Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

Policy driven and multi-agent based fault tolerance for Web services

TANG Jing-fan (汤景凡), ZHOU Bo (周 波), HE Zhi-jun (何志均)

(*School of Computer Science & Technology, Zhejiang University, Hangzhou 310027, China*)

E-mail: jingfan_t@zju.edu.cn; bzhou@zju.edu.cn; hezj@zju.edu.cn

Received Aug. 31, 2004; revision accepted Nov. 10, 2004

Abstract: This paper proposes a policy driven and multi-agent based model to enhance the fault tolerance and recovery capabilities of Web services in distributed environment. The evaluation function of fault specifications and the corresponding handling mechanisms of the services are both defined in policies, which are expressed in XML. During the implementation of the services, the occurrences of faults are monitored by the service monitor agent through the local knowledge on the faults. Such local knowledge is dynamically generated by the service policy agent through querying and parsing the service policies from the service policies repository. When the fault occurs, the service process agent will focus on the process of fault handling and service recovery, which will be directed with the actions defined in the policies upon the specific conditions. Such a policy driven and multi-agent based fault handling approach can address the issues of flexibility, automation and availability.

Key words: Policy driven, Multi-agent based, Fault tolerance, Web service
doi:10.1631/jzus.2005.A0676 **Document code:** A **CLC number:** TP391

INTRODUCTION

The widespread deployment of inexpensive communications technology, computational resource in the networking infrastructure causes difficulty in the management of hundreds of thousands of accessible services in distributed wide-area environment including fault tolerance. Fault tolerance is the survival attribute of service systems. Experience has shown that systems with complex interacting activities are very prone to errors and failures due to their extreme complexity and long running process. Furthermore, to achieve scalability and flexibility of Web services in distributed environment, dynamic replication technique (Lee and Weissman, 2001) is adopted to allow an arbitrary service to be performed by multiple service replicas on different servers. It brings more difficulty for the host servers to detect and handle the unexpected fault of the service replica if there is not enough information provided by it.

The faults of a specific service should be classified and pre-defined before the service replicas are deployed for execution. The actions and mechanisms for handling the faults should be also defined as they can be dynamically changed due to the different environments.

One challenge is to provide efficient mechanism to verify the occurrence of fault in a service in distributed environment. The main requirements include: (1) The verification of fault occurrence should be capable of being ensured by synthetic evaluation on basis of some specified factors; (2) The required information to evaluate the state of the service should be capable of being monitored and collected. It means the occurrence of fault can be detected; (3) The corresponding mechanisms on fault tolerance under different conditions should be provided in the approach.

In this paper, we propose a policy driven and multi-agent based fault tolerance framework for distributed Web services, which uses pre-defined policies to specify the classification of the faults and direct the process of services execution and fault handling by multi-agents. Each service has its own policies for fault specification and handling, which will enable the host servers to easily detect and deal with the fault occurrence of the service replica run on it.

We give the formal definition of the model, describe the whole architecture by highlighting the components, and discuss the mechanism details in our model of fault tolerance for Web services.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 presents an overview of our approach. Section 4 describes the service policies in detail. Section 5 discusses the detailed mechanism of fault monitoring and handling. Section 6 concludes the article.

RELATED WORK

The function of fault tolerance can be described as "to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service" (Avizienis, 1985).

The work of Alonso *et al.*(2000) concentrates on the two aspects of fault tolerance in the fault-tolerant workflow systems which include exception handling and replication strategies for increased availability. A novel method was proposed by Chang (2001) to enhance the fault-tolerance and recovery capabilities of critical network services in a distributed computing environment. A service can be dynamically dispatched onto any available host. The cluster monitor and fail over function comprise the center of the recovery scheme to improve the fail over and load-balancing network technology by reducing the resource usage and cluster-support overhead. El-Darieby *et al.*(2003) presented a new scalable fault notification protocol, which caused failure notification signals to travel vertically up and down a network hierarchy instead of horizontally along the services routes. A flexible failure handling framework for the Grid called Grid-WFS (Grid Workflow System) was proposed by Hwang and Kesselman (2003) to meet the Grid-unique failure recovery requirements.

