An Approach to Adaptive Distributed Execution Monitoring for Workflows in Service-based Systems

Stephen S. Yau, Dazhi Huang and Luping Zhu
Department of Computer Science and Engineering, School of Computing and Informatics
Arizona State University, Tempe, AZ 85287-8809, USA
{yau, dazhi.huang}@asu.edu

Abstract

Systems based on Service-Oriented Architecture are called service-based systems (SBS), and comprise of computing services offered by various organizations. Users of SBS often require these services to be composed into complex workflows to perform their high-level tasks. The users usually have certain expectations on the overall QoS of their workflows. Due to the highly dynamic environments of SBS, in which temporary unavailability or quality-degradation of services may occur frequently and unexpectedly, monitoring the execution of workflows in SBS is necessary, and should be done in distributed and proactive manner. In this paper, a virtual machine-based architecture for the execution, monitoring and control of workflows in SBS is presented. Based on this architecture, an approach to automated generation of workflow monitors for adaptive distributed execution monitoring of workflows in SBS is discussed

Keywords: Service-based systems, workflow, adaptive execution monitoring, virtual machine-based architecture, and automated generation.

1. Introduction

Due to the development of Service-Oriented Architecture (SOA) that enables rapid composition and integration of large-scale distributed systems, many critical systems, such as Grid, Global Information Grid (GIG), and healthcare information systems, have adopted SOA as their system architecture for cost-effective distributed applications with more flexibility and better service quality. Systems based on SOA are called *service-based systems (SBS)*, and often comprise of computing services offered by various organizations. These computing services provide well-defined interfaces for users to access certain capabilities offered by various providers, and are often hosted on geographically-dispersed computer systems. Due to different system capacities, active workloads or service contracts with users, such services usually provide various QoS levels, such as different service delays and security protections. A user of SBS will discover and access the most suitable services, which not only provide the required functionality, but also meet the user's expected QoS.

Besides direct access to an individual service, a user often wants to carry out a workflow composing of various services in SBS to perform a high-level task that cannot be done by an individual service. The composition of services can be automated based on the required functionality, referred to as the *goal*, of the entire workflow [1-3]. However, similar to the use of an individual service, the user of a workflow often has certain QoS expectations on the entire workflow, such as a deadline for completing the workflow. Although some research has been done to incorporate certain QoS aspects in service composition [4, 5], the execution of workflows in SBS may not satisfy users' requirements due to the highly dynamic environments of SBS, in which temporary unavailability or quality-degradation of services may occur unexpectedly.

From software cybernetics perspective [6, 7], such a problem can be tackled by constructing a close-loop control system to monitor the workflow executions, and dynamically control the selection and configuration of services in the workflows to meet users' requirements. However, the distributed and loosely-coupled nature of SBS has created many challenges to make SBS effectively usable

First, proactive execution monitoring support is needed to capture and report various problems affecting the execution of workflows in SBS in a timely manner. Such execution monitoring support needs to proactively monitor the computing resources used and detect any problems that may affect the execution of workflows. For example, if a service to be used in a workflow fails, it will allow more time to find a replacement for the failed service and reconfigure the corresponding workflow if the failure can be detected before an attempt to invoke the service is made. This would greatly increase the chance of completing the workflow satisfactorily. Secondly, distributed execution monitoring support is needed to overcome the inefficiency and potential security or dependability problems of centralized monitoring approaches.

In this paper, we will first present a virtual machine-based architecture for executing, monitoring and controlling workflows in SBS. For a workflow in SBS, three types of distributed software components running on our virtual machines will be generated to execute, monitor and control the adaptation of the workflow respectively. We will also discuss how to automatically generate adaptive distributed execution monitoring components based on specifications of both functional and non-functional requirements of workflows, and how these components can be coordinated in runtime through the underlying virtual machines to acquire accurate information on workflow executions with small overheads.

