DESERT: a Decentralized Monitoring Tool Generator

Paola Inverardi and Leonardo Mostarda
University of L'Aquila, Computer Science Department , L'Aquila, Italy
(inverard, mostarda)@di.univaq.it

ABSTRACT

This paper presents the tool DESERT that allows the generation of decentralized monitoring systems for component based applications.

Categories and Subject Descriptors: D.2.5 [Testing and Debugging]: Monitors

General Terms: Security and Reliability.

Keywords: Distributed Monitoring System, Enforcement Mechanism.

1. INTRODUCTION

A monitoring system is a tool that: (i) gathers system information; (ii) interprets the gathered information; (iii) after interpretation can undertake a set of reaction policies. Modern monitoring tools are used to increase security [7], dependability [8] and performance [9]. They are also used for debugging and testing purposes. Moreover they can implement part of the system functional requirements (i.e., they can build a control mechanism [1]).

In this paper we present the DESERT tool that allows the generation of decentralized monitoring systems for component based applications. We allow the specification of a global state machine that models the correct interacions among components and we use the components' interfaces to derive one filter for each component. Each filter locally monitors violations of the global property and undertakes reactions accordingly. Different reaction policies can be defined. For instance, the isolation reaction policy is applied to a component that misbehaves for malicious purposes thus enhancing system security. Retry, rollback and rollforward can be used to recover from a component misbehavior thus enhancing system dependability. Logging and tracing policies can expose the system state and can be used for testing purposes.

While monitoring approaches tend to locally collect data and forward them to higher entities, in our approach each filter locally performs all checks. In particular filters only exchange a small amounts of control data to simulate the global state machine.

The advantage of our approach is its distributed nature. Filters allow the monitoring of complex applications (e.g., wireless sensor and *ad-hoc* applications) where there is no a centralized point to which all information flows.

ASE'07, November 5–9, 2007, Atlanta, Georgia, USA.
ACM 978-1-59593-882-4/07/0011.

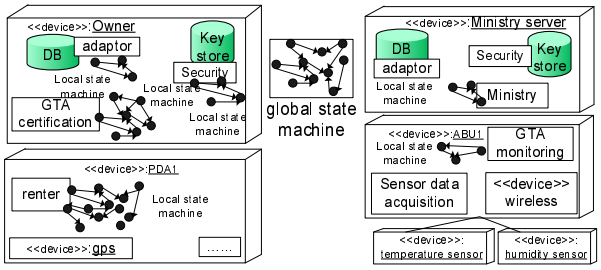

Figure 1: The CUSPIS system specification

The DESERT tool has been applied to generate decentralized intrusion detection systems for both mobile [6] and wired [3] infrastructure. It has been applied as an enforcement mechanism in the area of component based software engineering [5]. Recently, it has been used to enhance system dependability [4].

In the following we sketch all steps needed for a monitoring system generation, i.e., (i) component interface definition (i.e., system model definition); (ii) state machine definitions (i.e., property definitions); (iii) front-end application; (vi) back-end application. Each step is automatised through our tool.

2. THE DESERT TOOL

Component interfaces are described by means of an Interface Description language (IDL). Our IDL extends common ones (e.g. CORBA and RMI IDLs) with the definition of required services. Therefore, a component that always performs the client/server role defines only required/provided services while a component that performs both roles has both types of services. The user who specifies the monitoring tool can add *special* variables, procedure and function declarations to a component interface. Variables can be used for the property definition since they are an expressive mechanism to store system history. Procedures and functions can be used to update variables, can be used for the global interaction property definition and can be used to specify reaction policies.

We monitor four types of events: (i) client service invocation; (ii) server incoming request; (iii) server returned value; (vi) client incoming result[1].

[1]The black-box nature of the components means that our events can be only observable messages exchanged among them.

The correct sequence of events observed in the system (i.e., the global interaction property) is specified by means of state machines. As we show in Figure 1 our state machine language permits to define two types of state machines, i.e., local and global ones.

A local state machine is related to exactly one interface definition (e.g. C) and models the correct interactions of a C component type. A global state machine has the same syntax of the local one but can include events observed on different components. Each state machine can have a related time-out. This specifies the amount of time by which any transition must be applied.

In Figure 1 we show the CUSPIS deployment diagram [2] and both local and global state machines. This system is built to perform the secure transport of cultural assets. The ABU1 device contains the GTA monitoring component that monitors the sensors inside a cultural asset package. This check allow us to avoid having to unwrap the package. The local state machine associated to the GTA monitoring component: (i) checks that every k seconds the component reads the sensor data; (ii) ensures the consistency of the sensor data; (iii) ensures that data is forwarded to a central unit. The ministry, the renter and the GTA certification component produce digital certificates for the cultural asset transportation. A global state machine is used to gather events scattered over these three components, to correlate them and verify the correct certificate generation.

