Autonomic Business Processes Scalable Architecture

José A. Rodrigues Nt.[1], Pedro C.L. Monteiro Jr.[1], Jonice de O. Sampaio[1],

Jano M. de Souza[1,2], Geraldo Zimbrão[1]

[1] COPPE/UFRJ, Graduate School of Engineering

[2]DCC-IM Dept. of Computer Science, Institute of Mathematics

Federal University of Rio de Janeiro, Brazil

PO Box 68.511 – Zip code: 21945-970 - Rio de Janeiro - Brazil

{rneto, calisto, jonice, jano, zimbrao}@cos.ufrj.br

Abstract. Organizations have to face new challenges, hold new opportunities, conquer and maintain important customers and get a better strategic position as faster as possible. The principles used in Autonomic Computing can be adapted to help them survive in this dynamic business scenario. Thus, organizations should count on processes that can be able to self-configure, self-heal, self-optimize and self-protect, i.e., self-manage and self-adapt to better answer market and organization's changes and new challenges – Autonomic Business Processes. This work proposes a multi-agent rule-based scalable architecture to provide business processes with autonomic properties, reducing the need for human intervention, and improving overall organization's response time. ***Keywords*:** Autonomic Computing, Business Process, Workflow.

1. Introduction

The world is confronted with the new knowledge-driven economy, and enterprises have to face new challenges, hold new opportunities, conquer and maintain important clients and always find a better strategic position. Processes'

dynamics are high and their impact on management unavoidable. Such a scenario calls for a systematic and dynamic approach to maintain an organization competitive.

The management suffers the impact of constant environmental changes and, subsequently, its own processes changes. Systems that can reduce the burden of constant transformations and response-time are needed. New approaches to support management ought to be developed in order to help organizations to survive.

Analyzing the Information Technology (IT) scenario, where the number and complexity of systems has grown tremendously in the past decades, it can be found valuable lessons to cope with such challenges. The Autonomic Computing approach, which appeared due to the increasing complexity of current computational solutions and, consequently, their increasing management complexity, is a good example.

Making an analogy between the new economy and new systems, firms need to have autonomic characteristics to survive in this complex and agile economy. The principles used in Autonomic Computing can be adapted to help firms survive in this new business scenario. Thus, organizations should count on processes that can be able to self-configure, self-heal, self-optimize and self-protect, i.e., self-manage and self-adapt to new challenges and market changes – Autonomic Business Processes.

A successful enterprise – to ensure an effective monitoring of progress and to have a more coherent strategic direction – should design and implement not only autonomic-computational systems, but also autonomic-business processes, autonomic-management approaches and systems to support them. Therefore, the marriage of business management needs with autonomic principles opens new opportunities.

This work proposes a multi-agent rule-based architecture to provide business processes with autonomic properties, reducing the need for human intervention, and improving overall organization's response time.

The rule-based approach, supported with multi-level blackboards, considers expert knowledge and also considers business processes under the Complex Adaptive Systems paradigm [1], in the sense that the many variables involved in processes executions and their relationships demand systems capable of presenting emergent behaviors. It promotes flexibility by adaption, as classified in [2]. Additionally, the multi-level architecture provides a flexible solution that can be scaled up to work with higher level abstractions, closer to or at an organization's strategic level, through the composition of lower level autonomic processes.

This paper is organized as follows: section 2 presents a brief summary of related work and section 3 concepts involved in the propose solution. The Autonomic Business Process Architecture is presented on section 4, an example of its use in section 5 and conclusion and future work in section 6.

2. Related Works

Most work found on autonomic business process, or autonomic workflow, focus on predictable workflows, in the sense that a baseline execution path can be defined and all alternates paths mapped; flexible but a priori defined workflow instances; or software or system oriented workflows, like in grid applications.

While the work of Savarimuthu, Purvis and Fleurke [3] deals with business process execution in a multi-agent workflow system, it neither describes how agents actually handles flexibility, nor it worries with autonomic properties.