Software agents are accepted as powerful high-level abstraction for modeling complex software systems through the interactions among the autonomous agents to achieve the specific tasks. They had been used in Grid management (Overeinder *et al.*, 2002), mobile computing environment (Ding and Malaka, 2000), network monitoring (Bivens *et al.*, 1999) and fault-tolerance in process automation systems (Seilonen *et al.*, 2002), etc.

Policies (Liabotis *et al.*, 2001; Sloman and Lupu, 2002; Sacks *et al.*, 2003; Yang *et al.*, 2002; Katchabaw *et al.*, 1996; Hong *et al.*, 2002) are rules governing the choices in the behavior of a system. They are increasingly being for implementing flexible and adaptive systems for management of Internet services (Liabotis *et al.*, 2001). Yang *et al.*(2002) presented a new view of Grid architecture by integrating the policy-based Grid management middleware and active network middleware to improve the manageability, automation and flexibility of Grid system. Initial research was done by Katchabaw *et al.*(1996) on policy-driven fault management, focusing on application-level faults. Hong *et al.*(2002) presented a policy-based solution to support delivery and management of network infrastructure and enhance network services. It discusses policy-driven service management with emphasis on QoS management, dynamic service provisioning, and Internet pricing and billing.

In our approach, we use multi-agents to detect occurrence of fault and achieve intelligent mechanism of policy driven fault handling. The term 'agent' used here refers to a relatively simple autonomous system component that exists in a community with other agents and co-operates with them to achieve some complex objectives, such as fault tolerance.

OVERVIEW OF MODEL

To build fault-tolerant services, there are two major approaches, which are commonly known as the Primary-Backup approach (PB) and the State-Machine approach (SM). PB can tolerate crash and omission faults and runs more economically than SM, but SM can tolerate more serious faults, including arbitrary or Byzantine faults. Some other mechanisms, such as retrying, replication and checkpointing, can be applied on the task level so as to prevent task crash failures from being propagated to high level (i.e. service level).

In our approach, the fault specifications and the corresponding handling mechanisms of the services

are both defined in service policies. Such service policies are bound with specific service. Each service has its own policy on fault specification and handling. For example, the service may have special requirement on the time-out, or have replica running on backup servers, etc.

Furthermore, multi-agents are adopted in our model to monitor and manage the occurrence of the fault.

We can define our model *Fm* as:

$$Fm=(S, SP, AG)$$

Definition 1 A service (*S*) is defined by a quadruple (*SID*, *SD*, *SIM*, *SS*) where *SID* is the unique ID of the service, *SD* is the description of the service, *SIM* is the set of the service implementation and *SS* is the set of the service servers to run the service.

Definition 2 A service policy (*SP*) is also defined by a quadruple (*SID*, *PID*, *EF*, *Act*) where *SID* is the unique ID of the service, *PID* is the unique ID of the policy, *EF* is the evaluation function on the factors to decide the state of the service (*EF=evaluate_func*(p_1, p_2, …, p_n), where p_1, p_2, …, p_n mean the factors to evaluate the service state), *Act* denotes the corresponding actions upon the different values of the evaluation function (*Act=action*(*EF*)).

Definition 3 An agent group (*AG*) is defined by (*SID*, *PID*, *SPoA*, *SMnA*, *SPcA*) where *SID* is the unique ID of service, *PID* is the unique ID of policy, *SPoA* denotes service policy agent, *SMnA* denotes service monitor agent and *SPcA* denotes service process agent.

When starting a service, an agent group (AG) will be established including service policy agent (SPoA), service monitor agent (SMnA) and service process agent (SPcA). SPoA can acquire the corresponding service policy from the service policy repository (SPR) by submitting policy query with SID and generate the local knowledge with the information of fault description and actions for handling different faults, etc. The local knowledge will be stored into local knowledge repository (LKR) and be shared by SMnA and SPcA, and will be dynamically updated by SPoA according to the timely change of the service policy. During the execution of the service, SMnA will monitor the fault occurrence due to the fault description in the local knowledge repository. When fault occurs, it will report the fault to SPcA, and SPcA will be responsible for handling the fault and recovering the service due to the information provided from local knowledge repository. For example, it will restart the service, or dynamically re-allocate the service onto the available server, or startup the backup server as the primary server for service, or startup another implementation for service, etc.

Fig.1 shows the overview of our model.

