A TTCN-3 Based Online Test and Validation Platform for Internet Services

Peter H. Deussen, George Din, Ina Schieferdecker
Fraunhofer Research Institute for Open Communication Systems (FOKUS), Berlin, Germany
E-mail: {deussen, din, schieferdecker}@fokus.fhg.de

Abstract

Management of services over large distributed systems together with the growing complexity of underlying technologies has become a challenge in the present time.

Faults can occur anywhere in a distributed system. Online validation addresses the activity of evaluating the behavior of services and service environments in the production phase in order to determine functional faults immediately.

In this paper, we present our work on the design and implementation of an online validation platform. We describe a generic architecture, which can be instantiated for implementing validation systems for any type of distributed environments. We also present our experiences in using TTCN-3 to implement an online validation system for Active Network environments.

Keywords:
Online Validation, Active Networks, Online testing, TTCN-3

1. Motivation and Objectives

Nowadays, underlying technologies such as .Net, COM, EJB, or approaches on active networking used to develop business applications tend to hide the complexity of resource and infrastructure management behind generic and standard environments, where well-known services are deployed. The advantages of these approaches are obvious; they provide component architecture frameworks for creating distributed n-tier middleware, flexible management solutions and offer mechanisms for component based development.

In the context of designing, developing, deploying and managing services in such business-oriented platforms, the principal challenge is quality assurance. The complex interactions between platform components can obviously lead to unpredictable faults in the system. Malfunctions or failures within any component of the platform can always occur, which may affect the system functionality. Runtime configuration capabilities and programmability do not only increase the flexibility of system components, but may also introduce functional and performance problems that occur in the system's production phase and cannot be detected and avoided by testing during the development phases.

In this work, we address the problem of quality assurance by means of online validation[1] of deployed services within their runtime environment. We describe the elaborating a concept space and our framework for online validation. Online validation encompasses, on the one side, the concepts of continuous monitoring and checking run time functionalities and quality parameters of services and, on the other side, the control and coordination of target system activities - by acting on the fly in the system behavior or underlying platform.

Monitoring and validation is a challenging task as a service passes through several steps during its deployment live cycle: release, install, update, reconfigure, adapt, activate, deactivate, remove, and retire. Any of these steps requires control or at least monitoring. The distributed application components of the system to be validated are modified to produce events of interests at critical points, which are considered at different levels of inspection. The validation system collects, manages and interprets the system behaviour according to predefined patterns of events derived from the specifications of the system that is being monitored. The patterns of the events are described by means of online test cases.

Controlling on the fly the behavior of the target system (and, implicitly, of its services) is a new idea and inquire additional support and mechanisms for intervening into the system and service activities with the purpose to correct any observed fault.

Therefore, we concentrate our research on concepts for online validation and the design of an online validation system being able to automate the monitoring and controlling of services. We do not aim to develop a particular application but to define a generic online validation platform. Such a platform comprises two types of components: core components that are common for each instantiation of the platform and application specific components that are determined by the particular demands of the system under validation, and by the particular validation purposes.

Having a platform architecture, suitable technologies for the realization of concrete online validation systems

[1] Validation as opposed to verification means to check user requirements of systems and services with sampling techniques such as testing and not with mathematical proofs.

Proceedings of the Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)
0-7695-1876-1/03 $17.00 © 2003 IEEE

have to be identified. TTCN-3, standing for *Testing and Test Control Notation (3rd edition)*[2] [25] is novel in the area of online validation. TTCN is widely accepted as a standard for test system development in the telecommunication area. TTCN-3 comprises concepts suitable to all types of distributed system testing. Due to this generality, we started the experiment to "abuse" the notation for online testing purposes.

Clearly, as there are other approaches similar to the work presented here, TTCN-3 is not the only candidate to define and implement components of an online validation system. However, as being highly extendable, TTCN-3 turns out to be very suitable to "glue things together". We used several other components: XML based data storage, Java Server Pages, communication middleware, etc. Specifically, we selected a target system under consideration: the Caspian Active Network Environment developed in the EURESCOM project P926 [16]

This paper is organized as follows. Section 2 gives a brief overview on related work. Section 3 defines the online validation platform: concepts of online validation are discussed in Section 3.1, architectural components are described in Section 3.2, while Section 3.3 explains the proposed data flow of the platform. Section 4 gives a brief introduction into TTCN-3 and how we used it for online validation purposes. A case study that we performed is described in Section 5. Section 5.1 deals with the Caspian Active Network Environment. Section 5.2 describes the instrumentation that is used to make Caspian ready for online validation. Section 5.3 lists briefly other technologies that were used. Experiences and further applications are described in Section 5.4. Conclusions are drawn in Section 6.