Work done on medical workflows [4] [5] concentrate on treating exceptions and related mechanisms. While they provide a good basis for the self-healing dimension, they have no provision for self-optimization and, in minor scale, to self-configuration and self-protection. This is also the case of AgentWork [6], that concentrates on failure prediction and reaction, despite the extensiveness of the research.

Mangan and Sadiq [7], when providing flexible processes, do not focus on treating healing and protection issues and do not dedicate enough attention to real-time monitoring and reaction. Nevertheless, their analysis of processes definition and handling approaches helps on understanding the need for a non-deterministic component in our solution.

A dynamic workflow for grid environment is described in [8]. We can mention as autonomic properties dynamic configuration and reconfiguration of workflows, optimization of some behaviors to achieve a goal, recovering from failures, and optimized use of resources. However, it is important to mention that this solution just works with workflows in grids, for job execution, not business processes.

In [9], the authors propose a continuous and optimized computing environment, which updates itself according to high-level business objectives. While it considers business objectives as the driving force to process optimization, it focus the optimization efforts towards IT assets utilization.

FEEDBACKFLOW is an adaptive workflow generator for system's management [10]. This framework implements a general control loop of planning and re-planning, and generates workflows of system management actions in an adaptive manner.

Another related work is the view of a multi-agent workflow enactment as an Adaptive Workflow [11]. Although the focus of the work is workflow definition languages, it touches some important aspects closely related to our solution, as the use of multi-agent systems for coordination and the use of containers to preserve the workflow state of execution.

Web services oriented workflows are also subject of other studies. Autonomic Web Services (AWP) are web processes that support the autonomic computing properties [12]. In AWP, the processes are configured according to business policies. Failures are quickly responded and the workflow can be reconfigured due to environment changes. The work of Pautasso, Heinis and Alonso [13] about web services composition evaluates policies for composition configuration, not touching other

autonomic dimensions. On the same grounds, Pankatrius and Stucky [14] establish a formal foundation for workflow composition, instrumental to provide reconfiguration capabilities to workflow applications.

In [15] the design and performance evaluation of a dynamic workflow execution engine is presented. The system has a component to determine if the current configuration is optimal and, in the case of non-optimal configuration, it proposes an alternate execution plan. This is the closest approach to our autonomic workflow. The system contains a self-healing component to ensure that the workflow engine remains in a consistent state.

It is important to note though, that our work evolves from several aspects presented on these previously cited works and relies on some of their mechanisms for proper implementation, e.g. the formalisms proposed in [14] and [16].

3. Concepts

3.1 Autonomic Computing

The term autonomic computing originates in the human autonomic nervous system, which is responsible for managing of digestion, cardiac beating and other functions that humans do automatically, i.e. without reasoning and giving instructions. The autonomic computing paradigm aims at mimicking the human nervous system, providing systems with self-management capabilities, reducing human intervention.

Systems being developed [17] are increasingly more complex and tend to be increasingly harder to manage. This complexity can be found in architectures, networks, programming languages and applications. Autonomic computing is specially useful for this kind of systems, since, due to their complexity, the cost with human resources to keep them working can render a project impracticable.

A system need to know and understand itself to really be considered autonomic. The system have to know all its components, their current status, its operation environment, its capacity, the possible connections with others systems and the resources that it can borrow, buy or lend. The 4 basics aspects of autonomic computing are [17]:

- Self-configuring: refers to installation and activation of the system in an automated way. It makes possible the system's automatic adaptation to environment changes;

- Self-healing: the ability to discover, diagnosis and correct potential problems to ensure that the system runs smoothly;

- Self-optimizing: it treats resource monitoring and allocation to ensure that the system will be working in an optimal way; and

- Self-protecting: identification, detection and protection against numerous threats.

3.2 Agents

Agent is anything that has sensors to perceive the environment and act on it. Software agents have their perceptions and actions provided by encoded bit strings [18]. Agents can interact with other agents forming a multi-agent system.

