text{output}\rangle ::= \{\langle\text{type}\rangle\} \\
\langle\text{parameter}\rangle ::= \langle\text{name}\rangle + \langle\text{type}\rangle + \langle\text{value}\rangle \\
\langle\text{type}\rangle ::= \langle\text{id}\rangle + \langle\text{range}\rangle
\]

Figure 2. BNF notation of an activity plan.
could be gaps between their interfaces. For that reason, service agents that agree on forming a coalition must check the signatures of their input and output interfaces in order to avoid errors during service enactment.

If a coalition is complete (each activity of a plan has been allocated to a service agent) the coalition candidates must confirm their membership. After all coalition members have approved, the coalition registers with the blackboard for a certain plan (e.g. all members leave an appropriate message). The blackboard verifies the coalition and removes all corresponding advertisements from the actual plan. The blackboard has to assure that a specific agent is only member of exactly one coalition per plan for the sake of fairness. Of course, an agent is allowed to participate in different coalitions regarding several activity plans. However, if a coalition wants to register with the blackboard comprising of an agent, who is already member of another registered coalition, the coalition is rejected.

The outcome of the teaming up is a set of registered coalitions ready to engage in proposal generation and coalition to user agent negotiations. Hence, the teaming up does not finish at a certain time but produces coalitions over a period of time which is only limited by the deadline of the activity plan.

3.3. Negotiation Phase

The main purpose of the coalition formation process is to provide a user agent, having published an activity plan before, with proposals. The proposal contains a comprehensive offer for all QoS parameters of an activity plan and is subject to negotiation with the associated user agent. Coalition members must create a proposal once they have confirmed their willingness for cooperation. A coalition’s proposal should be a competitive offer able to compete with proposals of other coalitions.

3.3.1. Proposal Generation. The proposal generation bases on the agents’ ability to express their expected benefit to other coalition members based on their individual utility. The utility is determined usually with a mathematical function that maps the range of valid values of QoS parameters into the interval [0,1].

In general we can distinguish between two different approaches for proposal generation. First, coalition members determine the coalition’s overall utility by deriving it from local agent utilities. A very simple way to achieve this is to let coalition members negotiate weights, which will be used as factors for calculating a weighted sum of their local utilities. In this case the weights also define the fractions of the total benefit of a coalition each agent is eligible for. A major drawback of this approach is the fact that all coalition members have to reveal their local utility in order to prevent cheating. Secondly, coalition members directly negotiate overall values for all QoS parameters for example with an iterated contract net protocol. Thus, each agent informs the other coalition members which values it prefers (e.g. price =10 cent). The benefit each agent can gain is specified by that concrete value. The values for one QoS parameter type are summed up. All coalition members keep their utilities confidential. On the other side this approach causes the disadvantage that the coalition has to perform this kind of proposal generation for every new counter proposal from the corresponding user agent all over again.

Another important aspect is the software-like realisation of the proposal generation. Different approaches are possible. First of all, the coalition members can decide on a leader agent. The leader becomes a central point of the coalition and acts as a conductor for coordinating all proposal negotiation related interactions. The leader determines the final outcome of the proposal generation. Secondly, the coalition members pass a token between each other, e.g. in the order of the workflow described by the activity plan. When an agent takes possession of the token it can alter the current value of negotiated values. This approach can be applied iteratively until a certain termination condition is met. After termination the token keeps the result of the proposal generation. Thirdly, a fully distributed approach will imply all coalition members to interact directly for determining a proposal. A member agent sends its preferred set of values as a multicast message to all other coalition members. A member agent also receives messages from all other coalition participants. Once it has received messages from all partners the agent has to evaluate whether or not it accepts the current proposal. Therefore, a service agent needs to estimate its level of satisfaction. If the agent does not accept the value it initiates another round requesting adaptation from its coalition partners. The last approach allows for interesting experiments with social behaviour based on different satisfaction levels. Agents will have high satisfaction when they made a lot of turnover recently whereas less successful agents desperately try to converge the coalition’s proposal, based on their low satisfaction rate, close to the claim of the requesting user agent.