2. Related Work

There have been many efforts on monitoring software for different distributed systems at application level or very often at the network level. A number of network monitoring tools integrate sophisticated measurement or probing mechanisms but the common problem is that they use proprietary architectures and analyzing contexts, and none of them can be adopted as general solution for a larger area of validation aspects. A more general and generic solution is presented in DMF [9] where an architecture for distributed monitoring environment in so-called grid systems is presented. The aim of this work on designing a monitoring system is to detect faults and to enable performance analysis. The main components of the monitoring system are identified and the requirements for such a system are addressed. A complete solution based on SNMP is presented in [22] where besides the issues related to the architecture and measurement strategy the scalability and performance are addressed.

A management application for the faults in distributed systems at application level is presented in [15] A generic policy-driven fault management architecture is introduced that is able to detect, isolate, locate and correct faults.

The JAMM monitoring system, presented in [24] is an agent-based system that automates the execution of monitoring sensors and collection of event data. The agents proposed here are able to launch system or network monitoring tools meant to extract, summarize and publish the results.

In the Cadenus project [3] an integrated solution for the creation, provisioning, composition and validation of services is discussed and presented.

OLIVES, the P1108 EURESCOM project where this work was partially performed, aims at building an online validation system for a variety of services: probing, actuation, distributed communication, data analysis and correlation support, policy based control etc. [7]

3. Platform Architecture

Our goal is to design and build a platform for validation and management of a distributed system. Therefore, we firstly define the principal concepts related to online validation. Secondly, we present the components and the services associated within the platform.

3.1. Conceptual Framework

At first, it is important to define the notion of validation. Validation is the process of evaluating a system or a component during the development process or the production phase to determine whether it satisfies specified requirements. Actually, validation involves three main tasks:

- *Observation*: This task regards the activities to collect information about the status of the validated system.
- *Testing*: Here the informational units collected by probes are evaluated. Since those testing activities are quite similar to usually testing techniques, we call this type of evaluation *online testing*.
- *Reaction*: Online Validation deals with dynamic changes both in the structure of the target system to be validated and of the validation system itself. Whenever an error, malfunction, or inconsistent state arises and is detected by an online test case, a pre-established reaction must be triggered.

The validation activities act upon the *target system,* which is in the testing area usually called SUT (system under test). Any test performed on the SUT is based on structural and functional specifications of the SUT.

[2] Formerly called *Tree and Tabular Combined Notation.*

Classical testing considers the SUT to be a black-box, i.e. the internal structure of the SUT is unknown. We take a grey-box approach. The SUT is considered to be a collection of interacting *components under test* (CUTs). The interaction, communication, synchronization, etc. between those components is at least partially observable via so-called points of control and observation (see below).

In this paper we deal with validation of services in an Active Network environment. Each service is deployed into a distributed execution environment. The unit of testing (i.e. the CUT) is the service. Note that this definition implies that a service may consist of several (distributed) interacting components.

The life of a service can be represented as a collection of *phases.* Changing from one phase into another phase consists of a number of *steps*. With each step, an event is associated that indicates that a particular step has been performed; by these events, the execution of steps gets visible to the online validation system. Steps are not necessarily ordered in a linear fashion, since we consider validation in concurrent and distributed environments, the ordering can be partial: several steps may co-exist.

Analysis may require keeping internal states (i.e. cannot always be done memory-less), and the integration of external components as well. An example is a data base used as background storage for probed data and structural information about the target system. To enable those external components on a descriptive level, we do not require that steps are strictly associated with CUTs, but also with other components of the test system (clearly, local memory can also be considered as a "memory component" defining update and lookup operations).

A *mission* is then defined to be the description of the phases and steps of the life cycle of a service. Missions are used to describe the (observable and testable) behaviour of a collection of CUTs that is considered during the online testing process.