Agents present, at least, the following properties [19]: reactive to the environment, autonomous, goal-driven and continuous execution. Agents can be classified as stationary, those that stay in a single host, or mobile agents, which can migrate and execute in multiple hosts. They move not only their codes, but also carry their context, variables, execution pointers, stack and other state variables, which are restored in the new host. The main difference between mobile agent systems and process migration is that, in the first one agents can decide when to change host, while in the second one, the system decides when process changes [19].

Scientific workflow management systems may hide the integration details among distributed environment resources, allowing scientists to prototype experiments with computational tools using a high level abstraction. Agents approach provides techniques to decompose the control intelligence of flow execution and to encapsulate distributed resources [20].

3.3 Blackboards

Blackboard is a repository style architecture where loosely coupled entities share a common knowledge space [21]. In [22], the blackboard system is divided in 3 components: the blackboard, a global data structure presenting an application dependent organization that usually holds system's state information; knowledge sources, independent entities that handle knowledge and can interact using the blackboard; and a control component, driven by the blackboard state indication and proper knowledge sources reaction.

4. Autonomic Business Process

4.1 Attribute, Fact, Condition, Action, Rule and Priority

First we describe some concepts that are central to understand the architecture: attribute, attribute value, fact, condition, action, rule and priority.

In this proposed architecture, *Attribute* is a process's characteristic which is of interest, i.e., can affect its execution. *Value* is the attribute's measurement, in other words, is the result of monitoring an attribute and can be of Date, Numeric, Text or Boolean type. A *Fact* is an observed attribute with a specific value.

In the scope of this work, attributes are organized in time, human, input, output, tool, knowledge and cost resources. However the proposed architecture supports the creation of new kinds of resources.

The table 1 shows examples of several attributes, organized by defined resources.

Table 1. Attributes by Resource Types

Resources	Attributes
Time	▪ Expected initial date ▪ Expected final date ▪ Actual initial date ▪ Actual final date ▪ Expected duration ▪ Execution time
Human	▪ Executor availability ▪ Amount of available executors
Input	▪ Required artifact present, e.g. Purchase Order arrival
Output	▪ Generated artifact, e.g. Repair Budget delivery
Tool	▪ Available necessary tool, e.g. Video Conference Equipment availability
Knowledge	▪ Engineering knowledge availability
Cost	▪ Process estimated cost ▪ Process current cost

An *Attribute* can be associated to a value through the operators equal to (=), different from (<>), greater than (>), less than (<), greater or equal to (>=) and less than or equal to (<=) forming a *condition*. As attributes have values, an atomic *condition* is a relation of the attribute itself with its domain through an operator. The concept of *composed condition* is also used and it means the union of several atomic conditions. This union is done through the use of conjunctions *AND* and *OR*. A

composed condition example is: *Executor in use = false* ***AND*** *Time in execution = 0* ***AND*** *Foreseen initial date > Current date*.

Action is an intervention on a process, that can be direct, e.g. allocation of others resources to the process, or indirect, i.e. notification of an occurrence to a different handling instance, e.g. the assertion of new facts to a blackboard or a message delivered to another system or user.

A *Rule* is constructed using conditions and actions and it has the form: ***if*** *set of conditions* ***then*** *execute actions*. For each rule a priority is defined, thus, allowing for the proper processing order, when more than one rule should be executed.

We haven't explicitly used ECA [16] nomenclature because the working characteristics of monitoring agents/expert systems lead us to think that a fact, instead of event, treatment is more appropriate, since the event itself that can affect workflow execution is usually a composition of many facts asserted by agents. This way, rule triggering will occur based on rules' priorities, composition and temporal relations.

At the same time, considering that comparing to events, facts are lower level constructs, we believe this approach is more suitable to model expert's knowledge, since it allows for a more detailed understanding of process execution and the implication of its several aspects on desired results.

