International Journal of Advanced Technology in Engineering and Science
Vol. No. 12, Issue No. 01, January 2024
www.ijates.com
ijates
ISSN 2348 - 7550

NETWORK BASED SOFTWARE ARCHITECTURE FOR ADAPTIVE INTELLIGENT SYSTEM

Isharat Ali[1], Anurag Gupta[2]

[1]Research Scholar (Computer Science and Engineering, Shri JJT University, India)

[2] Assistant Professor (Computer Science and Engineering, ABES Engineering College, India) [1]mirzaishratali@gamil.com, [2]anurag.ideal09@gmail.com

Abstract

In this paper we describe an approach in which intelligent adaptation is supported by the use of software architectural to monitor an application and guide live changes to it. An important requirement for inescapable computing systems is the ability to adapt at runtime to handle varying resources, user/vehicle mobility, authorization, and intelligent system. The use of externalized events permit one to make reconfiguration decisions based on a global inescapable of the running system, apply analytic events to determine correct system monitoring. We illustrate the application of this idea to inescapable computing systems, focusing to intelligent based on performance-related criteria and architecture.

Keywords: high dependability robustness, adaptability, and availability, system evolution.

Introduction

An important requirement for inescapable computing systems is the ability to adapt intelligent at runtime to handle such things as user/vehicle mobility, resource variability, changing user activities, and system intelligent, in a inescapable computing world more and more systems have this requirement, because they must continue to run with only minimal human oversight, and cope with variable resources as a user moves from one environment to another, Recently several researchers have proposed an alternative approach in which system models – and in particular, software architectural systems – are maintained at runtime and used as a basis for system reconfiguration and repair [1]. An architectural system of a system is one in which the overall structure of a running system is captured as a composition of coarse-grained interacting components [28]. An architectural system can provide a global inescapable on the system allowing one to determine non-local changes to achieve some property. Architectural systems can make “integrity” constraints explicit, helping to ensure the validity of any change. By “externalizing” the monitoring and adaptation of a system using architectural systems, it is possible to engineer adaptation mechanisms, infrastructure and policies independent of any particular application, thereby reducing the cost and improving the effectiveness of adding intelligent-adaptation to new systems.

In this paper we illustrate how network-based intelligent adaptation can be applied to inescapable computing systems. Specifically, we show how to use the approach to support adaptation of applications in a inescapable computing environment.

Network based System Configuration

As we argued above, one of the main benefits of using software architecture is that the level of abstraction gives us the ability to use analytical methods to evaluate properties of a system's architectural design. To illustrate how this works, consider our example, where we have system in the application in a style [2].

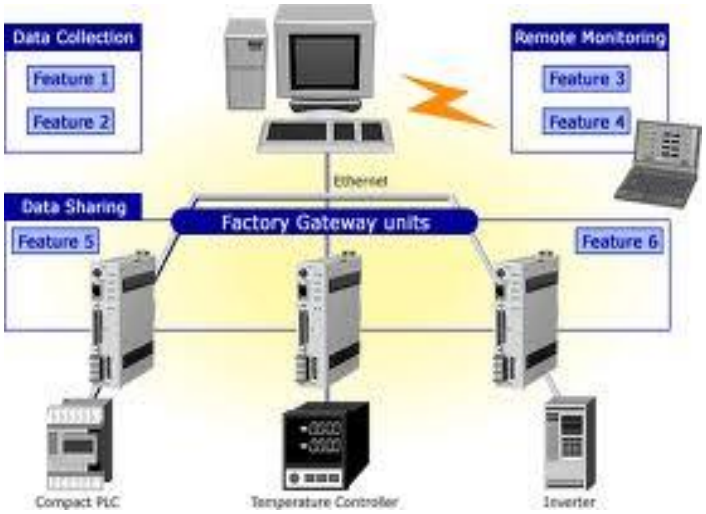


Figure: - Organizational Gateway

This analysis gives us parameters for a configuration of the architecture of the software that satisfies the above requirements. We use this information to configure the system to locate physical frisking, monitor the application to make sure it is in conformance with these requirements, and attempt to adapt the system transparently as the user moves about the environment. If the *Task Layer* changes the requirements, for example when the user begins using a large display, the analysis is performed again to determine a satisfactory reconfiguration of the system. Again, this can be done transparently.

Software Architecture for Adaptation Intelligent

As suggested above, these procedure and analyses need to be made available at runtime. This section discusses an augmentation to architectures that allows them to function as runtime adaptation method. This includes *adaptation operations*, based on the style of the architecture, to change an architecture and *repair strategies* that apply these operations to adapt the architecture. These operations need to be translated into operations on the runtime system. We consider the supporting runtime network based needed to make this work in practice.