In general, after a coalition has agreed on a proposal the members engage in negotiations with the corresponding user agent.

3.3.2. Coalition vs. user agent negotiation. By submitting its proposal, a coalition initialises the negotiation with the corresponding user agent. In general, different coalitions compete for an activity plan. Therefore, the associated user agent is free to choose. Since we assume that a user agent is programmed to maximise the outcome for its user, it will initiate multiple negotiations with all competing coalitions in order to obtain the most beneficial one.
We can mainly distinguish between two different approaches, mediated and interactive. The former involves the blackboard as a mediator between the opposing sites. Both, coalition and user agent, alternately place their latest proposals in a specific place of the blackboard. The blackboard informs associated counterparts of a negotiation after it has received a new proposal. The latter approach bases on direct communication between the user agent and a representative of a coalition. For that reason, coalition members determine a trustee by voting, e.g. the agent with the best success rate, or assign a specific agent, e.g. always the agent affiliated with the first activity of the associated activity plan.

Depending on the method used for proposal generation, it can happen that the coalition members repeat the proposal generation process iteratively with each new counter proposal received from the user agent. Finally, the user agent rejects completely or declares one coalition as winner.

3.4. Contracting Phase

After the user agent has decided on a winner it sends a confirmation message to the winning coalition and a cancellation message to all others. The user agent and the coalition then fix contracts such as Service Level Agreements (SLA) using the blackboard as trustee. Hence, the blackboard only allows registered coalitions to contract. Accordingly it performs integrity checks and might prevent a coalition from contracting in case of ambiguities. The contracting process can become more complex if all sub contractors of a coalition, the service agents, fix contracts between each other as well. The created contracts are binding for the affected service agents until the composite service enactment is completed.

4. Basic Implementation

This section summarises a first prototype of our software framework for agent-based service composition which aims at assessing the general feasibility of our concept. The base of our implementation is formed by the Tracy agent toolkit [8]. We decided on Tracy because of its simplicity regarding the implementation of software agents and its highly extensible software design supporting easy extension with additional infrastructure functionality such as a blackboard.

According to the architecture outline of Sec. 2, we need to implement a blackboard and two different agent types. Furthermore, our implementation comprises a graphical user interface (GUI) in order to ease the creation and configuration of user agents and service agents as well as for logging and debugging purposes. The GUI supports the creation of service agents by loading and starting configuration files. A configuration file needs to be structured hierarchically, containing information about all service agents to be started. Currently, the file must specify per service agent an agent identifier, the supported service type, and a QoS parameter called price. User agents can be created as well. An input form pops up for each newly created user agent requesting a user inputs of an activity plan and an initial value for the QoS parameter price, which is used for user agent vs. coalition negotiations. In that context, it needs to be mentioned that the ontology is kept simple at the moment. It only defines simple service type descriptions comprising a service type name and an associated QoS parameter price.

User agents are implemented with a finite state machine. In idle state, a user agent accepts the input of a new activity plan which comprises of a sequentially ordered set of service type names and a total price as overall QoS parameter. After passing the plan to the blackboard, the user agent switches to state waiting. It receives proposals, in case at least one coalition could be successfully formed. Now the user agent enters the negotiation state and concurrently performs a Contract Net Protocol-based negotiation with all candidate coalitions (Fig. 3). Based on the data of the given proposals, the user agent gets the chance to negotiate the price of the requested composite service with all candidate coalitions. It initiates the Contract Net Protocol by creating a call for proposal which is send to all coalitions. Then, the user agent waits for valid proposals and currently accepts the proposal with lowest price. After determining a winner coalition, the user agent stores the agreed price, informs all coalitions about its decision, and returns into idle state. Currently, the implementation of user agents is rather simple. However, this is caused by leaving out complex ontology, user querying and planning on one side, and applying a very simple negotiation protocol on the other side.
Service agents are also implemented with a finite state machine. A service agent starts in idle state and registers a so called plan listener with the blackboard. If a new plan becomes available on the blackboard, that comprises service types matching the service type specified with its plan listener, a service agent receives information about the plan. Then, a service agent registers an advertisement with the blackboard for each service type of the new plan it supports. After that, a service agent registers another listener in order to get informed when new service advertisements for the current plan get published and switches then into candidate state in which it is ready for coalition formation. Now, a service agent obtains service advertisements from the blackboard. They form the base for the coalition formation interaction protocol (see Fig. 4). A service agent sends a join-request message to all other service agents that have published advertisements for service types it cannot provide on its own but that are required by the given activity plan. Currently, service agents do not invoke complex decision making for identifying adequate coalition partners. After a service agent has received join-accept messages for all service types of the plan it cannot provide, it attempts to finalise the coalition by sending coalition-request messages to the corresponding service agents. After all candidate agents have approved their participation, they all precede into the member state in which they generate a proposal.

