A Case Study of Dependable Software Upgrade with Distributed Components

Xueshan Shan and J. Jenny Li
Avaya Labs (formerly part of Bell Labs)
233 Mt. Airy Rd, Rm#2C07
Basking Ridge, NJ 07920-2336, USA
{xshan | jjli}@research.avayalabs.com

***Abstract*- Technology presented in the paper [1] allows validation of software architecture before component upgrades. This paper presents a case study of applying this method to the upgrade of a wireless monitoring system. The converged network of voice and data introduces reliability-critical applications to conventional IP networks. Examples of such applications include voice-over-IP (VoIP), messaging, call centers, etc. Voice networks traditionally operate with an availability of 99.999%. Software upgrade must be done on-line to maintain such a high availability. Our wireless mobile monitoring (WMM) system includes clients and five server components: WAP Gateway (WG), Push Proxy Gateway (PPG), Push Initiator (PI), Content Server (CS) and Fault Monitor (FM). Our experimental results validate the system's feasibility for on-line upgrade and show that our method is effective in providing round-the-clock wireless network activities while upgrading the network types or the application software.**

I. INTRODUCTION

The explosive growth of varieties of internet and intranet applications has driven technology development rapidly. New release and new version of software are typically rolled out every six months. Some very large and complex systems, such as telecom systems, must and typically do exhibit exceptional dependability. These systems are seldom totally replaced with a new system because of the increased likelihood of a lapse in service. Rather, systems are upgraded incrementally while operational, albeit this often involves large-scale software changes. It is especially important then to ensure that new or replacement components are ready for on-line installation before they are incorporated into an operational system. The method presented in [1] includes two major steps: model derivation and model validation. Our case study uses both steps.

Wireless Network Monitoring system monitors the wired network faults and reports them to wireless devices. Such system informs network managers of network failures and their corresponding network status data disregard the time and their location. It releases network managers from sitting in their office for long hours. Based on this general idea, we then started with a set of behavioral flows that we wanted to implement. The first scenario is that the network manager should be notified of any network failures at anywhere and anytime. The second scenario is that the manager should be given access to other network information after he receives the failure notifications.

After analyzing the flows, we came up with an architectural design of the software system. Later on, a first version of the system was implemented and the functional flows are demonstrated. Afterwards, when we were reviewing the system, the question of whether we can improve the development process rises. We decided to apply the method in [] to the development of the second version of the wireless monitoring system and compare their results.

II. MODEL CONSTRUCTION

First, we put the two flows into an ArchFlow [1] table as shown in Table 1 (a) and (b). "Step" indicates the sequence of the flow. "Entities" includes the components involved in the flow step. "Action" describes the flow step. The first flow describes the fault notification process. A Failure Detector monitors the network and detects failures. It then invokes a Push Initiator. Push Initiator in turn generates a Service Indication for Push Proxy Gateway to send to mobile devices. The second flow describes network manager fetching additional information from the monitored network. Once the manager enters the fetching command, WAP Gateway will response by propagating the request to WML Content Generator, which further relays the request to the Failure Detector. Failure Detector runs automatic network diagnosis and sends back more failure information.

Step	Entities	Action
1)	Failure Detector (FD), Wired Network	FD monitors the network and detects a failure.
2)	Push Initiator (PI)	Invoked when a failure is detected.
3)	PI	Generates a Service Indication (SI) with a URL included.
4)	Push Proxy Gateway (PPG)	PPG pushes SI to mobile devices
5)	Mobile Device	Sound alarm and display alert message

(a) The First Flow Table

Step	Entities	Actions
1)	Network manager	Notice the alert message
2)	Network Manager	Click on "URL fetching"
3)	WAP Gateway (WG)	Receives the URL and sends request to fetch data content, more network information.
4)	WML Content Generator (WCG)	Receives fetch request.
5)	WCG	Sends request to FD to get more detailed data.

6)	FD	Response to WCG.
7)	WCG	Send new information to WCG.
8)	WCG, WG	WCG sends the content to WG.
9)	WG	Gives the data content back to the mobile devices.

(b) The Second Flow Table

Initially, we drew a partial architectural diagram. Based on the original idea of using wireless device for network fault monitoring, we knew that we needed to draw two networks, wireless and wired ones, the gateway between them, and a network-fault monitoring component. A partial architectural diagram is given in Figure 1.

