◆ Refactoring Administrative Services Module Platform Software

Ai G. Nguyen and Richard L. Sohn

A well-defined, fully backward-compatible application programming interface (API) allowed Lucent Technologies' administrative services module software to be transformed over six releases to incorporate architectural changes driven by market needs not envisioned in its original conception. We describe this software, the changing demands placed on it, how it was refactored, and our guiding strategies. © 2003 Lucent Technologies Inc.

Introduction

In late 1996, Lucent Technologies launched work on the administrative services module (ASM), an attached processor to the 5ESS® switch administrative module (AM). The purpose of the ASM was to expand the operations, administration, maintenance, and provisioning (OAM&P) capabilities of 5ESS® switch in ways not easily and inexpensively handled by the AM. It did this primarily by offering Internet protocol (IP) connectivity, expanded disk space, and larger processing power on a commercial platform (a Sun Microsystems* Netra* t1120). Since conception, it has undergone significant architectural changes to meet unforeseen market needs. Described herein are some key lessons learned from the first 6 years of the project.

Emergence of the ASM Platform

The ASM implements a 700 kb/s link to the AM using a proprietary interface known as the dual-serial-channel computer interconnect (DCI) on a custom peripheral component interconnect (PCI) card. The DCI offered the highest-speed input/output mechanism available on the AM.

Enhanced OAM&P capabilities supported by ASM included high-speed downloads of 5ESS® software updates, on-site evolution of switch data, and a high-speed provisioning interface. During the architecture phase, it became apparent that all features shared a common need for a set of messaging and file management services built on the DCI. Because this project was one of the first to bridge switching and commercial computing environments, it had to be concerned with interacting with 5ESS® system integrity code, gaining access to 5ESS® services, and achieving a degree of common look and feel. Rather than having multiple teams create similar code for these functions that required expertise outside their domains, it was decided to segregate these common functions into a platform, on top of which each feature became an application.

One of the initial charters for the ASM project was to explore new approaches to software that minimize time-to-market for switching system software. This meant delivering the ASM platform and applications in 18 months versus the 5ESS® norm of 3 years. This presented a fundamental problem because applications were fairly well understood, while the platform was only vaguely defined. Application development could begin, but there was no platform to run them on, which meant that developers

Bell Labs Technical Journal 8(3), 105–110 (2003) © 2003 Lucent Technologies Inc. Published by Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/bltj.10080

would have to do a lot of throwaway work just to get started.

Another component of the ASM charter was to find ways to apply conventional commercial software development practices to the specialized environment of a telecommunications system. In doing so, Lucent gained experience that was applicable to next-generation switching products such as softswitches [1].

In this crucible or early project start-up, a number of key strategies were developed that have served the ASM project well during its life and enabled it to adapt to new uses.

Application Program Interface

The fully backward-compatible ASM application programming interface (API) defined high-level system behavior and hid details of the operating system (OS) and hardware from applications. Creating stubbed versions of API libraries supported concurrent development of the platform and applications. Encapsulating segments of the platform into quasi-independent API sections facilitated integration with the applications. Backward compatibility was ensured through a stringent review process for API changes. The API remained a constant constraint throughout the subsequent refactoring of ASM platform software. **Figure 1** shows an initial decomposition of the ASM APIs.

Proofs-of-Concept

To mitigate engineering risk, all major development included a prototyping phase, where major blocking issues were discovered and circumvented. Results served as input to design and implementation, aiding in API evolution. A variety of techniques were employed to perform proofs-of-concept, such as creation of use-case scenarios (that subsequently drove product requirements), bare-bones implementation (that was converted into production code), and simulation models (that turned out to be worthy of productization). See discussion on the 5ESS® switch distinctive remote module (DRM) below. The specific tools and techniques used depended on the particular project, but the idea was to use readily available hardware and software components to simulate and test a simplified version of the desired end product without incurring the overhead of full production. Where third-party components were involved, our proofs-of-concept always included exercising these (hardware or software) components.

