Automated Composition and Reliable Execution of Ad-hoc Processes*

Walter Binder[1], Ion Constantinescu[1], Boi Faltings[1], Klaus Haller[2], and Can Türker[2]

[1] Artificial Intelligence Laboratory, Swiss Federal Institute of Technology Lausanne (EPFL)
{walter.binder,ion.constantinescu,boi.faltings}@epfl.ch

[2] Database Research Group, Swiss Federal Institute of Technology Zurich (ETHZ)
{haller,tuerker}@inf.ethz.ch

Abstract. This paper presents an architecture to automatically create ad-hoc processes for complex value-added services and to execute them in a reliable way. The uniqueness of ad-hoc processes is to support users not only in standardized situations like traditional workflows do, but also unique non-recurring situations. Based on user requirements, a service composition engine generates such ad-hoc processes, which integrate individual services in order to provide the desired functionality. Our infrastructure executes ad-hoc processes by transactional agents in a peer-to-peer style. Moreover, the service composition engine is used to re-plan in the case of execution failures.

1 Introduction

Users benefit from ad-hoc processes and distributed implementations of their execution environment, since for the first time information systems support users not only in standardized situations like workflow systems do. An example for an ad-hoc process is planning an evening. The user states its preferences, e.g., comedy movie, restaurant with French cuisine. Then, an ad-hoc process reserves a table in a restaurant and a ticket for a movie. In this scenario a personal agent (PA) tries to find a cinema showing a "good" comedy and a good French restaurant. For this purpose, the PA contacts a movie recommendation service in order to discover a good comedy, as well as a yellow page directory to select a French restaurant. Afterwards, the PA searches for a cinema which plays the selected movie and uses a recommendation service to ensure that the selected restaurant has a good rating. Finally, the PA returns a restaurant/cinema combination to the user. As it can be seen in this example, this new generation of ad-hoc processes can support users in their everyday life where situations are unique and usually not appear in the same way more than once.

In this way, ad-hoc processes imply a shift in the usage of process technology: Whereas employees in their offices can be assumed to be static, this does not hold for mobile users. Ad-hoc processes respect this by implementing location-aware services, e.g., time table information systems considering the current user location. Also, centralized approaches are not suitable for ad-hoc process execution. Besides, dynamic process changes must be supported. In case of the evening planner, the user might miss the metro and therefore not reach the cinema in time. Also, the ad-hoc process might figure out that there is no cinema in town. Both requires that the ad-hoc process is able to dynamically adapt, e.g., with the help of a decision support system such that there is a valid plan constructed for the evening.

* The work presented in this paper is supported by the Swiss National Science Foundation as part of the project MAGIC (FNRS-68155).

This paper presents an infrastructure which provides a service composition engine that allows to create and dynamically adapt ad-hoc processes. The underlying process engine ensures the reliable execution of processes by using transactional agents.

Section 2 discusses the overall architecture. Section 3 outlines how service capabilities and user request are represented. Section 4 discusses the service composition component of our architecture. Section 5 concentrates on the execution of the ad-hoc processes by transactional (mobile) agents. Finally, Section 6 concludes this paper.

2 Sketch of the System Architecture

This section very briefly presents the overall architecture of our work and illustrates the interplay of the different components. Figure 1 contains peers on the left side, which are the service providers. For example, there might be peers providing services for reserving tickets for a cinema movie. On the right side, there are the users with their requirements. In between is our planning and execution infrastructure, which fulfills the user requirements by invoking services within the context of ad-hoc processes.

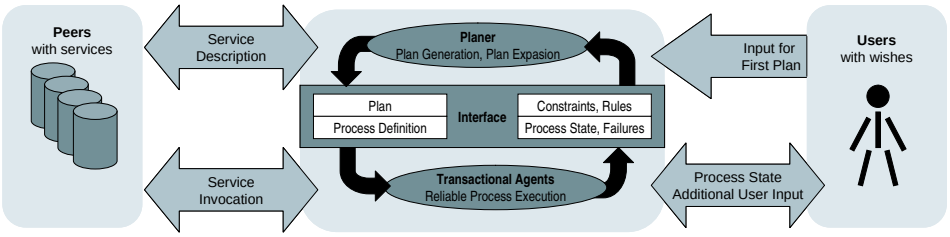

Fig. 1. Overall Architecture

The users have to inform the planner about their initial goals such that the planner can compose a first plan. This composition is based on the user requirements and on the services provided and described by the peers. The result of the composition is a plan, which can also be seen as the definition of a process to be executed in the system. The process definition is compiled into a transactional agent. This agent is responsible for the reliable execution of the process. During the execution the transactional agent invokes services on peers or presents results to the user and obtains new information from him. However, reliable execution not necessarily implies that the process always performs in the intended way. It can also mean that failures appear which of course must be detected. The process has to roll back until it reaches an expansion step which invokes the planner. Then, the planner modifies the plan. For this task, the planner can access the process state and the failure description, which can also be considered as a constraint for the planner.