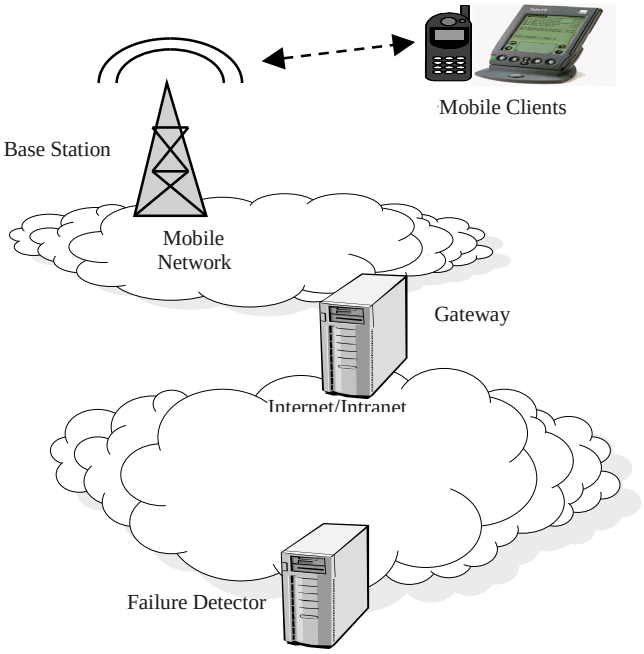

Figure 1: An Initial Partial Architecture Diagram

We ran the model constructor to add the flows and their corresponding entities to the architecture model. The tool first checks the flows against the partial architecture and adds the required components to the architecture. It then complete the specification for each component based on the information given in the flow descriptions. We obtained the following software architecture diagram of the system, Figure 2. Note that several additional entities are shown in the wired network. The actual tool-generated architecture model is given in an SDL-like syntax and semantics. A more formal architecture of the wireless monitoring system is given in Figure 3. Due to space constraint, the messages on each channel and the process symbols are not shown in SDL.

III. UPGRADE VALIDATION

The most commonly change part of Figure 2 is the type of the mobile network, 2G, 2.5G or 3G, and the Fault Monitor that has new releases every six months.

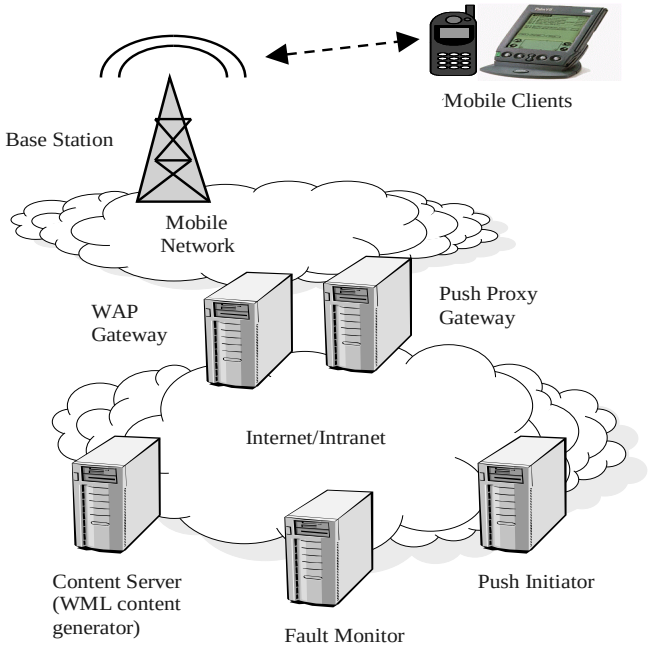

Figure 2: An Architecture Accommodating Both Flows

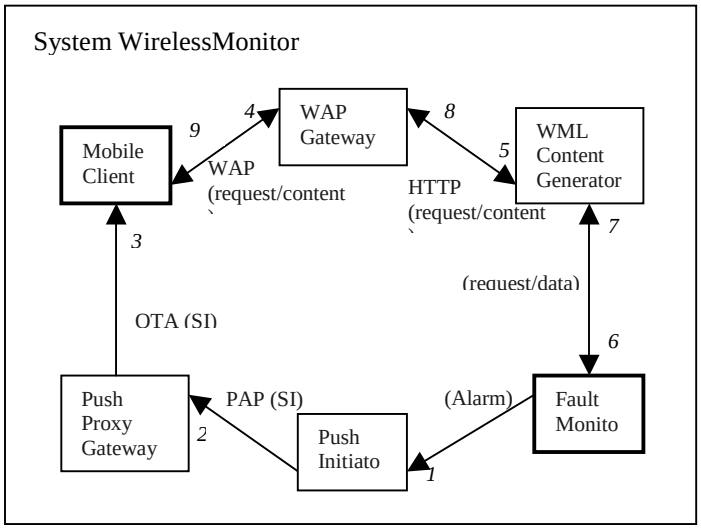

Figure 3: A Formal Architecture Model

After obtaining the architecture, we can use it to simulate and analyze the architecture with every upgraded FM. The architecture is modeled as a CEFSM system. Each component is modeled as an EFSM with the addition of behaviors from flows and execution activation signals. On the system level, the effect of flows can be seen in the addition of new channels that carry flow steps activation signals. In our example, we do not have broadcasting or non-FIFO type of signal communication.

