Policy Based Fault Tolerant Scheme in Pervasive Computing

P. G. V. Suresh Kumar

Associate Professor, Department of IT & SC, AAiT, Addis Ababa University
Addis Ababa, Ethiopia

Abstract *--- Pervasive computing integrates digital and physical devices. Users can access digital data and applications from the environment as easily as accessing them through their computers. The system has to be flexible to various kinds of faults and should be able to function even though the fault occurs. This paper deals with various classes of failures, their implication to pervasive computing, the challenges to be addressed in designing a fault tolerant pervasive computing system and a policy based fault tolerant pervasive computing system where the policy deals with the separation of the program into distinct features. This policy based approach is more flexible and adaptable.*

Keywords — *Fault Tolerant, Pervasive Computing, users etc.*

I. INTRODUCTION

Pervasive computing provides a platform for context-aware computing that enables automatic configuration of a pervasive system based on the environment context. Pervasive computing to be successful, it's functioning should be transparent to the user. Such transparency is achievable if faults in the system are masked and user intervention is sought only when absolutely required. Pervasive computing technology exists in the user's environment and aids the user in performing various tasks. The sustainability of this technology depends on it being non-intrusive. In order to achieve this goal, faults in a pervasive system should be automatically masked and user notified only when absolutely required. Fault Tolerance issues have not been well explored so far in pervasive computing research. Since pervasive computing environments operate in the same physical (as well as virtual) space as humans, they can be exasperating (and sometimes hazardous) if they are not resilient to faults. Several researchers have expressed the need for reliable pervasive systems and mention that reliability issues must be readdressed in the realm of pervasive computing

II. CLASSIFICATION OF FAILURES

A typical pervasive system consists of commercial off-the-shelf (COTS) software and devices whose reliability is not guaranteed. COTS software are sold as "black boxes" and may not be subject to rigid development, verification or testing processes. Interoperability issues further reduce the reliability of a pervasive system. Mobile devices such as handhelds and laptops, with limited battery power, cannot be regarded as totally reliable. Connectivity failures due to devices going out of range or other errors in networks add to faults in a pervasive system. Besides, a pervasive system has a core set of services (like naming, trading, file system, event delivery, and discovery and context services) that provide necessary functionality. These services can also fail. Broadly, faults in a pervasive system can be classified into device, application, network and service failures. We discuss these individually in the following sections.

A. Device Failures

A pervasive system consists of different kinds of devices such as desktops, laptops, sensors, actuators, displays, speakers, scanners, cameras and projectors. Each device has its own set of faults that can potentially contribute to the failure of the pervasive system. Mobile devices, such as laptops and handhelds, have physical constraints such as finite battery power and limited signal strength. So if the battery goes down or if the signal strength is too low they get disconnected from the pervasive system and are regarded as having failed. A more acute problem with devices is when they are alive but operate incorrectly. This is common in faulty sensors and is called a Byzantine failure.

B. Application Failures

Designing reliable software is an expensive process and the cost of debugging, testing and verifying can easily range from 50 to 75 percent of the total development cost. Even in well-tested software systems, bugs of varying severity are found. Pervasive computing includes commercial off the shelf applications that may not be well tested. In some situations, applications may work well as stand-alone software but may not inter-operate correctly or reliably with other software. Therefore, pervasive systems should make few reliability assumptions bout applications. Application failures include application crashes due to bugs, operating system errors, unhandled exceptions and faulty usage. Pervasive applications are also likely targets for malicious software such as viruses and worms. Viruses and worms cause fail-stop or Byzantine failures.

C. Network Failures

Pervasive systems consist of wired and wireless devices. Therefore, a reliable pervasive system should account for network failures caused by low signal strength, devices going out of range and unavailability of communication channels due to heavy traffic. Automatic detection of the failure type is an important issue in pervasive computing.

D. Service Failures

Essential services include naming, event and discovery services. Some pervasive systems support other services such as a trading service that enables device discovery, context services that enable context-aware computing and file system services for ubiquitous data access. Examples of service failures include service crashes due to bugs and operating system errors, faulty operation of services like sensing incorrect context, wrong inferring and lossy delivery of events. Service failures can potentially lead to failure of the pervasive system.

III. IMPLICATIONS OF FAILURES

Pervasive computing integrates digital devices seamlessly in our physical environment. Digital devices co-exist with physical devices to aid in accomplishing everyday tasks.