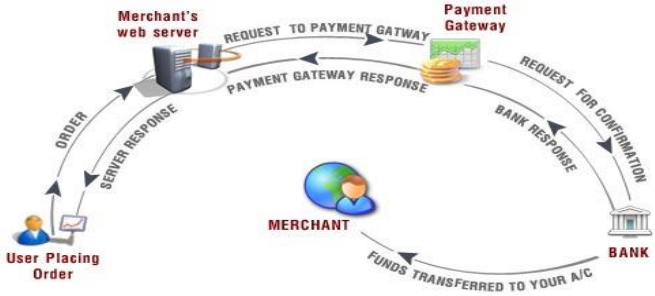


Figure: - Gateway between client and server

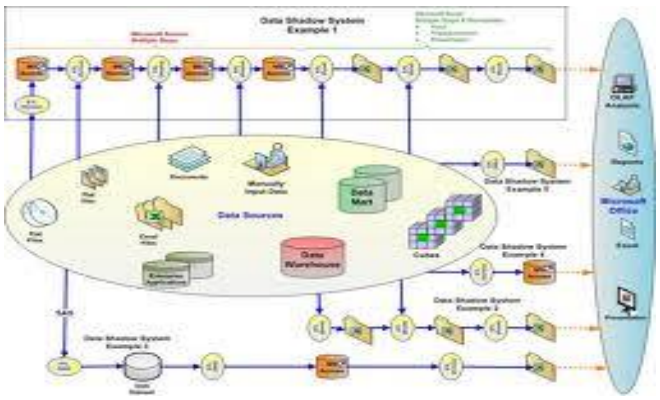
Monitoring

In order to provide a bridge from system level behaviour to architecturally-relevant observations, we have defined a three-level approach illustrated in Figure. This monitoring infrastructure is described in more detail elsewhere [12]: here we summarize the main features.

For instance, in the example we are concerned with the average intermission of user/vehicle. To monitor this property, we must associate a gauge with the average Intermission property of each user/vehicle role. Each intermission gauge in turn deploys a delve into the implementation that monitors the timing of reply-request pairs. When it receives such monitored values it averages them over some window, updating the intermission property in the architecture system.

Implementation

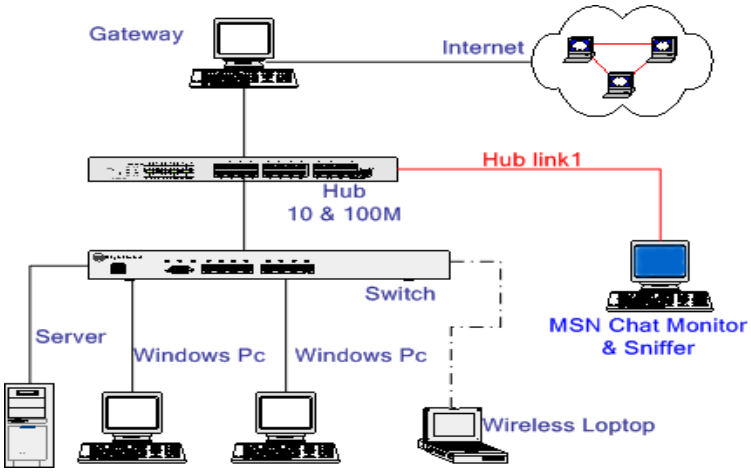
While the use of architectural systems allows us to provide automated support for intelligent adaptation at an architectural level, through use of constraints, operators, and analytical methods, we must furthermore relate system changes to the real world. There are two aspects to this. The first is getting information out of the executing system so we can determine when architectural constraints are violated. The second is propagating architectural repairs into the system itself.



Work on tools software architecture has mostly focused on design-time support. We have adapted these tools so that they can be used as runtime facilities.

Architecture environment can now make available an architectural description at runtime. This description can be analyzed by runtime versions of our constraint checking and performance analysis tools, as well as be manipulated by these tools.

In terms of monitoring, we have developed prototype delves for gathering information about networks, based on the system [19]. There are two parts: (1) an API, that allows applications to issue queries about bandwidth and intermission between groups of hosts; and (2) a set of *collectors* that gather information about different

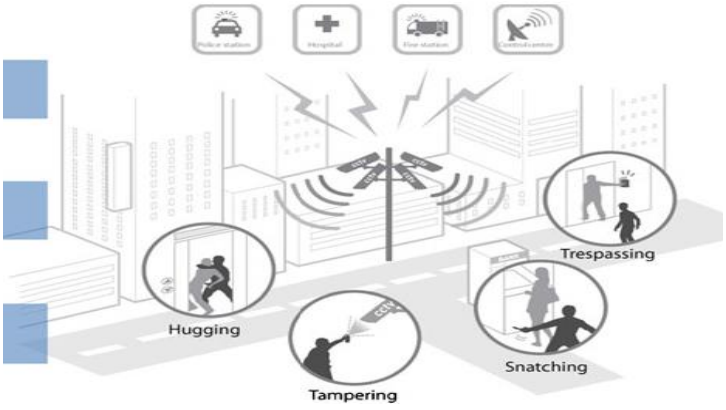


Parts of the network [21]. A delve uses remove to collect the information required for the delve and distributes it as events using wide area event bus [6]; gauges listen to this information and perform calculations and transformations to relate it to the software architecture of the system.

We are actively investigating effective means for specifying user tasks, as part of our broader research in the project for safe society at Manav Bharti University [27, 33].