2. Current State of the Art

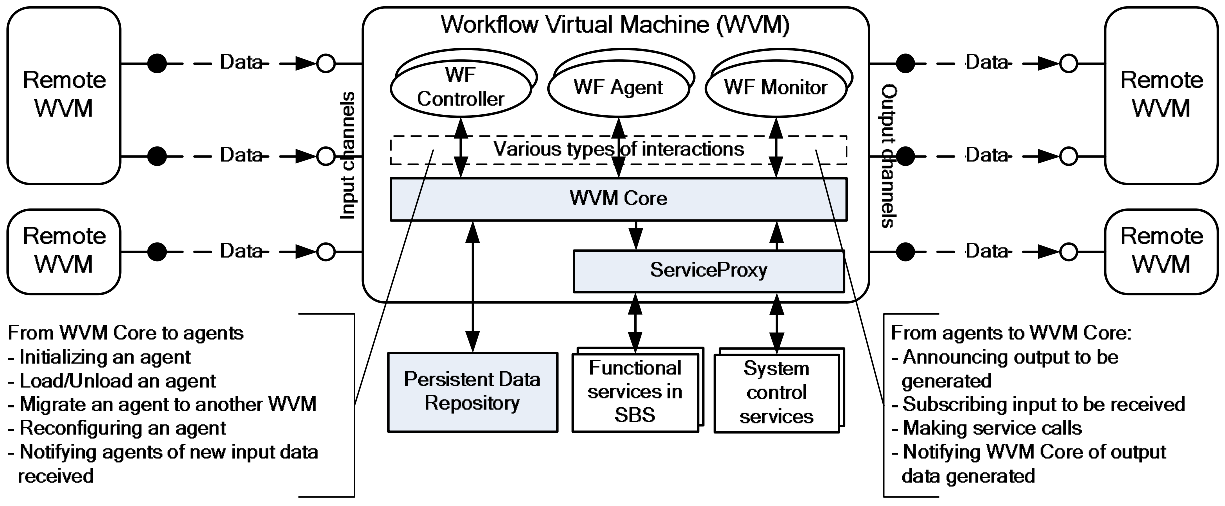
Execution monitoring of workflows has been considered as an integral part of workflow management systems. Recently, substantial research has been done on Grid workflow systems [8-12], which operate in similar environments of SBS.

DAGMan [8] provides a limited query capability for checking job status, which requires additional programming effort if some application-specific monitoring needs to be done. Gridbus [9] monitors workflow execution status based on event notification, and uses tuple spaces to exchange various event tuples indicating the execution status of distributed jobs. GridFlow [10] provides workflow execution and monitoring capabilities based on a hierarchical agent-based resource management system (ARMS) [13] for Grid, in which the agents serve as representatives of resources and are able to record performance data of the resources. Kepler [11] provides an easy-to-use environment for the design, execution and monitoring of scientific workflows. It relies upon specialized actors, which are interfaces to services on Grid, to provide job submission, execution and monitoring capabilities. Pegasus [12] has a Concrete Workflow Generator to automate the creation of a workflow in the form of DAG (Directed Acyclic Graph), and uses systems like DAGMan to execute and monitor workflows.

The workflow monitoring capabilities in these Grid workflow systems, as well as in many other workflow systems, share a similar characteristic: They only focus on the status of the current workflow execution, and not proactively acquiring and analyzing status information related to future workflow execution.

3. Virtual Machine-based Architecture

Before describing our overall approach to adaptive distributed execution monitoring of workflows in SBS, we will first introduce our virtual machine-based architecture for executing, monitoring and controlling workflows in SBS shown in Figure 1. In our approach, workflows in SBS are executed by workflow (WF) agents, monitored by WF monitors, and controlled by WF controllers. WF agents, WF monitors and WF controllers are specific types of software agents which can run on top of our Workflow Virtual Machine (WVM). Our WVM is an



Workflow Virtual Machine (WVM)
Remote WVM
Data
Input channels
Output channels
WF Controller
WF Agent
WF Monitor
Various types of interactions
WVM Core
ServiceProxy
Persistent Data Repository
Functional services in SBS
System control services
From WVM Core to agents
- Initializing an agent
- Load/Unload an agent
- Migrate an agent to another WVM
- Reconfiguring an agent
- Notifying agents of new input data received
From agents to WVM Core:
- Announcing output to be generated
- Subscribing input to be received
- Making service calls
- Notifying WVM Core of output data generated

Figure 1. Our virtual machine-based architecture for executing, monitoring and controlling workflows in SBS.

extension of SINS (Secure Infrastructure for Networked Systems) Virtual Machine [14]. We have presented approaches to composing, scheduling and deploying situation-aware workflows in SBS involving WF agents and WF controllers [15 – 19]. In this paper, we will discuss how workflow execution monitoring is supported by our virtual machine-based architecture, and how distributed WF monitors for a workflow are automatically generated.