3 Service Descriptions and Requests

We represent service advertisements and service requests through parameters and states of the world. Parameters are either *input* or *output*, and states of the world are either *preconditions* (required states) or *effects* (generated by the execution of the service). We

presume that service description terms are defined using a class/ontological language like OWL, whereas for primitive data types a language like XSD is used. As specified by the latest version of OWL-S, in our formalism each parameter has two elements:

- A *role* describing the actual semantics of the parameter (e.g., in a travel domain the role of a parameter could be $departure$ or $arrival$).
- A *type* defining the actual data type of the parameter (e.g., the data type for both $departure$ and $arrival$ could be $location$).

We define states of the world through preconditions and effects. We extend the normal semantics of concepts that can be included in preconditions or effects such that services can achieve behaviour equivalent to standard STRIPS planning operators which in turn allows us to model compensatable transactions.

A major difference between the approach used by STRIPS and ours is that traditional planners are based on a *closed world assumption* which is not applicable in our case since by definition web services are entities acting without a central authority. Consequently, we cannot use exactly the same formalism and semantics as for STRIPS. For example, a proposition might not be satisfiable for an agent due to its lack of global knowledge. Hence, in our approach we only use explicit states of the world where negation has also to be explicit. Formally, we do not use delete lists or terms with negation but we rely on the definition of *inverse* relations between terms (antonyms).

We use *inverse* preconditions or effects for modeling semantically compensatable services. Given two services S_1 and S_2 with effects e_1 and e_2, where $e_2 \equiv \neg e_1$, we can infer that S_2 can be used to semantically compensate S_1, since applying S_2 will actually undo the effects of S_1. E.g., in a banking system the states $credit$ and $debit$ could be defined such that $credit(x) \equiv \neg debit(x)$. Thus, a service that has an effect $debit(1000)$ can be compensated by calling another service with the effect $credit(1000)$. For details on the semantics of service advertisements and service request please see [2].

4 Service Composition with Partial Type Matches

The composition algorithms presented here take as input a request, consisting of a set of provided input parameters, a set of required output parameters, initial preconditions, and desired effects. The composition algorithm interacts with a service directory in order to retrieve relevant service descriptions. If a given service composition problem can be solved, the algorithm returns the workflow of a composed service.

Frequently, provided outputs and required inputs don't have complete compatibility because the type of a required service input may not cover the full range of possible request inputs. However, there may be multiple services that together cover the full range. To address this, we have designed a service composition algorithm based on forward chaining, which we call *forward chaining with partial type matches*. In this algorithm we do not require the range of a service input parameter to cover the full range of an available request input parameter. For a service to be considered, it is sufficient that for each required service input there is a parameter with matching roles and overlapping type provided by the request. We still require that all preconditions of s are fulfilled by r. This kind of matching between the types of the inputs of request r and of service s corresponds to the *overlap* or *intersection* match [5, 1].

Service composition with partial type matches combines several partially matching services into a composite service using a *switch* that maps each possible combination of parameter values from the space of available request inputs to one or more partially

matching services. To be able to fulfill the same functionality as a completely matching service, there has to be at least one service for each range combination of input parameters that accepts these parameter values. The switch corresponds to a non-deterministic planning operator: The choice point that it introduces allows for a number of possible service invocation paths to be followed without commitment to a particular one during service composition. The choice will be made at runtime during the executing of a service composition plan based on the input parameter values of the switch. Each of the branches in a switch provides a (possibly different) set of available parameters.

Experimental results carried out in various domains show that using partial matches decreases the failure rate by up to 7 times compared with an integration algorithm that supports only complete matches [2].

5 Process Execution

The definition of an ad-hoc process is quite similar to that of workflows. Ad-hoc processes are composed of a set of steps of following types:

- *Activities* are either service invocation or user interaction steps. These steps contact and/or influence the outside world. *Service invocation steps* invoke services on one or more peers, which may retrieve or manipulate data. An example is the reservation of a seat for a movie. *User interaction steps* present data to the users or allow them to give information to the process. For instance, the user can state that he does not want to see action movies.
- *Control flow steps* allow to define parallel as well as alternative execution paths. Parallel execution paths are specified by using *fork* and *join* steps.
- *Expansion steps* define situations in the process in which the execution state and failure information is delivered to the planner which in return provides the ad-hoc-process with steps by which the process is extended with. Placing user interaction steps strategically before expansion steps, the planner can use them to get information from the user.