The proposal generation process is the coalition’s internal negotiation in order to create a proposal for the requesting user agent. Currently, the proposal generation is implemented with a two-phase token-like approach. In the first phase, beginning with the service agent associated with the first service type in the sequential activity plan, a token is send to the coalition member’s successor. The token contains information about the member agent’s price claim and the total price of the coalition. The price claim of a service agent is determined based on utility function $u$ mapping all values of decision space $P$ into interval $[0,1]$. 

$$u : P \mapsto [0,1]$$

Currently, a one-attribute decision space price is supported. The utility function $u$ maps all prices in the interval $[p_{\text{min}},p_{\text{max}}]$ to 1, all other prices to 0. $p_{\text{min}}$ defines the minimum price a service agent charges for the use of its corresponding service. The service agent is configured with the minimum price during its creation. The maximum price is calculated with $p_{\text{max}} = p_{\text{min}} + v/n$ in which $v$ determining the price given with the activity plan and $n$ representing the number of coalition members. A service agent starts proposal generation at $p_{\text{max}}$ and decreases its claim by 10 per cent with each new iteration of the two phase token protocol. The total price of the coalition is calculated as the sum of all individual price claims. After the service agent, associated with the last service type in the activity plan, has received and updated the token the second phase commences. The token is now sent in the predecessor-direction of each coalition member in order to collect the members’ voting whether they are satisfied with the current proposal or not.

A service agent’s satisfaction is determined by another utility function $s$ mapping the quotient between the activity plan given price ($p_{\text{req}}$) and the coalition’s total price ($p_{\text{tot}}$) in the interval $[0,1]$. 

$$s : p_{\text{tot}}/p_{\text{req}} \mapsto [0,1]$$

The satisfaction level of a service agent is high, when $p_{\text{tot}}/p_{\text{req}} \approx 1$ and decreases for $p_{\text{tot}}/p_{\text{req}} \rightarrow 0$ or $p_{\text{tot}}/p_{\text{req}} \rightarrow \infty$. A service agent obtains its satisfac-
tion by calculating the utility function $s$ based on the data provided by the token. Starting with satisfaction level 1, the service agent checks whether the current satisfaction $s$ exceeds the given level or not. If the level is exceeded, the service agent marks its status as satisfied in the token and passes the token to its predecessor. If the level was not exceeded, the service agent marks its status as not satisfied. The two phase token protocol is iterated, with each iteration decreasing the critical satisfaction level by 10 per cent, until all coalition members are satisfied. The service agent associated with the first service type in the activity plan sends the agreed proposal to the user agent. In the following, all service agents return back into idle state after the user agent has decided on a winner coalition and has informed all participating service agents.

The third key component of our approach is the blackboard. It is globally known to all agents and contains a tree-like structure for storing activity plans and their associated components as well as different action listeners as depicted in the class diagram in Fig. 6.