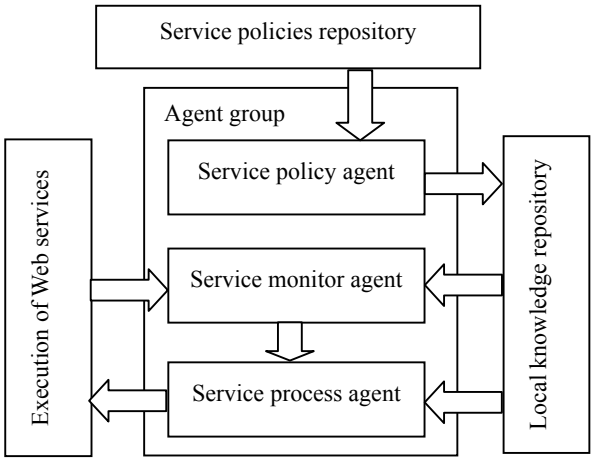

Fig.1 Overview of model

SERVICE POLICIES

The service policies are expressed in XML since it has a text-based representation which imposes few restrictions on network technology or protocols and (through the use of XML schema) it has a sufficiently strict syntax to permit automated validation and processing of information in an unambiguous way. Service policies allow selections from a range of options provided by the developer of the policy-controlled service, comprised of rules in the form of *if<condition>then<action>* (Yang *et al.*, 2002). This allows flexibility to be built into a system by supporting a range of different behaviors rather than hard-coding a particular behavior. Furthermore, the use of XML as an intermediate representation still allows service policies to be developed using any existing approach. Some kind of policy editor GUI can be provided to give a user-friendly and flexible way to manage the service-specific policies stored in SPR. There are corresponding policies defining dif-

ferent rules of fault specification and handling for different services. For instance, some services are critical and should be replicated for execution on different servers. Some services may have multiple implementations with different available execution behavior, i.e., the first implementation is fast but unreliable, and the second slow but reliable. Such different execution behavior can be interchanged due to different running environment.

Furthermore, the QoS (Liu *et al.*, 2003) requirements may be also included in the service policies, which will increase the complexity for fault tolerance. For example, if the load of the host system reaches a specific threshold, the service performance will decline or the service may crash. Such cases should be detected and avoided through the direction of the service policies.

The service policies are triggered after evaluating conditions involving locally available information. Some evaluation function should be provided on the synthetic evaluation of the variable conditions. It may be like: $Ef(p_1, p_2, \ldots, p_n)$, where $p_1, p_2, \ldots, p_n$ mean the locally available parameters. Such evaluation function may be complicated and related with the internal business logic of the specific service.

The following shows the information details, which will be included in the service policy (Fig.2 shows an example of a service policy in XML):

Service The referenced information of the service such as service ID, service name and service type, etc.;

Info The elementary information on the policy itself including policy ID, creator, etc. The policy ID and service ID will be used to identify the service policy;

Param-specs The locally available parameters (information) for evaluating the conditions, such as valid input and output, memory usage and CPU load, etc.;

EvalFunction The evaluation function of synthetic parameters for decision making on actions under different conditions;

Trigger The relevant policies triggered by different results from the evaluation function to trigger the actions;

Action The details of actions on the evaluation of the local available parameters;