- Hazard to Life
- Inappropriate Resource Control and
- Usage
- Security Vulnerabilities
- Inferring Incorrect Context
- Challenges Facing Fault Tolerance

IV. Challenges Facing Fault Tolerance

Each new area poses its own set of challenges for which the past techniques have limited applicability. The fault tolerance issues in the pervasive computing are given below:

A. Fault Detection

Efficient fault detection is a tough challenge in a Pervasive system. An application or device that stops on failure can be detected through timeout techniques such as *heartbeat* messages. These heartbeat messages significantly add to the network traffic and the number of messages can overwhelm fault detectors. Further, network failures can lead to unreachable nodes making it complex to distinguish between entity failure and network failure. Byzantine fault detection is a tougher problem. The Byzantine fault model includes failures in which entities do not stop but operate incorrectly. Byzantine faults result in inferring incorrect context and inappropriate resource usage. Heartbeat messages cannot be used to detect all Byzantine faults, as entities do not stop sending heartbeats.

B. Fault Containment

Once a fault is identified, it should be isolated to prevent its propagation to other parts of the system. A pervasive system contains interdependent applications and services that communicate fosters fault propagation and makes fault containment a tough challenge.

C. Transparent Fault Tolerance

Pervasive computing as a system that blends in with the physical environment and whose functioning is transparent to the user.

D. Good Fault Reporting Mechanisms

When a fault cannot be tolerated, it should be reported to the user in a non-intrusive manner Faults can be reported through visual representation on display devices, audio representation on speakers or any other means that can be perceived by the human senses.

V. Overview of the error tolerant architecture

The architecture is in view of the principles of policy-driven systems which are applied in various adaptive systems. As error messages can be regarded as context in the pervasive computing system, policy-based scheme can be applied to regain a normal status for enhance the dynamic adaptation of error-tolerant approach. The design tenet of policy is based on the Separation of Concerns principle. For that reason, it can separate the error-tolerant logic (rules) from the implementations of error recover (programming code). Context data is provided by context-aware component,

Policy Parser matches all polices with the Context data and select the appropriate policy.

- Then judge whether the conditions in the Context-Event table are met.
- If they are met, Event Monitor triggered by relevant events and notifies Policy Executor.
- Policy Executor will executes the predefined rules and lead to a change of the context information.

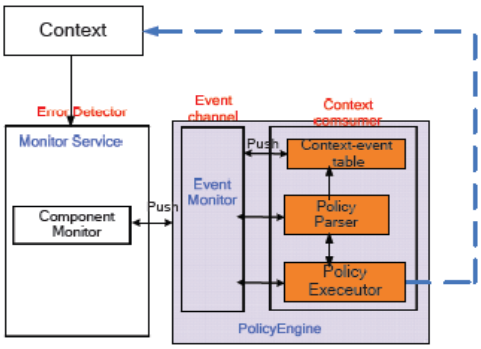

Figure 1. The basic idea of the adaptation architecture.

B. Process of Error Tolerance

The process of error tolerance is given below:

1. User defines the error policy, Policy Engine loads the policy file which is defined in XML.
2. Monitor Service monitors the status of the component. In case there is a malfunction of certain component, it can create the platform context of component fault and modify the context value of component status in the context list. Moreover, it pushes events of component fault to Event Monitor;
3. Event Monitor pushes the event to Policy Controller;
4. Policy Parser queries the error tolerance rule via Context-event Table;
5. Policy Executor executes the error tolerance action in the rule.

VI. Policy based component error tolerance approach

A.Extension of CCM Component Container for Component Fault Detection

The CCM component container for supporting component error tolerance shown in fig 2.The infrastructure of CCM component container including increasing the context list, context-event table and Policy Executor for supporting the parse and handle of policy. Context list is a two-dimension table, which consists of component name, context name and the value of the context data. Context-event table can describe the change of physical and information space. It includes the field of context name, event ID and event name.

Policies are defined as a set of sophisticated rules which is described rules which are described in XML.They indicate the execution of action reacts in a specific context. The functionality of Policy Engine is monitoring the change of the value of Context and executing the predefined polices. It comprises the Event Monitor, Policy Executor, Policy Parser and POA (Portable Object Adapter).

→Policy Parser is a CORBA object, which is a XML parser. The functionality of Policy Parser is to match all polices in the Context Event Table and select the
appropriate policy to dynamically generate the Policy Executor.

→Policy Executor is a two dimension pointer array. The first dimension is context name, and the second is a pointer which point to a group of CORBA objects and corresponding interfaces. The policy is storied as a structure of policy condition, action type, component name, method of component. Policy Executor can check the condition of policy is met, if it is true, the method of the component can be triggered by POA.

→Event Monitor can periodically monitor the change of value the Context List. When the change is detected, it judge whether the context event is in the Context Event Table, if it is false, insert the context event into Context Event List.