PCO (*point of observation and control*) is another concept borrowed from the testing area. In these points, probes or actuators are deployed, which aim at monitoring and controlling the target system. Conceptually, one can identify a number of PCOs:

- *Internal points*. These observation points are placed inside the services and reports information related to the activity and states within the service itself. Note that the usage of internal points does not violate the grey-box paradigm. Internal points are considered to be realized by special test or maintenance interfaces and therefore, behave in a specified, implementation independent way.
- *Border points*. Such observation points are used to monitor the interaction with other components, i.e. the activities that take place at the normal communication interfaces (ports) of a service. This concept subsumes *link points*, i.e. monitoring devices that observe the data flow on links between components.
- *External points*. Deployment and removal of services are also in the scope of online validation. However, neither the concept of internal points nor that of border points can be applied because the service may not yet exist at all or may be inactive. External points are PCOs that are associated with the execution environment of a service. Successful deployment, removal, or liveliness of services are considered to be controlled (or at least observable) by its execution environment.

3.2. Architectural Components

Our platform is based on the architecture presented in Figure 1. As illustrated, the architecture groups the components into two layers:

- *Core components*. At this level the data is analyzed and validated. The components communicate via a communication bus according to a standard interface called ICP (Internal Communication Protocol).
- *Target system specific components*. Probes and actuators act at this level. They are directly associated to the target system, since it is not achievable to define a standard communication interface with the target system.

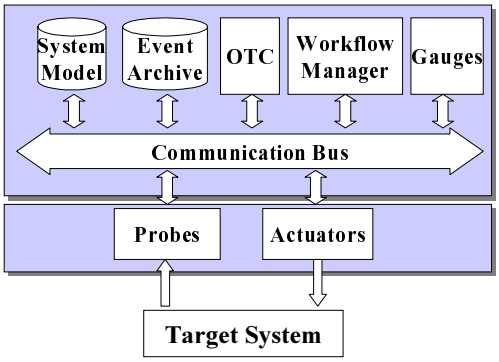

Figure 1. Online Validation Platform Architecture.

The components are in detail as follows:

Probes. Probes are dedicated components able to extract information about the SUT. They collect small units of information associated with each step in mission of a service and map them into a high-level data format understood by core components.

Event Transport Layer. The components enabling the validation platform communicate by sending and receiving events according to a standard communication

protocol. The circulated events fit into a standard data format, which is defined in XML.

Online Test Cases (OTC). The test cases, which are here obviously called online test cases, correlate and analyze the informational units probed in the target system, i.e. map these events into the mission description of a service under test.

Event Archive. It provides a storage support of the information received from the target system. This component offers services to create tables, lookup, and store or update data to any other component joining the communication bus. Usually the test cases need access to large amounts of data in order to perform more complex analysis.

System Model. It represents the knowledge about the target system and includes the administration policies according to which, the validation system functions.

Gauges. Gauges are visualization subcomponents, which subscribe to the event transport layer and receive also updates of the status of the target system. Their main goal is to provide the user (e.g. a system administrator) with the ability to interpret and explore the data collected about the target system. They are also able to interact with the Event Archive or System Model. In the Event Archive, they retrieve the data stored by test cases for historical or statistical analysis. The System Model provides information about the structure of the target system (network nodes, services in operation, etc.).

Workflow management. It controls the activity within the validation system. This functionality is driven by administration policies. It includes also mechanisms to react on faults detects during execution of online test cases. We call these mechanisms *reaction schemas.* A reaction schema is a high level representation of actions planned to be executed in the target system in order recover or correct its activity.

With the help of actuators, the workflow management system is able to interact with the target system to execute (apply) a reaction schema. The high level actions are transformed within the actuators into target system specific commands.

Actuators. Similarly to probes, actuators are embedded in the target system and perform predefined actions to repair or recover the services.

3.3. Data Flow

This subsection describes the processing levels and the information flow.

In the start-up phase of the online test system, the workflow manager reads and applies the policies stored in the system model. On top of these policies, the system decides upon further actions within the validation system. Besides these policies, the workflow manager will load from the system model also a number of configuration parameters related either to the validation system itself (information about probes or actuators) or to the target system (the location of execution environments, communication ports, etc.).

After the startup step, the system is able to collect information about the target system through the probing system. In particular, deployment and removal of services will be detected. This information is delivered by means of events, which are further distributed to the core components of the system via the communication bus.

Whenever a new service is deployed, the workflow management system checks its policies to determine if the service should be monitored or not. If the decision is to monitor that service, than the necessary probes, actuators and test cases will be deployed inside the validation and the target system.