Concerns

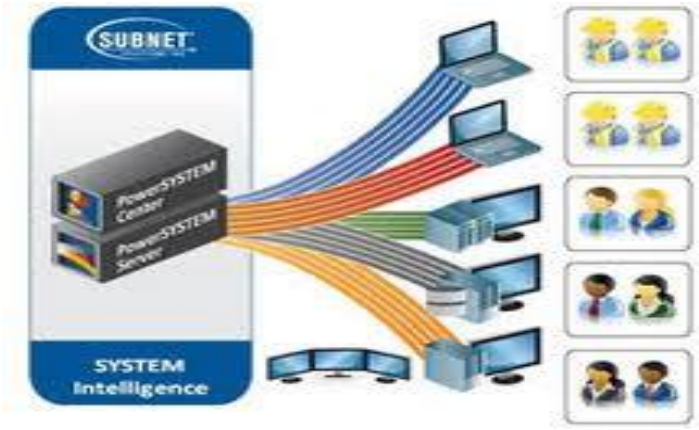
The concerns of dependability are the parameters by which the dependability of a system are judged. A dependability-centric view of the world subsumes the usual attributes of reliability, safety, and security (confidentiality and integrity). Depending on the particular application of interest, different attributes are emphasized.



Reliability

The reliability of a system is a measure of the ability of a system to keep operating over time. Depending on the system, long-term reliability may not be a concern. The reliability requirement is somewhat low in that it does not have to remain operational for long periods of time.

The reliability of a system is typically measured as its mean time to failure (MTTF), the expected life of the system.



Maintainability

The maintainability of a system is its aptitude to undergo repair and evolution. It is less precisely measured than the previous two concerns. MTTR is a quantitative measure of maintainability, but it does not tell the whole story. For instance, repair philosophy should be taken into account. Some systems are maintained by the user, others by the manufacturer. Some are maintained by both (e.g., the machine diagnoses a board failure, sends a message to the manufacturer who sends a replacement board to the user with installation instructions.

Safety

From a dependability point of view, safety is defined to be the absence of catastrophic consequences on the environment. Leveson [Leveson 95] defines it as freedom from accidents and loss. This leads to a binary measure of safety: a system is either safe or it is not safe. Safety is treated separately elsewhere in this report.

Confidentiality

Confidentiality is the non-occurrence of unauthorized disclosure of information. It is treated separately, in the "Security" section of this report.

Integrity

Integrity is the non-occurrence of the improper alteration of information. Along with confidentiality, this subject is treated separately in this paper.

Evolutionary Computing

Evolutionary computing comprises machine learning optimization and classification paradigms roughly based on mechanisms of evolution such as biological genetics and natural selection. The evolutionary computation field includes genetic algorithms, evolutionary programming, genetic programming, evolution strategies, and particle

swarm optimization. It is known for its generality and robustness. Genetic algorithms are search algorithms that incorporate natural evolution mechanisms, including crossover, mutation, and survival of the fittest. They are used for optimization and for classification. Evolutionary programming algorithms are similar to genetic algorithms, but do not incorporate crossover. Rather, they rely on survival of the fittest and mutation. Evolution strategies are similar to genetic algorithms but use recombination to exchange information between population members instead of crossover, and often use a different type of mutation as well.

Scope of Future work

Selecting various graph models to represent topologies of computer network had always been an active research area for network Scientists [3]. The reasonable criteria behind this selection are very significant because this put long testing impact on the network. How to make a graph model an ideal choice to implement a specific topology would always impose a huge challenge to the upcoming Scientists. The chance of finding optimal, if not optimum, graph model with unparallel properties entirely depends on the institution of the Scientists, in-depth analysis of the graph and technical feasibility of implementing computer network using that graph. Finally, it can be said very intuitively that this field still holds immense scopes for Scientists for further research work.

CONCLUSIONS

In this paper, we have developed an intelligent system that provides various services, such as object identification, object authentication, object localization, tampering detection, activity recognition, and moving object tracking. Intelligent systems have different established event rules and message exchanging rules with network-based software architecture. Thus, we have defined the metadata rules to exchange analysed information between distributed systems or heterogeneous systems.

REFERENCES

1. Oreizy, P., Gorlick, M.M., Taylor, R.N., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., and Wolf, A. An Architecture-Based Approach to Self-Adaptive Software. IEEE Intelligent Systems May/June 1999.
2. Shaw, M., and Garlan, D. *Software Architectures: Perspectives on an Emerging Discipline.* Prentice Hall, 1996.
3. N. Medvidovic and R. N. Taylor. A framework for classifying and comparing architecture description languages. In Proceedings of the 6th European Software Engineering Conference held jointly with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich, Switzerland, Sep. 1997.
4. G. D. Abowd, R. Allen, and D. Garlan. Formalizing style to understand descriptions of software architecture. ACM Transactions on Software Engineering and Methodology In Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering (SIGSOFT‘93), Los Angeles, CA, Dec. 1993.
5. G. Andrews. Paradigms for process interaction in distributed programs. ACM Computing Surveys, Mar. 1991.