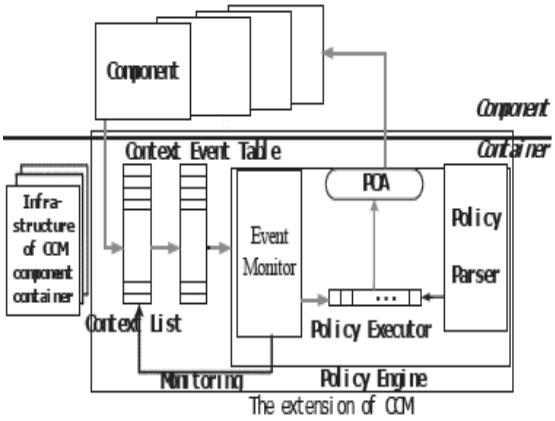

Figure 2. Extension of CCM Component Container for Component Fault Detection

B. Heartbeat scheme

As timeout based heartbeat techniques can be applied widely to detect the stops or failures of devices or applications, we propose a heartbeat scheme to detect the
status of components. The components subscribe the heartbeat event from the extended component container before starting. The context manager component will publish the heartbeat event periodically.When detected the component fault, the corresponding event is triggered. Moreover, Event Monitor notifies the Policy Executor to handle the error tolerant event. At last, the policy is activated via Policy Engine to reload or rest the failed component.

VII. Error Tolerance Policy

A) Component fault detection

We have developed a fault tolerance technique that uses context information to tolerate component faults. For the support of the error tolerance policy, we define platform context to describe the status of the component. The platform context is defined as a two tuple<Contextid,Value> Contextid indicates the name of the component, and Value gives the content of the status of the component which is “Running”, “Idle” or “Failure”. As fail-silence can be regarded as producing the proper output, a detectably invalid output, or no output, the extension of CCM Component Container provides a runtime environment with specific fail-silence detection mechanisms that provide error detection user applications. Each component sends a periodic heartbeat message to Context List informing that it is alive. Once the component fault is detected, it sends these monitored data to context list.

A) Policy driven component recovery

The adaptation error recovery policy consists in a set of rules, each of the form Event Condition Action form. The *event part* indicates the circumstances when a rule is to be triggered;

_ *Conditions are* constraints on the action and environment, evaluated when a rule is triggered to determine whether the action should be carried out.

_ *Action is* a (sequence of) operation(s) of components which are applied to the system when the rule is actuated.

There are two cases of component self-healing. One is to be healed by itself. In case of typical faults like abrupt change of memory status or processor speed or signal strength, Policy Executor simply restarts the component by executing the error tolerance action in the rule. To treat the unusual behavior (abrupt change of memory status or processor speed or signal strength) of the system due to the use of any specific application, Policy Engine simply deactivated that component without notifying the user. The other approach is to choose the substitutable component. There are two cases of choosing the substitutable component. One is occurred in the local component container. The second is in the remote component container.

VIII. Conclusion

Transparency is an important characteristic of pervasive computing. If pervasive computing has to be sustainable, it should be unobtrusive and its faults transparent to the user. The policy mechanism is based on not only event condition action rules, but also the Separation of Concerns principle for easily extensible to support the error recovery scheme. The proposed error recovery policy indicates the rules that govern how to recover the component. Thus, the policy should be flexible enough to accommodate changing contexts and this makes error tolerance management in we have presented policy based adaptive error recovery scheme for pervasive computing.

IX. ACKNOWLEDGEMENT

We would like to convey our gratitude to Sri. P. Bhaskara Rao, Retd. Teacher, India, Nune Srinivas, faculty of SECE, Mrs. Pendem Padmaja, India, a special thanks to Mr. P.V. Subrahmanyeswara Rao and Daniel Head ITSC under School of ITSC in AAiT Addis ababa University, Ethiopia befor their technical support to realize the this proposal discussed in this paper.

X. REFERENCES

[1] Avizienis A. Design of fault-tolerant computers. In: Proceedings of AFIPS Conf erence. Washington D C: Thompson Books, 1967. 31: 7,33-43.

[2] Shiva Chetan, Anand Ranganathan, Campbell, Roy. Towards fault tolerance pervasive computing, IEEE Technology and Society Magazine,2005 Spring, 38-44.

[3] Eung-Nam Ko. An adaptive fault tolerance for situation-aware ubiquitous computing, Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05), 144-149.

[4] Byoung uk Kim, Y. Al-Nashif, S. Fayssal, S. Hariri, M. Yousif, Anomaly-based Fault Detection in Pervasive Computing System, Proceedings of the 5th international conference on Pervasive services(ICPS'08), July 6–10, 2008, Sorrento, Italy,147-155.

[5] Christof Fetzer, Karin Hogstedt, Self*:A Data-Flow Oriented Component Framework for Pervasive Dependability, Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003). Jan 15-17, 2003 January, 2003, pp.67-73.68

[6] Lamport, Shostak and Pease, "The Byzantine Generals Problem", in *Advances in Ultra-Dependable Distributed Systems, N.Suri, C.J.Walter, and M.M.Huue(Eds.) IEEE Computer Society Press*. 1995.

[7] M.Satyanarayanan, "Pervasive Computing: Vision and Challengges", *IEEE Personal Communications*, pp.10-17, Aug. 2001.