After this step, the probes injected into the target system start keeping under observation the service for which they were deployed and deliver information about the service status to the core components. The reported information is encoded in form of internal events. A large number of event types coexist, but the specification format we designed is a general one and allows us to encode any type of information related to the target system.

Further, the test cases collect and correlate the events and extract the informational units encoded by probes into internal events. We identify as informational units any type of information about the target system: deployment steps, status information or statistical information. The test cases are abstract descriptions at a high level of the behavior of the target system, i.e. of aspects of missions of services under test, mirrored here by means of internal events. Usually, the test cases need to store the informational units with regard to further statistical or historical analysis. Hence, the event archive offering services to store, retrieve or remove data is used. Beside the activity of retrieving and extracting information from the events, the most important task is testing of the data. The test cases are able to evaluate the data received by comparing it with preset constraints (stored in the system model or specified statically within the test cases), and register the occurrence of steps and the change of phases.

Whenever an error is detected, a new event is generated. The content of this event is an id, which stands for the type of the detected error. The error event is collected by the workflow manager, which uses the error id to determine the strategy to guide the target system back to an error-free state, e.g. by repairing (if possible) the fault or by degrading the system. This strategy, called reaction schema, is a high level description of the measures which must be taken to (re)configure the target system. The mapping of these activities to concrete actions is done by the actuators, which interact with the target system at interface level. With the help of actuators,

the validation system is able to direct the execution environments to configure, remove or install running services.

4. TTCN-3

TTCN-3 is a test specification and implementation language to define test procedures for black-box testing of distributed systems. Stimuli are given to the system under test (SUT); its reactions are observed and compared with the expected ones. Based on this comparison, the subsequent test behaviour is determined or the test verdict is assigned. If expected and observed responses differ, then a fault has been discovered which is indicated by a test verdict *fail*. A successful test is indicated by a test verdict *pass*.

TTCN-3 allows an easy and efficient description of complex distributed test behaviour in terms of sequences, alternatives, loops and parallel stimuli and responses. Stimuli and responses are exchanged at the interfaces of the system under test, which are defined as a collection of ports being either message-based for asynchronous communication or signature-based for synchronous communication. The test system can use any number of test components to perform test procedures in parallel. Likewise to the interfaces of the system under test, the interfaces of the test components are described as ports.

TTCN-3 is a modular language and has a similar look and feel to a typical programming language. In addition to the typical programming constructs, it contains all the important features necessary to specify test procedures and campaigns for functional, conformance, interoperability, load and scalability tests like test verdicts, matching mechanisms to compare the reactions of the SUT with the expected range of values, timer handling, distributed test components, ability to specify encoding information, synchronous and asynchronous communication, and monitoring.

A TTCN-3 test specification consists of four main parts:

- type definitions for test data structures
- templates definitions for concrete test data
- function and test case definitions for test behaviour
- control definitions for the execution of test cases

Considering the architecture of the online validation platform, selected components can be realized in TTCN-3

- the system model is realized by a model test component with a managing port to modify system model characteristics and with a coordination point to the main test component of the online test case in order to control the execution and assessment of online test cases

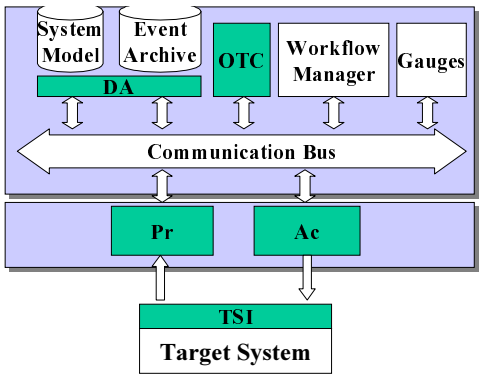

Figure 2. Online Validation Platform in TTCN-3.

The architectural components of the online validation platform described in Section 0 are realized as follows.

The test system interface (TSI) represents the access to the target system. TSI is a set of message-based ports via which stimuli are given to the target system (with send operations) and observations are made (with receive operations). There are two types of probes (Pr): push and pull probes. A push probe that is triggered by sets of observations about the target system is described by a loop on message receptions at specific TSI ports for the set of events being defined by so called templates. A pull probe that actively requests to receive certain information about the target system uses in addition a send operation to ask for that specific information. An actuator uses send operations to control resources in the target system. The system model and the event archive are both realized as XML databases while the database access including querying, adding or modifying data is realized with a database adaptor (DA) in TTCN-3, which uses send operations for database updates and receive operations for database queries.