Others Contains other information such as security and QoS requirements of the service, etc.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<policy xmlns = "http://~sszutc/policy" xmlns:xsi =
"http://~sszutc/XMLSchema-instance" xsi:schemaLocation
="http://~sszutc/policy file:///home/~sszutc/policy.xsd">
<service>
        <service-id>SIN001</service-id>
        <name>service001</name>
        <type>Single</type>
</service>
<info>
        <policy-id>Policy1</policy-id>
        <creator>tjf</creator>
        <authority>ALL</authority>
</info>
<param-specs>
        <param-spec variable=var_1>
                <description>descript_1</description>
        </param-spec>
        <param-spec variable=var_2>
                <description>descript_1</description>
        </param-spec>
</param-specs>
<evalFunction value=evalfun_variable>
        EvalFun(param-specs)
</evalFunction>
<trigger variable=evalfun_variable>
        <Branch>
                <Condition value=value_1>
                        <Action>Act_1</Action>
                </Condition>
                <Condition value=value_2>
                        <Action>Act_2</Action>
                </Condition>
        </Branch>
</trigger>
</policy>
```

Fig.2 An example of a service policy in XML

POLICY DRIVEN AND MULTI-AGENT BASED FAULT TOLERANCE

Faults (Alonso *et al.*, 2000; Katchabaw *et al.*, 1996; Seilonen *et al.*, 2002) are natural occurrences in any software system. Due to the inherently unreliable nature of the services system in distributed environment, faults or error conditions include hardware failures (e.g., host crash, network partition, etc.), software errors (e.g., memory leak, numerical exception, etc.) and other sources of failures (e.g., machine rebooted by the server owner, network congestion, excessive CPU load, etc.).

Fault tolerance for Web service is the ability of a service to continue valid operation after the service, or

part of it, fails in some way. In order for a service suffering a failure to continue, the state of the service should be able to return to the previous valid state to continue its tasks.

In our approach, we adopt policy driven and multi-agent based fault tolerance for Web services to achieve automation and flexibility.

Service policy agent

As illustrated in Fig.3, service policy agent (SPoA) is responsible for acquiring policies from SPR and generates local knowledge into the LKR for sharing. In SPoA, policy query module is used to query the service policy from SPR by sending query messages with SID and PID to it. Here, we assume a service has different policies under different environments. After the policy is obtained from SPR through XML messaging, the policy parsing module will extract the specification of faults and corresponding handling mechanism from the policy to generate the knowledge, which can be recognized locally. Local knowledge generator will encode the knowledge and store it into LKR. Such encoded knowledge may include the information for checkpoints (stable states of the service), fault description, fault detection, fault handling, etc.

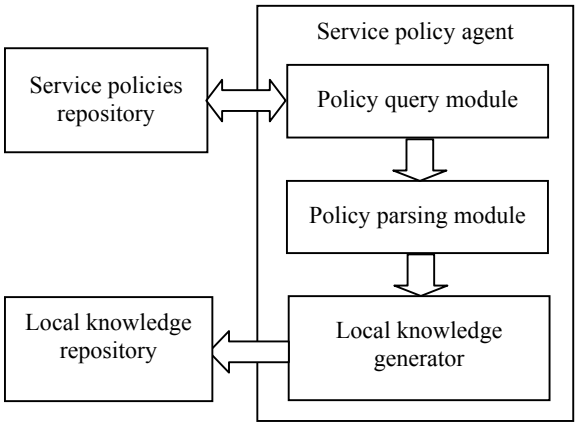

Fig.3 Service policy agent

Service policies may be changed during the execution of the service. SPoA will be responsible for acquiring the up-to-date data from SPR and updating the corresponding knowledge of LKR dynamically. So some synchronization mechanism should be adopted on the operations (such as read and write) on LKR, and SMnA and SPcA will adjust their behaviors adaptively to meet the new requirements of the service policies. For example, the evaluation function in service policies is modified.

Service monitor agent