4.2 System Architecture

This work proposes a system's architecture to autonomically manage an organization's business process execution, reducing the need for human intervention, and improving overall organization's response time.

The proposed architecture is shown in figure 1. Each defined activity of the target process has associated *monitor agents*, a *local blackboard* and *actuator agents*. The set of such schema over all the process' activities defines the lower level of the architecture, i.e. the level closer to the activities.

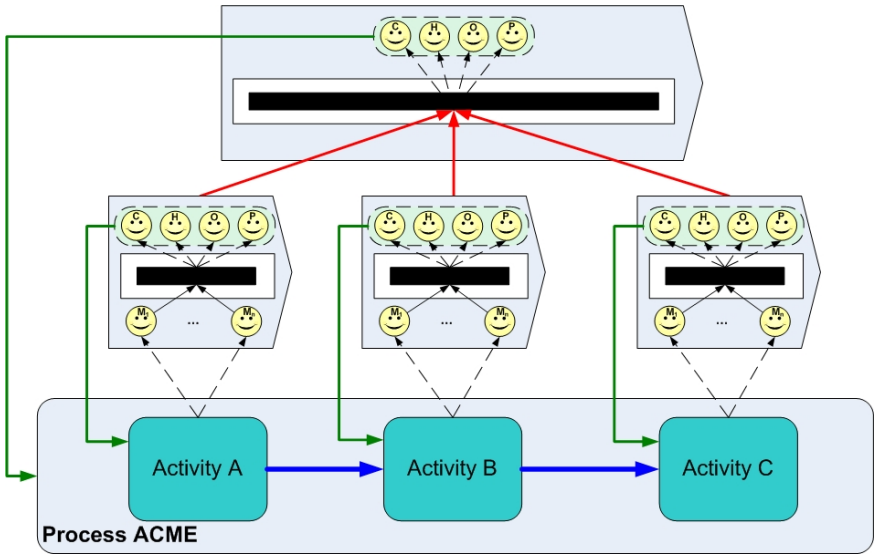

Fig. 1. System Architecture

Monitor agents, identified by the letter M inside them in figure 1, have the function of sensing the activity and writing results, as facts, onto the local *blackboard*. Each of these agents is responsible for monitoring only one resource, like the ones presented in table 1. Section 4.3 explains how user defines such agents.

Local *blackboard* then receives facts related to all resources defined for the activity.

Actuator agents are responsible for fact interpretation and action taking. This is done by rule execution. They contain rules defined by the user and work like any expert system. Actually, the main idea is to have agents implementing behaviors as processes' specialists. They monitor the blackboard looking for a set of facts that match a set of conditions that is valid for one of their rules. When a rule is triggered by such matching, they execute the prescribed action.

Each *blackboard* works with four *actuator agents* and each one will be responsible for one autonomic computing dimension. In figure 1, they are represented with a C, for the self-configuration element, with an H, for self-healing, with an O, for self-

optimization and a P, for self-protection. Further details on how to define actuators are provided in section 4.3.

The next level of the architecture has a general *blackboard* that serves the whole process. This general *blackboard* receives information (facts) from the lower level *actuators*, i.e., the *actuators* at the activity level work as monitors for process level. The information then is treated the same way as it is at the lower level – an *actuator agent* for each autonomic dimension.

Having defined the architecture as a whole, we can now briefly describe how it works. The user defines the process, i.e. the workflow, and for each activity defined, specifies the attributes, facts, condition and rules that shall be applied. When the process is started, the *monitors* keep sensing the activities, registering facts (attributes and respective values they are responsible for) in the *blackboard*. The *actuator*s read the *blackboard* and, when a set of conditions matches a rule, the *actuator* with the matched rule executes the indicated actions. Usually, one of the prescribed actions is to assert a fact onto the upper level *blackboard*, to promote coordination, i.e. to allow for the implementation of the autonomic behaviors for the process as a whole. The same behavior explained for the lower level is then manifested at the upper level.