The online test cases (OTCs) are essentially represented by a controller for the creation and execution of test components to fulfill specific test procedures. Additionally, each test component can initiate the creation of other test components to process subtasks if needed. In this way, online test cases can be hierarchically structured.

TTCN-3 turned out to be well suited for the realization of all components described above. Though, two major drawbacks were identified: TTCN-3 allows static test system interfaces in terms of number of ports only. Since an online validation system should be adaptable to the potentially changing configuration of the target system (e.g. in the case of Active Networks), a predefined system configuration as to be assumed. Even more, port arrays are not supported. It would be of great advantage to enable test system interfaces with several interfaces of the same kind. We investigate the extension towards port arrays, which then can be used to make the test system interface adapting to the target system:

```
type component TSI_type (integer n)
{
     ThePortType thePortArray[n];
}
```

Such an extension has to be carefully dealt with as component types could now be changed dynamically even during execution: therefore, the binding of the array parameter should be done during compile time.

Last but not least, the support of an object-oriented type system within TTCN-3 would have eased the representation of stimuli and observations to and from the target system – in particular, as the target system used itself object oriented data. Such an extension to TTCN-3 however will require further work: question to be investigated are e.g. how inheritance and pattern matching can be combined.

5. Case Study

We have implemented a prototype of this platform in order to experiment with the validation of services deployed in active networks. We describe Caspian (the adopted Active Network implementation) and discuss the implementation of adequate probes and actuators. At last, we present our experiences in designing online test cases.

5.1. Caspian Architecture

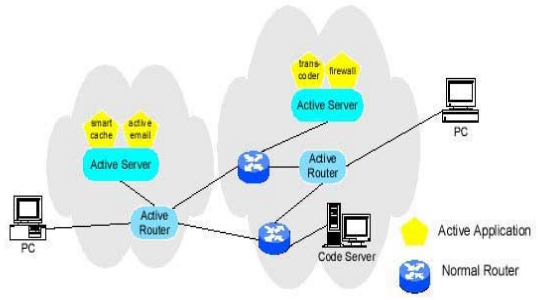

Figure 3. Caspian Architecture.

Caspian is the active network prototype developed in the EURESCOM project P926. The current implementation provides a flexible Linux based environment for the deployment of active services. The Caspian architecture, presented in Figure 2, is composed of active routers and active servers. Active Routers catch IP packets and filter them according to administrator defined filtering rules. Further, the packets are delivered to Active Servers with various running active services. An active service is notified by the Server whenever an IP packet requests it. The services are loaded on demand from a code server either at start-up or at runtime. The packets are sent back to the network if no adequate service is installed for their treatment.

5.2. Instrumentation for Monitoring and Actuation

Caspian had to be instrumented with new code to provide the collection of information on the status of services and to actuate the target system in case of errors. This new code implements several types of PCOs realizing these functionalities.

Probing subsystem. The probe architecture used for probing Caspian services is presented in Figure 3.

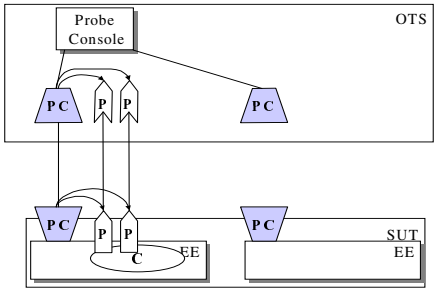

Figure 4. Probe deployment and communication with OTS

The Probe Console starts a Probe Controller for each Execution Environment (EE). Further, whenever a new component (C) is deployed, a set of probes is associated with this component is also deployed.

The probes, placed everywhere within the active services and active sever (here as execution environment), are coordinated by a probe console. Since Caspian involves the presence of many Active Servers, one probe controller per Server is needed. The probe controller is split into two parts, one for each side: validation platform and target system. The two parts of the ProbeController communicate via a probing management protocol. Whenever a new service is about to be deployed into the Active Server, the ProbeControllers interact in order to deploy new probes to monitor that service. The decision to deploy new probes belongs to the validation system, which applies this process according to user defined polices.

The deployed probes consist also of two parts (one for the validation platform and one for the target system) and interchange messages containing informational units.

Usually, a complex probing system requires the use of many types of probes specialized for certain probing tasks. Our system provides therefore a generic mechanism for deploying probes of any type. Moreover, this mechanism is able to deploy probes on demand, any time

during the probing task. Such support is important since the probes have not to be deployed at the beginning, but can be deployed later to execute more refined observations.