During the life cycle of a service, which includes initialization, starting, running, ending and finalization, some state conversion may occur (Fig.4). For instance, the state of a service may be changed from normal to abnormal when the host server is overloaded or other cases happen. Abnormal is an interim state, which will be inclined to failure. So it should be recovered to the state of normal through specified adjustment. A service is expected to be run under the state of normal to achieve its tasks. Some parameters (factors) can be collected to evaluate the state of a service such as valid input and output, memory usage and CPU load, etc.

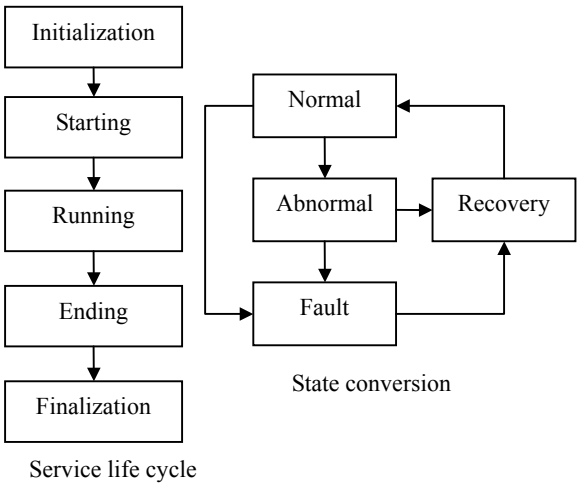

Fig.4 Service life cycle and state conversion

Service monitor agent (SMnA, see Fig.5) provides interfaces in our model to communicate with service servers (if the service is executed on multiple servers) to collect such running parameters of both the services and the host servers. Through the evaluation function, which is dynamically defined in service policies (see Section 4), the state of the service can be validated and the behaviors of SMnA will be decided upon the synthetic evaluation. For instance, when the result of evaluation function shows that current conditions may result in fault occurrence or the actual fault occurs, SMnA will report to SPcA and wait for

its response via standard XML messaging.

In our approach, checkpoint technique (Zhang and Chakrabarty, 2003; Clematis *et al.*, 1998) is one of the techniques adopted for service recovery for fault tolerance. Checkpoints repository (CPR) is used to keep the checkpoints of service for service recovery with the mechanism of checkpointing defined in the service polices. When the execution of the service arrives at a checkpoint, SMnA will keep the context information of the checkpoint in the checkpoints repository for the recovery of the service. Some other mechanisms may be adopted on the service recovery for fault tolerance, such as replication technique (Lee and Weissman, 2001), which will be decided by the specification in service policies.

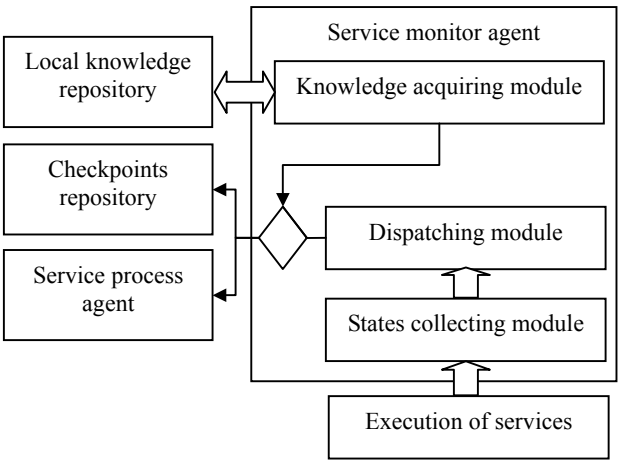

Fig.5 Service monitor agent

Service process agent

Fig.6 shows the processing of policy driven fault handling by service process agent (SPcA).

SPcA receives the information from SMnA through the fault receiving module. The knowledge acquiring module will query the action policy from LKR according to the fault condition. Policy execution module will then perform the specified actions.

Some actions for fault tolerance and service recovery are listed as follows:

1. Recovering the service from checkpoint

As mentioned in Section 5.2, the checkpoint of service will be stored in the checkpoints repository during the execution of the service. The service can be recovered from the last updated checkpoint like the recovery process in DBMS.

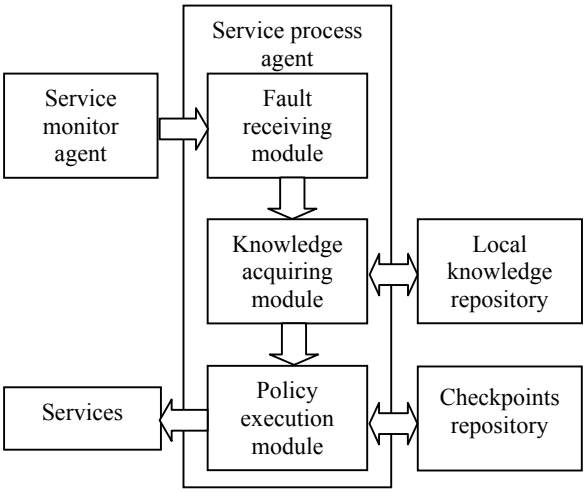

Fig.6 Service process agent

2. Restarting the service

When some fatal fault occurs in the service application (i.e. the service application exits abnormally), it will be initiated and restarted.

3. Dynamically re-allocate the service onto the available server

If the host server is overloaded or crashes, the service will be moved to other available server and recovered from the checkpoint (if there is checkpoint for the service).

4. Startup the backup server as the primary server for service

If the service is duplicated on multiple servers and the primary server fails, the backup server will be run as primary server to continue the service.

5. Startup another implementation for service

If the service has multiple implementations and the current implementation fails, another implementation for the service can be started up.

When the service is recovered to execution, SMnA will continue to monitor the service during the life cycle, until the service ends.

CONCLUSION

In this paper, we proposed a novel approach to address the issues of fault tolerance of services in distributed environment, which is policy driven and multi-agent based. XML is used in our approach to describe the service policies to provide a flexible way for managing them. The fault specifications and the