4.3 Templates

Templates are complimentary to the architecture. They provide the means for the user to define activities/processes initial configuration, i.e. which attributes, facts, condition and rules will be used.

On templates we define an initial set of attributes, organized by resource types. The user chooses the wanted attributes and defines their domains. Another set of templates is now used to define the rules associated with each autonomic dimension (CHOP).

User defines rules based on the attributes chosen, specifying their relations with their domains, defining the actions to be taken and the associated priority.

We show an example of these templates use in section 5.2.

5. Case Study

This section presents a fictitious case study to demonstrate how the architecture works. The example used in this work is adapted from an organization that provides Information Technology services to customers.

5.1 Process' definition

Whenever a customer demands a system support, a request arrives at the department. The department assistant handles the request, collecting the needed data and generating a Support Report. Based on this report, the assistant creates a work order and defines the technician that will execute the service, with the assistance of the HR System. This technician, that should possess knowledge in the required support, generates the budget within a two-hour limit, performs the jobs under $100 and delivers the invoice to the customer. At the end of the process, the assistant verifies customer's satisfaction regarding the service provided.

As explained in section 4.2, the first step is the workflow definition. In this example, the definition is presented in figure 2, as an UML activity diagram [23].

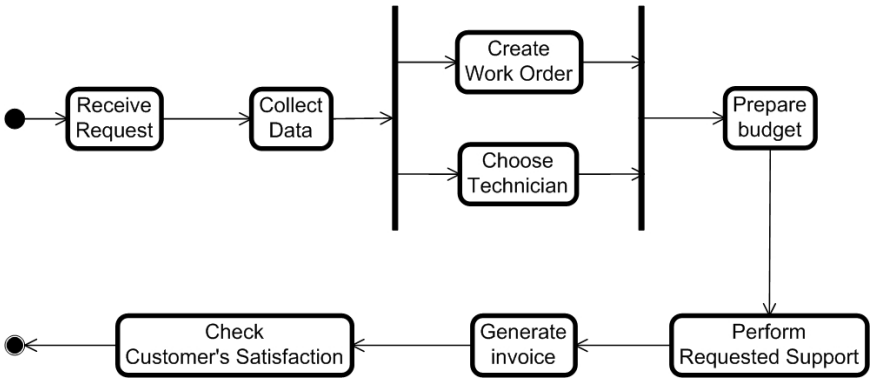

Fig. 2. Process Support Customer definition

5.2 Execution Definition

The second step is to define the parameters required for autonomic process execution, i.e. definition of facts and rules to be used. Using the templates explained on section 4.3 the following tables are defined. Table 2 shows for each resource which attribute (with its respective types) and with which frequency the monitor agent should update its blackboard. It is important to note that, although table 2 shows attributes organized by resource type, they are defined with a per activity analysis.

Table 3 shows which rules will be considered for each process. For each set of conditions the user defined the actions to be taken, including the assertion of new facts.

Table 2. Attributes from activities of Process Support Customer

Resources	**Attributes**	**Type**	**Frequency**
Time	Expected duration	Number	Every 1 hour
	Execution Time	Number	Every 1 hour
Human	*Assistant* assigned	Boolean	Once in start
	Technician assigned	Boolean	Once in start

	Process without executor	Boolean	Once in start
Input	Required *Support Report* present	Boolean	Every 1 hour
Output	Generate *Support Report*	Boolean	Once in end
Tool	*HR System* Available	Boolean	Every 1 hour
Knowledge	Required *Support Knowledge* present	Boolean	Once in start
Cost	Process expected cost	Number	Once in start
	Process current cost	Number	Every 1 hour