For this prototype we used two types of probes:

- Status probes (located at external points) – These probes report the status of the service during its live cycle (e.g. configured, deployed, running, terminated, removed, …).
- Smart probes (located at border points) – These probes are more intelligent and able to perform some computation tasks inside the target system. We used such a probe, for example, to monitor an Active Firewall service, where we need information about the number of blocked or passed packets. The usage of smart probes allows reducing time consuming interactions with the online test system.

Actuation subsystem. An architecture similar to the probing subsystem was used for the actuation subsystem. The actuators are here also coordinated by a console. For each Active Server we deploy an actuator controller (also split into two parts). The controller pair deploys actuators whenever the workflow manager finds an adequate reaction schema to repair, recover, or downgrade the broken service. The implemented actuators are specialized recovery tasks able to recover (if possible) or to reinstall a service.

5.3. Other technologies

TTCN-3 is not the only technology used for our case study. In fact, we found that TTCN-3 makes it easy to incorporate different technologies and unify heterogeneous components on the level of interaction.

For the system model and the event archive we used XML technologies. For parsing XML messages we used JAXP implementation of DOM and JAXEN to perform Xpath queries for data selection.

The communication among different software components enabling the test system is implemented on top of Siena, a scalable event-based middleware and stand-alone Java process that manages the distribution of events to interested parties.

With the help of Cougaar, an open-source Java-based decentralized coordination infrastructure, we implemented the Workflow management.

Gauging is achieved by using the Jahia Web Portal. Several small JSP pages inside the portal help to visualize the most important events related to the activity in the Active Node. The data is requested from the system model base by using Servlets and Java Beans.

5.4. Experience and new opportunities

As a first example application of our online test system, we defined and implemented a setup to check the successful deployment and removal of an Active Firewall service of Caspian. The deployment of an active service in the Caspian system is done by performing three steps: (1) retrieving a configuration file containing a firewall code location, (2) determining the validity of this code location, and (3) retrieving and startup of the active service.

Another task was to verify the consistency of firewall configurations. Rules defining the configuration of a firewall may be conflicting. For instance, blocking all UDP traffic but allowing HTTP interactions is inconsistent because HTTP is defined on top of UDP.

Clearly, performing tests like the above requires the analysis of incoming and outgoing traffic. An approach that mirrors this traffic and sends it to the online test system will fail because of insufficient performance. Instead, we used smart probes that count blocked and passed traffic of packets that fulfills predicates that were derived from rules of the firewall configuration files. The predicate (a pattern similar to tcpdump filters) was configured during the deployment process of probes.

To capture malfunctions in the deployment or removal process and inconsistent firewall configurations we implemented a single reaction scheme: the removal of the original firewall and the fallback into a safe configuration by deploying a “safe”, i.e. all-blocking firewall.

Of course, tasks like this could also be validated offline, but online validation allows for comprehensive and easy to implement checks and for immediate recovery. More challenging applications include for example:

- Intrusion detection. In [19] characteristics of attacks on e.g. enterprise networks are described. Intrusion attack detection and prevention bases on traffic analysis on packet level. Adding a system that is able to perform policy based correlation of data that is collected on different points of the network helps to evaluate the quality of intrusion detection systems in the context of the protected network and allows for forensic analysis. Note that the online test system can act on a larger time scale than the system under test.
- A promising idea presented in [10] that is closely related to online network analysis is a technique of packet trajectory sampling. Using this technique, it is possible to trace the actual path that a single packet took through a network, however, only with a certain probability. A hash function is used to identify those packets instead of direct marking. Trajectory sampling allows for performance testing, functional protocol testing, reliability validation, etc.

6. Conclusions

In this paper we presented a generic architecture for an online validation platform. We described the functionality of each component and presented how the components interact via events of a general data format. We also discussed how this architecture can be instantiated by presenting an experimental prototype for validating services deployed in Active Networks.