Panel 1. Abbreviations, Acronyms, and Terms

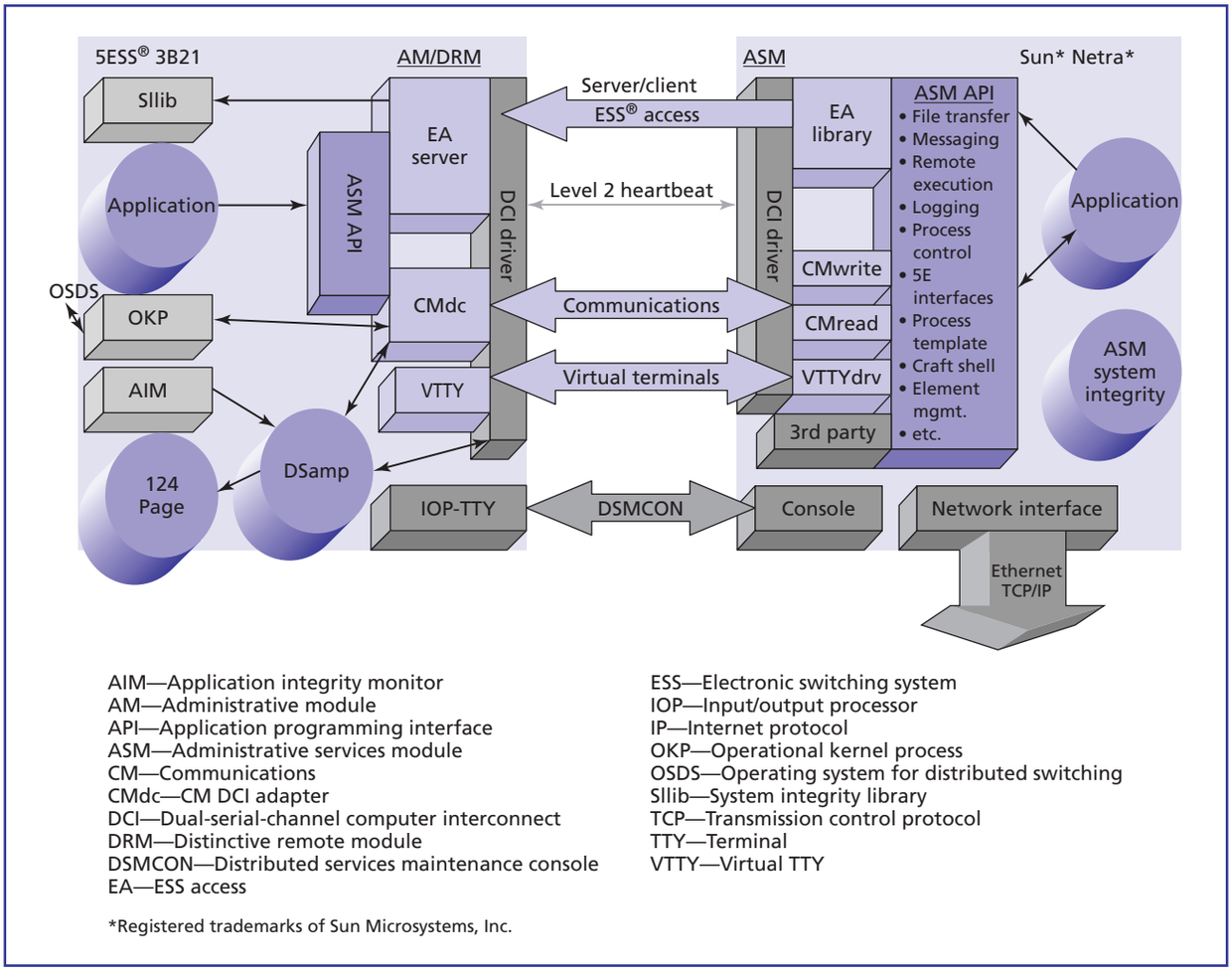
AM—administrative module
API—application programming interface
ASM—administrative services module
CORBA*—Common Object Request Broker Architecture
DCI—dual-serial-channel computer interconnect
DRM—distinctive remote module
EMS—element management system
IP—Internet protocol
OAM&P—operations, administration, maintenance, and provisioning
OMG—Object Management Group
OS—operating system
PCI—peripheral component interconnect
SI—system integrity

Incremental Development

To mitigate business risk, development was phased using "little gulps," because changing development direction after a single phase was likely to be less costly than changing amid a larger, monolithic project plan. The incremental effort had to remain consistent with funding and staffing availability.

A Unifying Vision

The concept was to have a unifying vision, but "to hold the implementation" until needed, i.e., to avoid future proofing until the envisioned future arrived and to simultaneously avoid design and implementation alternatives that were inconsistent with longer-term plans. An informal product and platform vision provided guidance to incremental development. Adherence to this principle helped the project to survive, while other follow-on products attempting to achieve a complete platform all-in-one-go never reached the market.