Table 3. Example of process rules

Process	Conditions	Actions	Facts Generated	Pr
Receive Request	*Assistant* assigned = false	Allocate available assistant	*Receive Request* without assistant	1
Collect Data	*Assistant* assigned = false	-	*Collect data* without assistant	1
	Generate *Support Report* = false	Execute activity again	Generate *Support Report* = false	1
Create Work Order	*Assistant* assigned = false	-	*Create Work Order* without assistant	1
	Required *Support Report* present = false	-	Required *Support Report* present = false	1
Choose Technician	*Assistant* assigned = false	Allocate available assistant	*Choose technician* without assistant	1
	HR System Available = false	Execute activity without *HR System*	*HR System* Available = false	2
Prepare	*Technician* assigned =	Allocate available	*Prepare budge* without	1

Process	Conditions	Actions	Facts Generated	Pr
Budget	false	technician	technician	
Perform Requested Support	*Technician* assigned = false	Allocate available technician	*Perform requested support* without technician	1
	Expected duration <= Execution Time	Continue executing process	*Perform requested support*'s expected duration <= *Perform requested support*'s execution time	2
	Perform requested support expected cost < *Perform requested support* current cost	-	*Perform requested support*'s expected cost < *Perform requested support*'s current cost	1
	Required *Support Knowledge* present = false	-	Required *Support Knowledge* present = false	2
Generate Invoice	*Technician* assigned = false	-	*Generate Invoice* without technician	1
Check Customer's Satisfaction	*Assistant* assigned = false	Allocate available assistant	*Check Customer's Satisfaction* without assistant	3

Process	Conditions	Actions	Facts Generated	Pr
Process	*Receive request* without assistant	Place request in queue	-	1
	Collect data without assistant	Execute all workflow again	-	1
	Create work order without assistant	Place request in queue	-	1
	Choose technician without assistant	Place request in queue	-	1
	Check Customer's Satisfaction without assistant	Cancel *Check*	-	3
	Prepare budget without technician	Execute workflow again from activity *Choose technician*	-	1
	Perform request support without technician	Execute workflow again from activity *Choose technician*	-	1
	Generate Invoice without technician	Execute *Choose technician* and execute *Generate Invoice*	-	1
	Generate Support Report = false	Execute all workflow again	-	1
	Required Support Report present = false	Cancel Request	-	1
	HR System Available = false	Inform user failure in *HR System*	-	2
	Perform Requested Support's expected duration <= *Perform Requested Support*'s execution time	Inform user the delay	-	2

Process	Conditions	Actions	Facts Generated	Pr
	Perform Requested Support's expected cost < *Perform Requested Support*'s current cost	Inform user the excessive cost	-	1
	Required *Support Knowledge* present = false	Inform user training necessity	-	2

6. Conclusions and Future Works

The objective of our research is to provide organizations with autonomic systems for process execution. The architecture presented here is one of the steps on this direction.

We believe the proposed architecture is simple, yet powerful. The approach combining agents and rule-based systems allows for the use of this architecture with modern software technology, e.g. SOA, and bringing the systems closer to the organization, since it is also based on business expert's knowledge.

It is important to note that although presented here in two levels only, the proposed architecture can easily be scaled up to work with many levels. This possibility goes towards our goal of providing autonomic properties to high levels of an organization. We believe, and we are also researching, the assembly of such architecture in multiple levels, where processes can report to a superior "super-process" blackboard, and so on, until a higher level, close to the organization upper management level can be reached. In the highest level though, in a process-oriented organization, we expect to deliver what we call the Autonomic Balanced Scorecard.

References

1. Tan, J. Wen, H. Awad, N. Health Care and Services Delivery Systems as Complex Adaptive Systems. Communications of the ACM, vol. 48, issue 5. ACM Press, 2005.

2. Heinl, P. 1998. Exceptions during workflow execution. In Proceedings of the Sixth International Conference on Extending Database Technology (Valencia, Spain, Mar.), H. -J. Schek, F. Saltor, I. Ramos, and G. Alonso, Eds.