After the user has specified an ad-hoc process and the planner has generated an initial plan, this plan is compiled into a transactional agent [4] which is executed by our AMOR prototype [3] in a peer-to-peer fashion. An transactional agent moves from peer to peer to invoke the specified services on the corresponding peers. Without relying on a centralized coordination engine, AMOR ensures globally correct execution. To realize this, each peer is equipped with a local coordination layer that maintains meta-data required by transactional agents to invoke local services. Firstly, the service repository holds information about locally available services. This allows to find concrete service instances implementing the service types specified by the planner. Secondly, the network repository manages connections to other peers. Whenever a transactional agent intends to execute a service invocation step, its coordination layer contacts the local coordination layer of the peer on which the agent currently resides on. The latter returns a reference to an appropriate peer on which the service can be executed. Therefore, each transactional agent carries its process description and manages its execution state such that its coordination layer can control the current execution context without requiring a centralized engine. For exchanging information within the process, the process coordination layer includes additional components, e.g., a *process board*. The latter manages variables of the process. Information flows between different steps of a process via this process board. Inter-process synchronization is performed by cooperation of the *process synchronization components* of the corresponding agents and peers.

When a process reaches an expansion step, it invokes the planner. An expansion step can be specified using late or post modeling. In late modeling, a plan expansion can be the last step of a regular path because it is not known how to proceed. Here, the planner cannot know at process definition time exactly what is happening afterwards (e.g., whether after the cinema a bus or a cab is taken or the user wants to go to a restaurant). Thus, the process is executed until that point, then the planner has enough knowledge to construct the next steps. In post modeling, expansion steps allow to cope with "unexpected" failures. Here, an expansion step specifies an alternative. For instance, if the reservation of the cinema fails, the process take an alternative path that the planner generates, e.g., by booking a theater ticket instead of reserving seats in cinema.

Concurrency control and recovery is based on a decentralized serialization graph. The nodes of this graph correspond to processes while the directed edges refers to conflicts occurred between these processes. The execution is correct if the corresponding serialization graph is acyclic. Traditionally, cycle checking is performed by a central coordinator. In our scenario, however, there is no such global coordinator. An important aspect of our approach is to bridge the gap between the available, local view of transactional agents and the global knowledge needed to enforce the correctness criterion. To do so, we equip each transactional agent with a local serialization graph which is kept up-to-date by communication between the transactional agents and the peers they access. A transactional agent is allowed to commit autonomously if it does not depend on another uncommitted transactional agent. In this case, the corresponding no has no incoming edge (from an uncommitted process). In order to avoid cyclic waiting situations, graph information is exchanged among dependent transactional agents.

6 Conclusion

This paper presented a novel service composition approach to complex value-added services. Our approach bundles services provided by peers in a peer-to-peer network to ad-hoc processes, which fulfill user needs. Our execution platform runs the ad-hoc processes in a completely distributed way with transactional guarantees. In case the execution engine itself cannot resolve a failure situation, it interacts with the service composition engine to find an alternative solution. Here, dynamic process expansion (redefinition) takes place. An important contribution of this paper is that it combines service composition techniques (from the A.I. community) with mechanisms for reliable ad-hoc process execution (from the database community).

References

1. Ion Constantinescu, Walter Binder, and Boi Faltings. An extensible directory enabling efficient semantic web service integration. In *3rd International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, November 2004.
2. Ion Constantinescu, Boi Faltings, and Walter Binder. Type-based composition of information services in large scale environments. In *The 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, Beijing, China, September 2004.
3. K. Haller, M. Ackermann, C. Munari, and C. Türker. Enhanced User Support for Mobile Ad-hoc Processes. In *Workshop "Grundlagen und Anwendungen mobiler Informationstechnologie", Heidelberg, 23.-24.03.2004*. Fakultät für Informatik, Universität Magdeburg, 2004.
4. K. Haller, H. Schuldt. Consistent Process Execution in Peer-to-Peer Information Systems. In *Proc. 15th Int. Conf. on Advanced Information Systems Engineering*, 2003.
5. L. Li and I. Horrocks. A Software Framework For Matchmaking Based on Semantic Web Technology. In *Proc. 12th Int. Conf. on the World Wide Web*, 2003.