The DESERT component (see Figure 2) can be applied locally to each component host and is composed of a front-end and multiple back-ends. It takes as input a component name (e.g, C), the interfaces definition, the C-local state machine, the global one and outputs a C-local filter for a C-component type. The filters realise a distributed monitoring system that ensure the properties specified by the state machine. In particular each filter locally checks each local state machine and cooperates with the remaining ones to simulate the global one. It is worth noticing that while interfaces and state machines have to be specified by a user, the filter generation is completely automated by the front-end and the application of a back-end.

In the following we describe the application of DESERT front-end and back-ends to the *renter* component. The front-end (see Figure 2) is composed of the following components: (i) parser; (ii) semantic controller; (iii) local automaton generator. The parser and semantic controller performs all syntactic and semantic checks, respectively. The local automaton generator performs the following steps: (i) extracts from the global state machine a *ministry*-local one (in the following referred to as *ministry* local projection); (ii) enriches this *ministry*-local state machine with synchronization messages. The state machine obtained in (i) contains only events defining *ministry*-local interactions. In (ii) we enrich this state machine with synchronization messages in order to simulate the global one.

The local state machine and the local projection are implementation independent and can be translated in different filter implementations. We have implemented different back-ends (e.g., RMI , CORBA and so on) that take as input a component name and the related local state machines and output a filter implementation for a specific communication layer. For instance for distributed applications where the components communicate by using the CORBA middleware we have implemented a CORBA back-end. This back-end automatically produces a new CORBA component (the filter) that is interposed between the component communications and the environment. The filter exports all services required by the component it resides on. Moreover, it provides to the environment all services implemented by the component it resides on. In this way the component based application is not affected by the filter insertion.

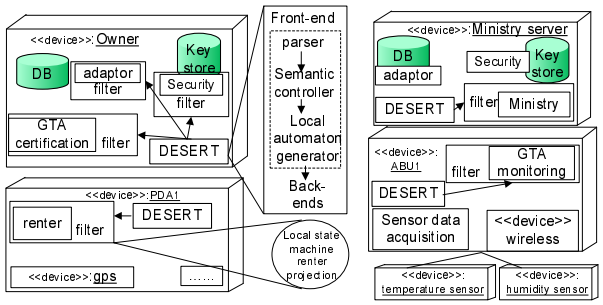

Figure 2: DESERT tool application

At run-time a filter captures all events locally observed on the component it resides on. It verifies that the events comply with both local and projection state machines and sends the synchronization messages (if any). In case of a property violation a filter can apply different reaction policies and can require cooperations with other ones to enforce the property specified by the state machines.

3. CONCLUSION

The DESERT generator allows the generation of distributed monitoring systems for component based applications. Local state machines permit the implementation of scalable monitoring systems suitable for mobile applications. The introduction of a global state machine allows us to generate a more effective monitoring system that correlates information scattered over several components.

4. REFERENCES

[1] N. Delgado, A. Q. Gates, and S. Roach. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools. *IEEE TSE journal*, 30(12), 2004.

[2] European Commision 6th Framework Program - 2nd Call Galileo Joint Undertaking. Cultural Heritage Space Identification System (CUSPIS). www.cuspis-project.info.

[3] P. Inverardi and L. Mostarda. A distributed intrusion detection approach for secure software architecture. In *EWSA*, 2005.

[4] P. Inverardi and L. Mostarda. A distributed monitoring system for enhancing security and dependability at architectural level. In *book chapter of Architecting Dependable Systems IV*, 2007.

[5] P. Inverardi, L. Mostarda, M. Tivoli, and M. Autili. Automatic synthesis of distributed adaptors for component-based system. In *ASE*, 2005.

[6] L. Mostarda and A. Navarra. Distributed inutrusion detection systems for enhancing security in mobile wireless sensor networks. *Internation Journal of Distributed Sensor Networks.*, 2007.

[7] J.-M. Orset, B. Alcalde, and A. Cavalli. An EFSM-based intrusion detection system for ad hoc networks. In *ATVA 2005*.

[8] K. Sen, A. Vardhan, G. Agha, and G. Rosu. Effecient decentralized monitoring of safety in distributed system. *ICSE*, 2004.

[9] M. Taufer, P. Cicotti, , and A. Chien. Dgmonitor: a performance monitoring tool for sand-box based desktop grid platforms. *In PMEO-PDS 2004*, 2004.