Peer-to-Peer Execution of BPEL Processes

Weihai Yu

Department of Computer Science
University of Tromsø, Norway
weihai@cs.uit.no

Abstract. This paper presents a peer-to-peer approach to execution of BPEL processes. It does not pre-allocate resources as in existing decentralized approaches. Nor does it involve global coordination for normal executions. The approach is of continuation-passing style, where continuations, or the reminder of executions, are passed along with messages for process execution. Two continuations are associated with an execution: a success continuation and a failure continuation. Recovery plans for processes are automatically generated at runtime and attached to failure continuations.

1 Introduction

WS-BEPL [8], or simply BPEL, is becoming a de-facto standard for services composition based on the workflow technology. Using BPEL, a composite service is a BPEL process that uses other services (processes) in some prescribed order. Today, executions of BPEL processes are typically conducted by heavyweight central engines. The cost of deploying a central engine is usually too high for a large number of small businesses or end-users. Moreover, a central engine can become a potential processing and communication bottleneck as well as a central point of failure [1].

Several decentralized or peer-to-peer approaches have been proposed [2][3][4][5][6][7][9]. Common to most of these, a process is instantiated prior to its execution. During instantiation, proper resources and control are pre-allocated in the distributed environment. These approaches inevitably allocate resources even for the parts that are not executed. They also tend to have limited adaptability at runtime due to the complication of re-allocating the pre-allocated resources and control.

Our peer-to-peer approach does not involve process instantiation before execution. The approach is of continuation-passing style, which is a common practice in the functional programming community. Basically, a continuation represents the rest of an execution at a certain point of the execution. It is automatically derived during the execution. By knowing the continuation of the current execution, the control can be passed to the proper subsequent processing entities without the involvement of a central engine. In addition, the approach supports automatic process recovery by associating two continuations with any particular point of execution. The success continuation represents the path of execution towards the successful completion of the process. The failure continuation represents the path of execution towards the proper compensation of committed effects after certain failure events.

2 BPEL Processes

In BPEL, processes and (composite) services are synonymous. A (sub-) process is an *activity*, which has a hierarchical structure. Basic activities include empty activities, activities for providing and invoking services, cast of fault events, etc. A structured activity consists of a collection of activities to be executed in some prescribed order, such as in sequence and in parallel.

Activities run within *scopes*, which provide boundaries for fault handling and recovery. A process instance runs within a top-level scope. A scope can be associated with a number of activities, including a number of event handlers and fault handlers, an optional compensation handler, and a primary activity that defines its normal behavior. Within a scope, a fault can be thrown with a fault name. The fault will be captured by the scope and handled with a corresponding fault handler. A fault handler typically contains a compensation activity, which executes the compensation operations currently installed within the scope.

3 Continuation-Passing Messaging for Process Execution

Basically, a message tells a site what to do next. If a message also contains a continuation, the site can figure out the execution plan that follows up. In our approach, conducting the execution of a process is the sequences of sending and interpreting messages that contain continuations.

There are some specific issues to be addressed for peer-to-peer process execution using continuations.

1. A partially executed process must be rolled back if some fault event occurs. To enable process rollbacks, two continuations are associated with any particular execution point. The *success* continuation represents the path of execution towards the successful completion of the process. The *failure* continuation represents the path of execution towards the proper compensation of committed effects after certain failure events.
2. Some management tasks, such as proper handling of parallel branches and termination of scopes, must be carried out by the proper sites at proper time. The management tasks are defined as *auxiliary activities* that are automatically added into continuations during execution.
3. Parallel branches must be kept track of in order to: (a) stop all branches of a scope when the scope terminates, and (b) stop and rollback all branches of a scope in case of some fault event. This is done with *scope agents*.

More specifically, a message for process execution contains a control activity and two continuations. The *control* activity is the activity to be executed immediately. It is either a BPEL activity or an auxiliary activity. One of the two continuations, the success or the failure continuation, is to be executed after the control activity. A continuation is represented as a stack of activities.

The local architecture at a site is shown in Figure 1. Requests for process executions at the site are delivered to its message queue (1). A process interpreter is a pool of threads that interpret the messages. A thread dequeues a message from the message queue (2) and decides the next action according to the control activity of the message. There are two possibilities here: either can the process move on with local processing, or it is dependent on some other messages that are not available yet, such as a provided service waiting for an incoming invoking message. In the former case, the thread invokes (3, 4) some local programs, which might interact with human users. In the latter case, the current message is put in the pending message pool (5). This message will be used later (6) when a dependent message is available (2 again). After the execution of local programs, new messages are either put in the pending message pool (5) or sent to a remote site (7).