corresponding handling mechanisms of the services are both defined in policies. The dynamic policies will be used by the agents to automatically detect the occurring of fault and adopt intelligent mechanisms of fault handling. Such a policy driven and multi-agent based mechanism of fault tolerance address the issues of flexibility, automation and availability for Web services.

References

Alonso, G., Hagen, C., Agrawal, D., Abbadi, A.E., Mohan, C., 2000. Enhancing the fault tolerance of workflow management systems. *IEEE Concurrency*, **8**(3):74-81.

Avizienis, A., 1985. The *N*-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, **SE-11**(12):1491-1501.

Bivens, A., Gao, L., Hulber, M., Szymanski, B., 1999. Agent-Based Network Monitoring. Proceedings of the 3rd International Conference on Autonomous Agents, p.41-53.

Chang, W., 2001. A Resource Efficient Scheme for Network Service Recovery in a Cluster. IEEE 2001, p.1087-1091.

Clematis, A., Deconinck, G., Gianuzzi, V., 1998. A Flexible State-saving Library for Message-passing Systems. Proc. 6th Euromicro Workshop on Parallel and Distributed Processing, IEEE Comp. Soc. Press.

Ding, Y., Malaka, R., 2000. An Agent-based Architecture for Resource-Aware Mobile Computing. Proc. Intelligent Interactive Assistance and Mobile Multimedia Computing (IMC2000).

El-Darieby, M., Petriu, D., Rlia, J., 2003. Hierarchical End-to-End Service Recovery. Proceedings of the 8th IEEE Symposium on Integrated Network Management (IM'03), p.649-661.

Hong, L., Dong, B., Wei, D., 2002. A Policy-Based Solution for Management of Enhanced Network Services. Proceedings of IEEE TENCON'02, p.1684-1687.

Hwang, S., Kesselman, C., 2003. Grid Workflow: A Flexible Failure Handling Framework for the Grid. Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03).

Katchabaw, M.J., Lutfiyya, H.L., Marshall, A.D., Bauer, M.A., 1996. Policy-Driven Fault Management in Distributed Systems. Proceedings of the Seventh International Symposium on Software Reliability Engineering (ISSRE' 96).

Lee, B., Weissman, J., 2001. Dynamic Replica Management in the Service Grid. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), p.433-434.

Liabotis, I., Prnjat, O., Sacks, L., 2001. Policy-based Resource Management for Application Level Active Networks. Second IEEE Latin American Network Operations and Management Symposium.

Liu, Z.L., Zhu, M.L., Jiang, M., Wu, T.F., 2003. An overview of QoS protocols and architecture. *Journal of Zhejiang University (Engineering Science)*, **37**(3):288-294 (in Chinese).

Overeinder, B.J., Wijngaards, N.J.E., van Steen, M., Brazier, F.M.T., 2002. Multi-Agent Support for Internet-Scale Grid Management. Proceedings of the AISB'02 Symposium on AI and Grid Computing, p.18-22.

Sacks, L., Prnjat, O., Liabotis, I., Olukemi, T., Ching, A., Fisher, M., Mckee, P., Georgalas, N., Yoshii, H., 2003. Active robust resource management in cluster computing using policies. *Journal of Network and Systems Management*, *Special Issue on Policy Based Management of Networks and Services*, **11**(3):329-350.

Seilonen, I., Appelqvist, P., Halme, A., Koskinen, K., 2002. Agent-Based Approach to Fault-tolerance in Process Automation Systems. Proceedings of the 3rd International Symposium on Robotics and Automation.

Sloman, M., Lupu, E., 2002. Security and management policy specification. *IEEE Network Special Issue on Policy*, **16**(2):10-19.

Yang, K., Gailis, A., Todd, C., 2002. Policy-Based Active Grid Management Architecture. Proceedings of 10th IEEE International Conference on Networks (ICON02), p.243-248.

Zhang, Y., Chakrabarty, K., 2003. Fault Recovery Based on Checkpointing for Hard Real-Time Embedded Systems. Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, p.320-327.