Our WVM has a set of input/output channels, uniquely identified by their names. WVMs communicate with each other through these named channels to exchange data for coordinating activities of agents running on the WVMs. The communication over these named channels is supported by SPREAD [20], which is a reliable group communication package. Each software agent running on a WVM will take certain input data received by the WVM, and send the generated data, if any, through the WVM. Some specific named channels are reserved for exchanging messages generated by virtual machines, such as load/unload messages for an agent.

Like SINS, our WVM provides the capabilities for loading and unloading agents, accepting an agent's announcement on the input data required by the agent and the output data that will be generated by the agent, sending and receiving the data through the named channels, and notifying the agents when new input data is available.

In addition, our WVM provides the following new capabilities:

- Parameterized initialization of agents, which subsequently enables runtime reconfiguration of agents.
- Invocation of functional services in SBS and system control services, such as system performance packages, network management and migration, to support workflow execution, monitoring and control in SBS.
- Event logging into a persistent data repository for recording workflow execution history.

The WVM Core is mainly responsible for these new capabilities. The ServiceProxy in the WVM provides an interface for each functional service or system control service in SBS. When an agent (may be a WF agent, WF monitor or WF controller) evaluates its input data and makes a decision to invoke a certain service, the WVM handles the service invocation as follows:

i) The agent generates a service invocation request with the necessary information, such as the name of the service to be invoked and the identity of the user invoking this service if access control for the service is required. Specifically, the name of the input channel, where the agent expects to receive the result of this service invocation, is included in the service request.
ii) The agent puts the request into an output channel named *ServiceInvoke*.
iii) Instead of sending out the request in the *ServiceInvoke* channel, the WVM Core parses the service request, performs security checking, if necessary, and invokes the corresponding interface in the ServiceProxy.
iv) Once invoked, the corresponding interface in the ServiceProxy constructs an appropriate service request or API call, depending on the form of the service to be invoked. (For example, if the service to be invoked is in the form of Web Service, a SOAP message will be constructed.)
v) When the ServiceProxy receives the result of the service invocation, it puts the result into the input channel specified by the agent in (i).

The above service invocation process in the WVM not only provides agents the capability to make asynchronous service invocations and perform other tasks in parallel, but also provides richer information regarding service invocations, which will be very useful in our workflow execution monitoring approach. For example, service delay can be easily measured in the ServiceProxy to capture more semantics-rich events, not just the information on success or failure.

Our virtual machine-based architecture provides the necessary capabilities for WF monitors to perform execution monitoring, subscribe to important events, such as status of service invocations, be notified by WVMs when such events occur and invoke necessary system control services without knowing the low-level details of such services.

4. Adaptive Distributed Execution Monitoring of Workflows in SBS

An important difference between our approach to workflow execution monitoring and those in [8-12] is that the WF monitors in our approach will proactively acquire the status information of computing resources to be used in future workflow execution, so that the problems which may affect future workflow execution can be captured earlier. However, such proactive monitoring approach will have larger overhead since more resources need to be monitored at the same time. Such overhead, if not carefully handled, may negate the possible benefits in terms of the increasing number of successfully completed workflow executions.

To address this problem, we develop an adaptive execution monitoring approach, which follows the similar philosophy of adaptive sampling designs of experiments [21]. Adaptive sampling designs, also known as response-adaptive designs, are statistical experiment designs, where the experimental designs will be adapted based on data collected from previous experiments to improve the quality of experiments.

Applying this idea to our adaptive execution monitoring approach, our WF monitors will not only monitor all the resources to be used in future execution of a workflow at the beginning of the workflow. Instead, our WF monitors will first start to monitor a subset of resources. When any problem occurs, WF monitors will reconfigure themselves or load new monitors to acquire more information to create a clearer view of system status.

Our approach is based on the observation that many problems in networked computing systems have subtle logical connections among them. For example, a sudden slowdown of downloading a large file from a remote host may indicate a problem with the network connection to the remote host, or a problem with the remote host itself. Either problem may further lead to failures for accessing other resources at the same host or using the same connection. Hence, for workflow execution monitoring, it is possible to select a subset of resources to be used in the workflow as the *indicators*, whose problems often lead to or co-occur with problems of other resources. When problems with the indicators occur, WF monitors should proactively check the status of those resources related to the indicators. The benefit of our approach is even more in a large-scale SBS, where many concurrent workflows are being executed simultaneously. In such a system, problems occurred in the execution of one workflow may indicate that related problems may occur in the execution of other workflows.