Currently, our focus is on decentralised coalition formation and proposal generation without coalition leader. So far, our first implementation shows good results. However, we want to extend this first prototype to achieve more complex agent behaviours as introduced in this document.

$$\begin{align*}
\text{Plan} & \quad \text{Listener} \\
\text{Plan} & \quad \text{Blackboard} \\
\text{Header} & \quad \text{Body} \\
\text{Task} & \quad \text{Coalition} \\
\text{Advertisement} & \quad \text{Listener} \\
\text{Advertisement} & \quad \text{OUTpar} \\
\text{QOSpar} & \quad \text{INpar} \\
\end{align*}$$

Figure 6. Blackboard structure class diagram.

5. Related Work

Along with the latest developments of Web and Grid services, the research community has been debating how to integrate software agents with service concepts. The enormous effort results into a vast amount of approaches and implementations, e.g. [3, 4, 5, 9, 11, 12, 14, 19, 20, 22]. In general, these approaches make use of software agents to manage the composition and enactment of multiple services. Most papers describe broker or mediator architectures that implement middle agents for centralised decision making based on complete knowledge about available services. Software agents that represent services are usually simple passive software entities acting as proxies. In some cases, they add negotiation capabilities. Our approach differs from existing approaches in two aspects. First, it is based on a decentralised decision making based on distributed knowledge. Secondly, our approach applies agent-based coalition formation for the service composition process, which is, to the best of our knowledge, a unique approach so far.

Referring to the second aspect, we have to consider existing work on agent-based coalition formation. Generally, coalition formation related literature distinguishes between two major approaches concerning coalition formation. A game theory based approach employs sets of self-interested agents that compete for resources, whereas a complementary approach bases on agents that can achieve their goals only through collaboration. The degree of collaboration depends on the level of social commitment of agents. Self-interested agents [17] are maximising their individual utility, whereas group-oriented agents maximise group utility [16]. Our approach fits in the category of collaboration-based approaches. However, the degree of social commitment of each agent is not fixed per definition but variable and evolves with time.

All current approaches are based on restrictive assumptions in order to achieve stable or optimal agent coalitions, such as a priori knowledge about the value of a coalition, the set of possible coalitions, and/or agents and their capabilities which is known to a central coalition leader [1]. In addition, the quest for stable or optimal solutions is paid off with poor computational performances, e.g. exponential complexity [6] or polynomial complexity [17]. Our approach does not rely on assumptions that restrict its applicability in dynamic environments. In contrast, it leads on the one side to high flexibility for coalition member selection and good performance but causes on the other side sub-optimal solutions. A coalition’s benefit is distributed among its members according to the concessions made during proposal generation. Our coalition formation process might also terminate with no result, depending on the characteristics of available services.
Finally, we would like to point to some interesting approaches that will affect our future work. On the one hand, trust and motivation in multi-agent communities [13] can be used to establish more stable coalitions. On the other hand, learning can help to improve efficiency, stability, and profit of a coalition formation process ([15, 18, 1]).

6. Conclusion and Outlook

We outlined a two layer architecture and a general concept for dynamic and adaptive service composition based on software agents. The architecture defines a service layer which is focussed on standardised service access, and a composition layer containing a multi-agent system for the management of the service composition process. The architecture forms the base of our approach for service composition which bases on agent-based coalition formation. A first prototype proofs the overall concept as feasible but also pinpoints the need for further effort to improve the efficiency of our approach.

Foremost, we are aiming at more sophisticated methods especially for the coalition formation process applying sophisticated reasoning abilities, voting, and satisfaction calculation. Accordingly, we want to check our solution for crucial aspects such as avoiding deadlocks or preventing malicious agent behaviour.

Secondly, we expect the incorporation of historical data to improve trust in general and the capabilities of service agents to better assess quality and willingness to cooperate of other service agents. In summary, our basic approach can be enriched with existing approaches that can be compared this way. We expect an increase in efficiency in the coalition formation process, a reduction of communication effort, and speed up for negotiation phases.

Finally, as a long term goal, we aim at the design of an appropriate user interface for user agents including the development of an easy to use query language.

References