AIM—Application integrity monitor
AM—Administrative module
API—Application programming interface
ASM—Administrative services module
CM—Communications
CMdc—CM DCI adapter
DCI—Dual-serial-channel computer interconnect
DRM—Distinctive remote module
DSMCON—Distributed services maintenance console
EA—ESS access
ESS—Electronic switching system
IOP—Input/output processor
IP—Internet protocol
OKP—Operational kernel process
OSDS—Operating system for distributed switching
Sllib—System integrity library
TCP—Transmission control protocol
TTY—Terminal
VTTY—Virtual TTY

*Registered trademarks of Sun Microsystems, Inc.

Figure 1.
ASM functional decomposition.

Reuse of a Third-Party Application

Even before ASM had its initial market deployment, a critical new application feature stretched the ASM platform in a new direction, i.e., providing large amounts of disk space for storage of billing records. An existing Lucent product, BILLDATS™ billing and data system provided the bulk of the required functionality. By creating an adaptation module that used ASM APIs to provide coordination with the 5ESS® billing code, the existing product could be easily reused.

Refactoring for Distributed Operation

Market need for the DRM drove this phase. The DRM offers centralized OAM&P for one or more 5ESS® switch very compact digital exchanges (VCDX) switches located up to 2,000 miles away from the host 5ESS® switch.

A proof-of-concept preceded development of the virtual DCI (which layers the DCI protocol on top of a transmission control protocol stack). A subset of the API was refactored to accept a parameter identifying the destination switch (VCDX or 5ESS® switches). The unchanging API hid the complexity of the DRM network environment, making it transparent to existing (and future) applications, which were now able to communicate with the VCDX AM over virtual DCI, in addition to communicating with the 5ESS® AM over DCI. Enhancements made to a relatively small segment of platform code extended the capabilities of the rest of the platform and its applications,

providing evidence of good decomposition of platform functions.

Refactoring for Internationalization

This was a case where "holding the implementation" came into play. The initial deployment of ASM was for the North American market, but global deployments were anticipated. To keep start-up development costs down, the decision was made to defer investing in internationalization until there was a market for it. ASM-resident code was designed from day one to support global deployment; however, differences in 5ESS® software for international markets meant that the API needed to change to handle both variants.

The software design for API services took into account that most of the market-specific implementation was delegated to 5ESS®-resident code and hidden behind a common API. As was the case for the DRM, a relatively small number of changes to low-level APIs provided the necessary compatibility for the platform and applications.

Evolution for High Availability

Market need drove the creation of a high availability configuration that integrated duplicated hardware, as well as an array of redundant external disks, in an active/standby configuration. An active ASM provides read and write access to the external disks, controls input/output devices, and runs all applications. A standby ASM has read-only access to the external disks, is not visible to external systems, and maintains itself in readiness to take over in the event of a failure of the active system. A third-party software package was chosen as the high-availability middleware.

In this case, the project did not follow the proof-of-concept strategy as thoroughly as in earlier phases, due to scheduling constraints. A bare-bones implementation of the proposed architecture would have identified challenges for the selected middleware to meet our requirements. Because these challenges were not identified earlier, ill-informed design and architecture decisions caused significant platform churn during development, as well as caused schedule slippage. Ultimately, refactoring and enhancement of the ASM system integrity (SI), which resides on top of the middleware, made the middleware transparent to applications.

Refactoring of ASM databases also contributed to transparency. Where appropriate, application data files were migrated to the shared external disks. In other cases, the API was enhanced to distribute data updates across the duplicated hardware.

Inheriting from 7R/E® Switch for an Element Management System

Market need for ASM to manage the 7R/E® switch packet network drove this phase of ASM evolution. The 7R/E® architecture envisioned ASM and the OneLink Manager™ element management system (EMS) as coexisting element managers. [2] ASM was responsible for OAM&P for 5ESS®, DRM, and several packet switching components. Common Object Request Broker Architecture (CORBA)* and simple network management protocol (SNMP) packages were required for communication with OneLink Manager™ EMS and managed elements.

Proofs-of-concept were planned, but never completed for CORBA communication with OneLink Manager™ EMSs, due to a decision to discontinue that project. As a legacy of this effort, the OneLink Manager™ EMS graphical user interface (GUI) remains a part of the ASM platform software today. Proofs-of-concept and development were completed for SNMP communication with packet components. However, this part of 7R/E® development was discontinued before field deployment.

Although these "little gulps" were not entirely swallowed, our experience indicated a need for an abstraction layer between applications and externally obtained protocols suites. Ideally, applications should not be aware of the communication protocol (CORBA, SNMP). We discouraged (but did not prevent) applications from coding directly against the third-party APIs, since this would limit future refactoring and would increase the risk of vendor lock-in. Methods for enforcing for proper API usage remain an active topic for research.