The main technology used is TTCN-3, which helps us to describe at an abstract level the functionality of the components. We found that although many concepts of TTCN-3 are tailored to offline testing, provisioning of parallel test setup, test data classification capabilities, and procedural syntax make TTCN-3 also suitable for the definition and implementation of online test systems. External functions and attributes allow the integration of external components from other technologies. Finally, due to the high degree of abstraction provided by TTCN-3, development of elements like online test cases can be done in a generic, comprehensible way.Major problems that we found were:

- TTCN-3 supports the representation of heterogeneous test data by means of unions. This makes the description of message content that is of variable type and size sometimes very clumsy. An object oriented mechanism would be more useful here.
- As already noticed, TTCN-3 supports only static test interfaces.
- Although this is not directly a TTCN-3 problem (as the TTCN-3 standard leaves this point open), current TTCN-3 implementations lack a deployment platform for distributed setup (for our case study we had to find a work around for this problem). In particular, the dynamic deployment of test components (i.e. of online test cases) is not supported. That means that we have to tear down the online test system and to reconfigure it offline in order to introduce new online test components.

Acknowledgment

This work was partially supported by the EURESCOM project P1108 [17] (OLIVES).

References

[1] Aide – Active Interface Development Environment, Website http://www.cs.wpi.edu/~heineman/dasada/

[2] G. Alonso, "Workflow Assessment and Perspective", in International Process Technology Workshop, 1999.

[3] CADENUS: Creation and Development of End-User Services in Premium IP Networks, Website http://www.cadenus.org/

[4] Cognitive Agent Architecture (Cougaar) Open Source Website, http://www.cougaar.org/index.html

[5] Corba Component Model, OMG Standard, Version ptc(99-10-04, 1999; information avail. at http://www.eurescom.de/public/projects/P900-series/p924/default.asp

[6] Dasada Home Page. http://www.if.afrl.af.mil/tech/programs/dasada/

[7] Deussen, P., Valetto, G., Din, G., Kivimäki T., Heikkinen, S., Rocha, A., Continuous On-Line Validation for Optimized Service Management, accepted for SUMMIT 2002, Heidelberg, 2002.

[8] Document Object Model (DOM) Level 1 Specification, Version 1.0, W3C Recommendation 1, 1998, avail at. http://www.w3.org/TR/REC-DOM-Level-1/

[9] Distributed Monitoring Framework (DMF), Website http://www-didc.lbl.gov/DMF/

[10] N. G. Duffield, M. Grossglauser, "Trajectory sampling for direct traffic observation", SIGCOMM, pp. 271-282, 2000.

[11] Dynamic Assembly for System Adaptability, Dependability, and Assurance (Dasada), Website http://www.if.afrl.af.mil/tech/programs/dasada/

[12] Jahia Portal Server 3.0, Website http://www.jahia.org

[13] Java Beans Component Architecture Documentation, avail. at http://java.sun.com/products/javabeans/docs

[14] Jaxen - Java Xpath Engine, Website http://jaxen.org

[15] M.J. Katchabaw, H.L. Lutfiyya, D. Marshall, and M.A. Bauer, "Policy-Driven Fault Management in Distributed Systems", Proc. of the 7th Int. Sym. on Software Reliability Engineering (ISSRE '96)

[16] P926 Project Page at Eurescom, Website http://www.eurescom.de/public/projects/P900-series/P926/default.asp

[17] P1108 Project Page at Eurescom, http://www.eurescom.de/public/projects/P1108-series/p1108/default.asp

[18] .NET, information avail. At http://msdn.microsoft.com/net

[19] T.H. Ptacek, T.N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", 1998, avail. at http://citeseer.nj.nec.com/ptacek98insertion.html

[20] Scalable Internet Event Notification Architectures (Siena) Website, http://www.cs.colorado.edu/users/carzanig/siena/

[21] Stump, R., W. Bumpus, Foundations of Application Management, J. Wiley & Sons, 1998.

[22] R. Subramanyan, J. Miguel-Alonso and J.A.B Fortes, "A scalable SNMP-based distributed monitoring system for heterogeneous network computing", IEEE 2000.

[23] D.L Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden, "A survey of active network research", IEEE Communications Magazine, vol. 35, no. 1, pp. 80-86, 1997.

[24] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, M. Thompson, "A Monitoring Sensor Management System for Grid Environments", Proc. of the 9th IEEE Int. Sym. on High Performance Distributed Computing (HPDC'00)

[25] Tree and Tabular Combined Notation, version 3, ITU-T Recommendation Z.140, 2001 avail. at http://www.itu.int/ITU-T/studygroups/com10/languages

[26] XML Path Language (XPath), Version 1.0, W3C Recommendation 16, 1999, avail. at http://www.w3.org/TR/xpath