Obviously, our adaptive execution monitoring approach takes advantage of knowledge on the relationship between possible problems that may occur during workflow execution to adjust monitoring tasks performed by WF monitors. Such an approach requires specific knowledge on failures in the execution environment of the SBS and the workflow to be incorporated in the design of WF monitors. Manually developing such WF monitors in a large-scale SBS is not feasible due to the possible large development cost. Hence, a tool for automatically generating such WF monitors is needed.

Figure 2 shows our process of automated generation of WF monitors. This process is based on our previous work on composing, scheduling and deploying workflows in SBS [16, 18, 19]. We assume that before the process of generating our WF monitors starts, a workflow has been composed [16] or manually defined by developers, and the initial schedule, resource assignment and deployment plan for this workflow have been generated [18, 19]. We generate a WF monitor for each resource to be used in the workflow.

Our ideas on automated generation of WF monitors can be summarized as follows:

1) Monitoring target identification. Resources to be used in the workflow execution and constraints on the usage of these resources, which mainly include the duration of time that the resources will be used and the amount of resources required, will be extracted from the schedule and resource assignments. Various events indicating the progress of the workflow will be extracted from the workflow specification. These resources, constraints, and events are the targets of monitoring. Specifically, the

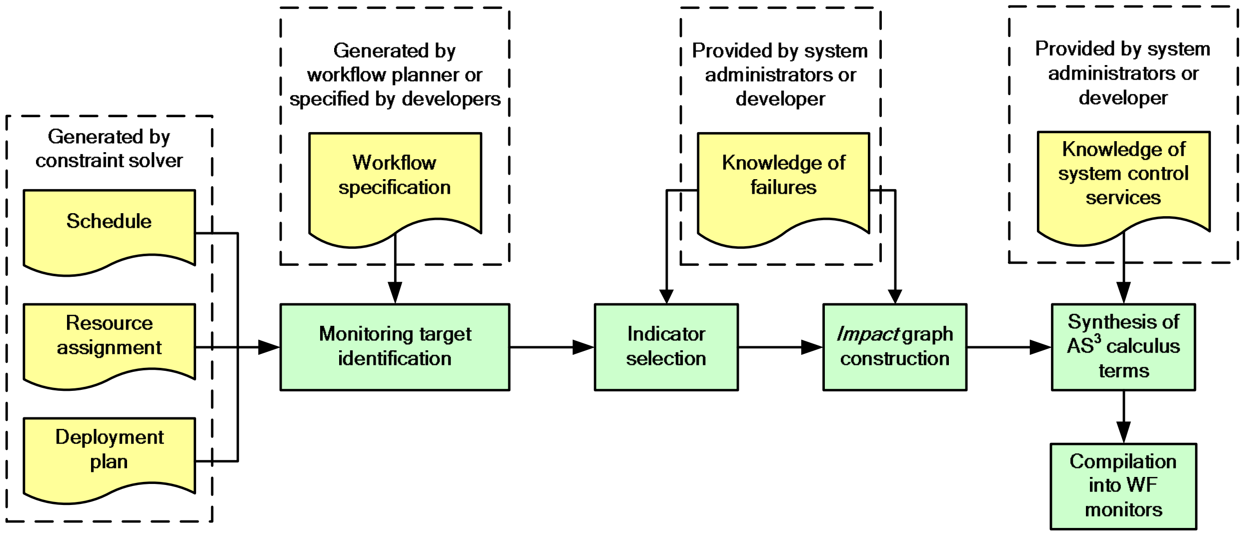

Generated by constraint solver
Schedule
Resource assignment
Deployment plan
Generated by workflow planner or specified by developers
Workflow specification
Monitoring target identification
Indicator selection
Provided by system administrators or developer
Knowledge of failures
Impact graph construction
Provided by system administrators or developer
Knowledge of system control services
Synthesis of AS^3 calculus terms
Compilation into WF monitors

Figure 2. Automated generation of WF monitors for adaptive distributed execution monitoring of workflows in SBS.

identified constraints and events will be associated with the corresponding resources to simplify the following step (Step 2).

2) Indicator selection. To select an appropriate subset of resources as the indicators, certain knowledge of possible failures in the execution environment, including the possible failures which may occur during workflow execution, the causes of the failures, and the relations between different failures, is needed. Such knowledge can be provided by human experts, such as system administrators or developers, or can be obtained through data-mining on history data of failures. Such knowledge needs to be specified in a machine-understandable way so that automated reasoning on the knowledge can be performed. In our approach, AS^3 logic [15-17] is used to specify the temporal and causal relations between failures. (For example, a simple rule "$A \rightarrow \Diamond B$" in AS^3 logic states that the occurrence of an event *A* may lead to *B*.) Based on the specifications of such knowledge, we can construct a graph with all the resources to be monitored as the vertices of the graph. A directed edge is added from a resource R_1 to another resource R_2 if there is a rule in the specifications indicating that a problem of R_1 may lead to or co-occur with a problem of R_2. Thus, a candidate set of indicators is the set of nodes, whose outgoing edges cover all the nodes in the graph.