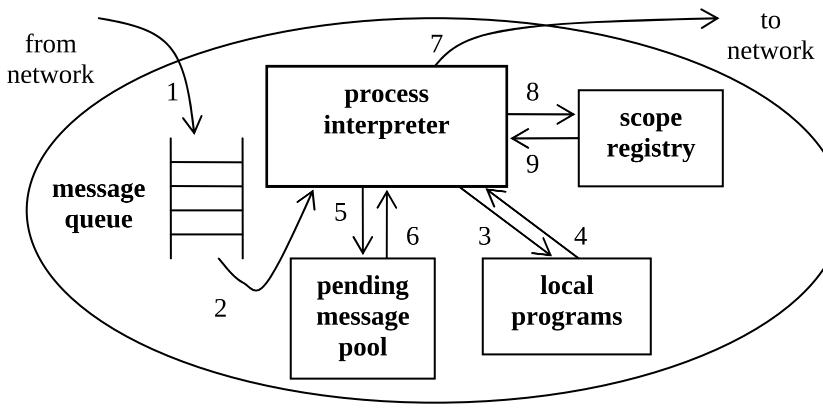

Figure 1. Local architecture of a site

If a site is also a scope agent, it maintains the scope state in the scope registry (8, 9). Basically, the scope state contains the current locations of all active parallel branches. The location of a branch changes when a message is sent. To keep this location state up to date, when a site sends a message (7), it also notifies the management agent of the immediate enclosing scope. To terminate a scope, the scope agent asks all registered sites to stop and clear up the corresponding local activities. To rollback a scope, all these sties run the respective parts of failure continuations.

4 Discussions and Related Work

One important property of our approach is that the next step a site should take is only dependent on local information. No global coordination is needed. Moreover, housekeeping of runtime states is limited within scopes. This inherently addresses the scalability and reliability problems of centralized approaches [1].

Another important property of our approach is that it allows for dynamic invocation of services and just-in-time distribution of resources. That is, the resources can be allocated when the services are to be executed. This is contrary to most other decentralized approaches [3][4][5][6][7][9] where resources for the entire process are pre-allocated during process instantiation a prior to their executions. INCA [2] goes a step further than most of the decentralized approaches. A message (called information carrier, INCA) contains a log of the execution so far and rules for further enactment. The rules and the log thus play the roles similar to success and failure continuations of our approach. INCA does not totally eliminate pre-allocation of resources: workflows are enacted using both rules carried in messages and pre-installed local rules.

This paper extends our earlier work [10] by dealing with features specific to BPEL processes, including dependencies between parallel branches and scope management.

5 Conclusion

Our contribution is a new peer-to-peer approach to execution of BPEL processes. The continuation-passing style of the approach makes the conduction of process execution as local operations rather than global coordination. As a distinct feature, it does not unnecessarily pre-allocate resources prior to process executions. Furthermore, it supports process recovery by automatically generating recovery plans into failure continuations. The approach is supported by a working prototype.

6 References

[1] Alonso, G., A. Agrawal, A. El Abbadi, C. Mohan, "Functionality and Limitations of Current Workflow Management Systems", *IEEE Expert 12(5)*, 1997.

[2] Barbara, D., S. Mehrotra and M. Rusinkiewicz, "INCAs: Managing Dynamic Workflows in Distributed Environments", *Journal of Database Management, Special Issues on Multidadatabases, 7(1),* 1996.

[3] Benatallah, B., M. Dumas and Q. Z. Sheng, "Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services", *Distributed and Parallel Databases*, 17(1), pp 5-37, 2005.

[4] Chafle, G., S. Chandra and V. Mann, "Decentralized Orchestration of Composite Web Services", *13th international World Wide Web conference (Alternate track papers & posters)*, pp 134-143, May, 2004.

[5] Gokkoca, E., M. Altinel, I. Cingil, N. Tatbul, P. Koksal, and A. Dogac, "Design and Implementation of a Distributed Workflow Enactment Service", *2nd IFCIS International Conference on Cooperative Information Systems (CoopIS 97)*, pp. 89-98, June, 1997.

[6] Marazakis, M., D. Papadakis and C. Nikolaou, "Aurora: An Architecture for Dynamic and Adaptive Work Sessions in Open Environments", *9th International Conference on Database and Expert Systems Applications (DEXA 98)*, pp. 480-491, LNCS 1460 Springer-Verlag, August, 1998.

[7] Muth, P., D. Wodtke, J. Weißenfels, A. K. Dittrich and G. Weikum, "From Centralized Workflow Specification to Distributed Workflow Execution", *Journal of Intelligent Information Systems*, 10(2), pp 159-184, 1998.

[8] WS-BPEL, *Web Services Business Process Execution Language Version 2.0*, public review draft, QISIS Open http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf, August 23, 2006.

[9] Yan, J., Y. Yang, and G. Raikundalia, "Enacting Business processes in a Decentralised Environment with p2p-Based Workflow Support", *4th International Conference on Web-Age Information Management (WAIM 03)*, LNCS 2762, pp 290-297, Springer-Verlag, September, 2003.

[10] Yu, W. and J. Yang, "Continuation-Passing Enactment of Distributed Recoverable Workflows", *22nd Annual ACM Symposium on Applied Computing (SAC 2007)*, Seoul, Korea, March 11 - 15, 2007.