Life-Cycle Management

Because ASM depends on numerous third-party components, the continued availability of these components over time (their product life-cycle) became an increasingly critical issue. Life-cycle management activity was driven by Sun's plans to discontinue support for the existing hardware and OS. This phase was split into two smaller gulps. The first gulp focused on changing the OS, while the second gulp focused on adapting to the hardware. The ability to split the project allowed completion of these little gulps in synchrony with 5ESS® release schedules.

Although significant differences existed between the old and new hardware, the ASM API hid such details from applications, while still maintaining transparency. These experiences suggested a new strategy, i.e., a platform needs to provide some degree of hardware and OS abstraction even when the vendor claims backward compatibility.

Life-cycle management also was an issue for other third-party software. Over time, multiple vendors chose to discontinue support or increase support costs for software on which the ASM depended. In response, for example, we eliminated dependence on a commercial Web server by deploying the open source Apache Web server in its place.

Our CORBA vendor discontinued support for their product, which was used by the ASM GUI. Lack of backward compatibility in their new product forced a costly redesign. This further underscored the need to insulate applications from the details of third-party software, since vendors tend to benefit from shorter lifecycles, while Lucent customers require more stability in price and functionality from the ASM platform. A mitigation strategy was to require all licensed software to be wrapped in an API germane to the domain in which it is used.

Platform Evolution

In contrast to market needs 5 years ago, when minimizing time-to-market was a critical issue, today's needs are to minimize cost and extend lifecycles. In response, this future phase will be characterized by significant refactoring of the ASM platform, keeping in mind the following two gulps that reduce vendor lock and alleviate the short lifecycles of commercial products:

- *Increased hardware independence.* The ASM needs better abstraction of the underlying hardware so that the ASM software can operate on a wider range of hardware. This will alleviate our current vendor lock and unacceptably short hardware lifecycle.
- *Independence from the OS and other commercial software.* The ASM will incrementally replace commercial software with open source implementations. While reliance on third-party products is a useful technique to accelerate time-to-market, when the cost of extended support in the context of a telecommunications lifecycle is factored in, the overall value is significantly diminished. This is particularly true if use of these products is generic and equivalent open source implementations are available.

Conclusions

The ASM platform began with a charter to minimize time-to-market. Over time, the charter evolved to providing Lucent customers with longer product lifecycles. During the course of this evolution, several key strategies proved to be enduring in the ASM environment:

- Select an API that hides the OS and hardware and insulates applications from variability.
- Use proof-of-concepts to guide API evolution and refactoring, and to minimize development risk.
- Engage incremental development to maximize scheduling flexibility.
- Establish a unifying vision to counterbalance the incremental approach.

Along the way, additional strategies were uncovered:

- Carefully consider when to incorporate third-party software.
- Isolate licensed third-party software behind the API.
- Avoid reliance on vendor claims of backward compatibility.

***Trademarks**

CORBA is a registered trademark of Object Management Group, Inc.

Netra and Sun Microsystems are registered trademarks of Sun Microsystems, Inc.

References

[1] International Softswitch Consortium, <http://www.softswitch.org/educational/reference.asp>.

[2] K. W. McKiou and D. Esposito, "OneLink Manager™ EMS for the 7R/E™ Switch," Bell Labs Tech. J., 5:4 (2000), 80–96.

(Manuscript approved June 2003)

AI G. NGUYEN is a member of technical staff in SFUS—System Data Development & Evolution at Lucent Technologies in Naperville, Illinois, where his current responsibilities include Administrative Services Module development and architecture. He holds a B.A. in physics and mathematics from Calvin College in Grand Rapids, Michigan, and a Ph.D. in particle physics from Michigan State University in East Lansing. Dr. Nguyen professional interests are in software platform/middleware engineering. He is the recipient of the 2002 Bell Labs President's Gold Award for the large capacity tandem solution. He has also received a number of awards for his work on the ASM project.

RICHARD L. SOHN is a consulting member of technical staff for the Softswitch System Engineering and Architecture Department at Lucent Technologies in Naperville, Illinois, where his current responsibilities include platform hardware and software architecture for the Lucent Softswitch. He holds a B.S. degree in electrical engineering from Valparaiso University in Indiana, and he is member of IEEE. Mr. Sohn received the 1999 Bell Labs President's Silver Award for work on the ASM project. His areas of interest include computing platform architectures, particularly those that leverage off-the-shelf components, management systems, and music. ♦