For a resource not in the selected set of indicators, its corresponding WF monitor will not be loaded unless it is loaded by another WF monitor or the current workflow execution is using this resource. If a resource belongs to the selected set of indicators, the WF monitor for this resource will be loaded even when the current workflow execution has not yet required this resource.

3) Impact graph construction. After the set of indicators is selected, an *impact* graph for each indicator will be constructed. The *impact* graph for an indicator is a subgraph of the graph constructed in Step 2). This subgraph contains the indicator, and all other nodes which have at least one incoming edge from the indicator. Hence, an *impact* graph for an indicator essentially contains the information about what other resources may have failures if the WF monitor of the indicator detects a problem of the indicator occurs. Such information will be used in Step 4) to synthesize appropriate AS^3 calculus terms for the WF monitor for the indicator.

4) Synthesis of AS^3 calculus terms for WF monitors. Similar to our automated agent synthesis approach for situation awareness in SBS, various monitoring activities to be performed by WF monitors can be described using AS^3 calculus [15-17], which is based on classical process calculus. AS^3 calculus can model timeouts, failures, service invocations, and communications. In particular, it can describe input and output actions on named channels and migration of processes between different ambients (as in the ambient calculus [22]). As discussed in Section 3, our WF monitors will utilize system control services, such as system performance packages, to acquire the status information of various resources. To facilitate the synthesis of WF monitors, these system control services are specified using AS^3 logic in the same way as functional services used by the workflows in SBS [15, 16]. Such logical specifications allow the discovery of appropriate system control services through automated reasoning on the service specifications according to the monitoring targets identified in Step 1). The constraints on the resources to be monitored are encoded as conditional evaluations in AS^3 calculus. Information on various events, such as the detection of a failure of a certain resource, is exchanged through named channels. The names of all the named channels in different WF monitors are generated following the same naming standard to ensure WF monitors to communicate with each other correctly.

5) Compilation of AS^3 calculus terms into executable code for WF monitors to be run on our WVM. An AS^3 calculus to Java compiler has been developed to generate Java code for an executable agent running on our WVMs from the AS^3 calculus terms describing the behavior of this agent [16, 17].

5. Conclusions and Future Work

In this paper, we have presented the virtual machine-based architecture, on which distributed WF agents, monitors and controllers coordinate to execute, monitor and control the adaptation of workflows in SBS. We have also presented some ideas on the automated generation of distributed WF monitors based on workflow specifications and knowledge of the execution environment. In our approach, the generated WF monitors for a workflow proactively acquire status of resources to be used to provide early detection of problems which will likely affect future workflow execution. If a problem in a resource to be used is detected by a WF monitor, the monitor will collect more information about the problem by dynamically loading and activating new monitors. In this way, our WF monitors allow more time and provide more accurate information for workflow adaptation. Currently, a prototype of our WVM (workflow virtual machine) has been developed on SINS [14]. We have also manually implemented a WF monitor in a demonstration system to illustrate our ideas. Currently, we are developing the generator of WF monitors. Future work includes performing simulations and experiments to examine the overhead and benefits of our approach and compare our approach with other existing approaches.

Acknowledgment

This research is supported by the DoD/ONR under the MURI Program Contract No. N00014-04-1-0723, and National Science Foundation under grant number CNS-0524736

References

[1] S. Ponnekanti and A. Fox, "Sword: A developer toolkit for web service composition", *11th Int'l World Wide Web Conf. (WWW 2002) Web Engineering Track*, 2002. Available at: http://www2002.org/CDROM/alternate/786/

[2] J. Rao, P. Kungas and M. Matskin, "Application of linear logic to web service composition", *Proc. 1st Int'l Conf. on Web Services*, 2003, pp. 3-9.

[3] E. Sirin, J. A. Hendler and B. Parsia, "Semi-automatic composition of web services using semantic descriptions", *Proc. Web Services: Modeling, Architecture and Infrastructure (WSMAI) Workshop* in conjunction with the 5th Int'l Conf. on Enterprise Information Systems (ICEIS 2003), 2003, pp. 17-24.