3. Savarimuthu, B. T., Purvis, M., and Fleurke, M. 2004. Monitoring and controlling of a multi-agent based workflow system. In Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web intelligence, and Software Internationalisation - Volume 32 (Dunedin, New Zealand). J. Hogan, P. Montague, M. Purvis, and C. Steketee, Eds. ACM International Conference Proceeding Series, vol. 54. Australian Computer Society, Darlinghurst, Australia, 127-132.

4. Han, M., Thiery, T., and Song, X. 2006. Managing exceptions in the medical workflow systems. In Proceeding of the 28th international Conference on Software Engineering (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM Press, New York, NY.

5. Mourão, H. and Antunes, P. 2007. Supporting effective unexpected exceptions handling in workflow management systems. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM Press, New York, NY, 1242-1249.

6. Mueller, R., Event-Oriented Dynamic Adaptation of Workflows: Model, Architecture and Implementation. PhD thesis, University of Leipzig, 2002.

7. Mangan, P. and Sadiq, S. 2002. On building workflow models for flexible processes. In Proceedings of the 13th Australasian Database Conference - Volume 5 (Melbourne, Victoria, Australia). ACM International Conference Proceeding Series, vol. 18. Australian Computer Society, Darlinghurst, Australia, 103-109.

8. Nichols, J., Dermikan, H., Goul, M.: Autonomic Workflow Execution in the Grid. In IEEE Transactions on systems, man, and cybernetics, 2006

9. Aiber, S., Gilat, D. , Landau, A., Rainkov, N., Sela, A., Wasserkrug, S.: Autonomic self-optimization according to business objectives. In Proceedings of the International Conference on Autonomic Computing, 2004

10. Andrzejak, A., Herman, U., Sahai, A.: FEEDBACKFLOW - An Adaptive Workflow Generator for System Management. In International Conference on Autonomic Computing, 2005

11. Buhler, p. A., Vidal, J. M., Verhagen, H.: Adaptive Workflow = Web Services + Agents. In Proceedings. of the International Conference on Web Services, 2003

12. Verma, K., Sheth, A. P.: Autonomic Web Processes. In Proceedings of the Third International Conference on Service Oriented Computing, 2005

13. Pautasso, C., Heinis, T., and Alonso, G.: Autonomic execution of Web Service Compositions. In Proceedings of the IEEE International Conference on Web Services – ICWS'05. pp. 435-442, 2005. IEEE Press. 2005.

14. Pankratius, V. and Stucky, W. (2005). A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets. In Proc. Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, Australia. CRPIT, 43. Hartmann, S. and Stumptner, M., Eds., ACS. 79-88.

15. Heinis, T., Pautasso, C., Alonso, G.: Design and Evaluation of an Autonomic Workflow Engine. In Proceedings of the Second International Conference on Autonomic Computing, 2005.

16. Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G. 1999. Specification and implementation of exceptions in workflow management systems. ACM Trans. Database Syst. 24, 3 (Sep. 1999), 405-451.

17. Murch, R.: Autonomic Computing. Prentice Hall PTR, 2004

18. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995

19. Oshima, M., Lange, D. B.: Programming and Deploying Java Mobile Agents with Aglets. Second Printing, Addison Wesley, Boston, 1998

20. Zhao, Z., Belloum, A., Sloot, P;M;A., Hertzberger, L.O.: Agent Technology and Generic Workflow Management in an e-Science Environment. In Hai Zhuge and G.C. Fox, editors, Grid and Cooperative Computing - GCC 2005: 4th International Conference, Beijing, China, in series Lecture Notes in Computer Science, vol. 3795, pp. 480-485. Springer, November 2005. ISBN 3-540-30510-6. (DOI: 10.1007/11590354_61)

21. Shaw, M. Garlan, D. Software Architecture – Perspectives on an Emerging Discipline. Prentice Hall, 1996.

22. Corkill, D.D.: Blackboard Systems. AI Expert, 6(9):40-47, September, 1991

23. http://www.uml.org