The CEFSM model is represented in SDL. The high level SDL specification includes a set of Extended Finite State Machine (EFSM) communicating through channels. There are two types of channels: delaying channels and non-delaying channels. The flow step activation signals are sent through non-delaying channels unless the architecture

document states that there is a delay between two steps. The rest of the signals are sent through delaying channels because it is assumed that each EFSM resides at a different physical location. Such physical delay is non-negligible on this architecture abstraction level. Each component is specified in one SDL-Process. The input and output signal sets of an SDL-Process include both the original message communicating between the components and the flow step activation signals.

The behavior of each component and the attributes of each component, such as throughput, failure rate, availability, and price, are given as additional information in the architecture document. They are used to generate automatically environment and attribute simulators. The execution is based on an SDL engine.

We first employ a coverage-based model-checking approach that uses a dynamic slicing technique to guide a subsequent simulation. A conventional model checker verifies all the possible system states. Our method includes four steps:

Step 1: We first identify the "impacted model", i.e. FM, WCG and PI.

Step 2: We used a slicing technique to discover the most critical part of the impacted model to validate based on its contribution to overall simulation coverage. The transition in FM with the highest count is the critical point of the model.

Step 3: We manually generate simulation test cases to cover the critical parts of the model.

Step 4: After each simulation, the overall achieved test coverage percentage for the model was updated.

Next, we investigate an operational-profile-based approach for predicting overall system properties. Again, we use the specification-level dynamic slicing technique that relies on the execution of the specification to obtain simulation traces for the specification. Traces record the coverage repetition of basic transitions, i.e., how many times the same basic transition has been simulated. From this basic transition coverage, we can deduce the following five kinds of coverage: 1) basic transition coverage itself, 2) symbolic state coverage, 3) decision coverage, 4) component coverage, and 5) variable value coverage.

The operational profile provides us with data on the realistic usage of the target system. They can be used to guide the simulation of the model to determine the relative frequency of use for the various components or channels during typical operations. These usage frequencies indicate the relative contributions to certain system properties.

Lastly, we validate the behavior of the upgraded system. In the underlying operational system, each EFSM has one Input-Port that contains a FIFO queue of signals. The delay on channels is nondeterministic. They can be modeled as a queue where the departure of an element from a queue occurs at a random time. This queue is stored in the Path processor of the underlying SDL machine.

When the simulator receives a copy of the same input as the target system, it stamps the current time onto the input signal, and then gives it to the System processor. The System will decide whether to deliver it to the Path or the Input-Port. Signals will eventually be delivered to the destination SDL-Process. When an SDL-Process encounters nondeterminism, it will inform the System to duplicate a similar copy of itself carrying different behavioral belief. When the System Processor of ASDLM notices that there is no active SDL-Process at a certain moment, it checks through all the Input-Ports of the SDL-Processes. It picks up a signal with a smallest stamped time, sends it to the SDL-Process for processing, and updates the global timer to the time carried on the signal time stamp. Note that the signals in Input-Ports are arranged in descending sequence according to their time stamps.

Experiments were carried out using the simulator derived to assess behavior of the software architectural design. The experimental result shows that the information generated by the simulator can be used to validate the behavior and the attributes of an upgraded system solely based on its architecture documents.

VI. Conclusions

This paper presents a case study of applying the method in [1] to the validation of system dependability with component upgrade. Although the example application is an wireless network software, the projected results would be suitable for software systems in general, especially real-time systems requiring high availability.

In our experiments, we find the tools developed by [1] being useful. It helped to generate formal models to decide quickly whether a candidate component is a suitable upgrade. In all, these tools reduce the cost of component-based software modifications, because a go/no-go decision at the design stage may avoid the high cost associated with installing unsuitable components.

For the on-line upgrading activity, the experiments were carried out under normal network conditions. One possible future work is to test the method on more extreme conditions such as multiple simultaneous voice calls. For mobile notification, WMM monitoring system will be able to allow network managers to make direct calls using WTA to the technicians who are available in the geographical area that fault occurs and with the best-suited skills. WMM will benefit from the deployment of high-speed packet data networks. It will further reduce the time and the cost related to network repair and make WMM a more valuable technology to network managers.

References

[1] J. Jenny Li, Dennis Mulcare, and Eric Wong, "Dependability of Complex Software Systems with Component Upgrades", IEEE COMPSAC, Taiwan, Nov. 2002.