[4] X. T. Nguyen, R. Kowalczyk, and M. T. Phan, "Modeling and solving QoS composition problem using fuzzy DisCSP", *Proc. 2006 IEEE Int'l Conf. on Web Services (ICWS 2006)*, 2006, pp. 55-62.

[5] R. Berbner, et al., "Heuristics for QoS-aware web service composition", *Proc. 2006 IEEE Int'l Conf. on Web Services (ICWS 2006)*, 2006, pp. 72-82.

[6] Kai-Yuan Cai, T. Y. Chen, and T. H. Tse, "Towards Research on Software Cybernetics", *Proc. 7th IEEE Int'l Symp. on High Assurance Systems Engineering (HASE'02)*, 2002, pp 240-241.

[7] Kai-Yuan Cai, J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "An Overview of Software Cybernetics", *Proc. 11th Annual Int'l Workshop on Software Technology and Engineering Practice*, 2004, pp 77-86.

[8] T. Tannenbaum, D. Wright, K. Miller and M. Livny, "Condor – a distributed job scheduler", in *Beowulf Cluster Computing with Linux,* T. Sterling (eds.), MIT press, 2002. Available at: http://www.cs.wisc.edu/condor/doc/beowulf-chapter-rev1.pdf

[9] J. Yu and R. Buyya, "A novel architecture for realizing Grid workflow using tuple spaces", *Proc. 5th IEEE/ACM Int'l Workshop on Grid Computing (GRID 2004)*, 2004, pp. 119-128.

[10] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: workflow management for Grid computing", *Proc. 3rd Int'l Symp. on Cluster Computing and the Grid (CCGrid 2003)*, 2003, pp. 198-205.

[11] B. Ludäscher, et al., "Scientific Workflow Management and the Kepler System", *Concurrency and Computation: Practice & Experience*, vol. 18(10), 2006, pp. 1039-1065.

[12] E. Deelman, et al., "Mapping abstract complex workflows onto Grid environments", *Jour. Grid Computing*, vol. 1(1), 2003, pp. 25-39.

[13] J. Cao, et al., "ARMS: an Agent-based Resource Management System for Grid Computing", *Scientific Programming*, vol. 10(2), 2002, pp. 135-148.

[14] R. Bharadwaj, "Secure middleware for situation-aware naval C^2 and combat systems," *Proc. 9th Int'l Workshop on Future Trends of Distributed Computing System (FTDCS'03)*, 2003, pp. 233-240.

[15] S. S. Yau, et al., "Automated Agent Synthesis for Situation-Aware Service Coordination in Service-based Systems", technical report, Arizona State University, 2005, available at: http://dpse.eas.asu.edu/as3/papers/ASU-CSE-TR-05-009.pdf.

[16] S. S. Yau, et al., "Automated Situation-aware Service Composition in Service-Oriented Computing", *Int'l Jour. on Web Services Research (IJWSR)* , to appear.

[17] S. S. Yau, et al., "Automated Agent Synthesis for Situation Awareness in Service-based Systems", *Proc. 30th Annual Int'l Computer Software and Application Conf. (COMPSAC 2006)*, 2006, pp. 503 - 510.

[18] S. S. Yau, D. Huang, L. Zhu and Kai-Yuan Cai, "A Software Cybernetic Approach to Deploying and Scheduling Workflow Applications in Service-based Systems", *Proc. 11th Int'l Workshop on Future Trends of Distributed Computing Systems (FTDCS 2007)*, 2007, pp. 149 - 156.

[19] S. S. Yau, L. Zhu, D. Huang and H. Gong, "An approach to Automated Agent Deployment in Service-based Systems", *Proc. 10th IEEE Symp. on Object-oriented Real-time distributed Computing (ISORC 2007)*, 2007, pp. 257-264.

[20] Y. Amir, C. Nita-Rotaru, J. Stanton, and G. Tsudik, "Secure Spread: An Integrated Architecture for Secure Group Communication", *IEEE Trans. on Dependable and Secure Computing*, vol. 2(3), 2005, pp. 248-261.

[21] J. Hardwick and Q. F. Stout, "Flexible algorithms for creating and analyzing adaptive sampling procedures", *New Developments and Applications in Experimental Design*, vol. 34, 1998, pp. 91-105.

[22] L. Cardelli and A. D. Gordon, "Mobile Ambients," *Theoretical Computer Science*, vol. 240(1), 2000, pp. 177-213.