Computer

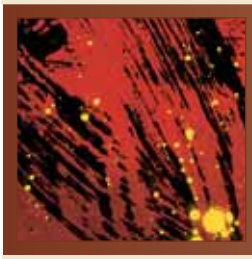
How Business Goals Drive Architectural Design

Ragvinder S. Sangwan and Colin J. Neill

Vol. 40, No. 8
August 2007

How Business Goals Drive Architectural Design

Raghvinder S. Sangwan and Colin J. Neill
The Pennsylvania State University



Designing system architectures requires more than using the technology du jour.

In a keynote address at the Third SEI Software Architecture Technology User Network Workshop this past May, Ian Gorton relayed an interesting story about the Australian Customs Service, which needed to upgrade its two-decade-old legacy system that handles the nation's sea and air cargo.

After careful deliberation, the ACS selected a solution built around XML Web services. The solution could easily wrap legacy software elements behind the façade of services while simultaneously opening the system over the Internet to clients including freight forwarders and customs brokers.

The new Integrated Cargo System (ICS) went live in October 2005. To users' dismay, the system's response time quickly increased from seconds, to minutes, to hours, and finally, to days, before it stopped responding altogether. With Christmas presents piling up at Australia's docks and ports, the ICS fiasco soon made the headlines in major newspapers. But why did it happen?

Technically speaking, the wrapped legacy system provided controlled access to the outside world, which limits the number of transactions it processed. Opening it through Web services increased the load well beyond its capacity.

However, the real lesson isn't that we should consider load, stress, or capacity constraints more carefully, but that we shouldn't regard any cookie-cutter architectures the latest technology offers as appropriate for a particular system.

Stated another way, system architectures are critical and, more importantly, distinct from implementation technologies' architecture. Unfortunately, in practice many discussions around architecting software systems devolve to questions on particular technologies: whether to use event-based or message-based communication, to build the system with .NET or J2EE, to use XML Web services or not.

These debates lead to the technology du jour influencing architecture design decisions instead of criteria that truly impact an organization's mission and bottom line: business goals and quality attribute requirements.

BUSINESS GOALS

Consider a company that primarily sells hardware devices but uses software applications to manage them. The software loses money but helps sell the hardware devices, thereby constituting a loss leader.

The company realizes that the commoditized hardware will shrink profit margins. To sustain business over the long term, the company creates a new profitable software management system. It accomplishes this by reducing internal development costs and expanding the market.

Replacing existing applications with the new software management system helps cut internal development costs. Entering new and emerging geographic markets and opening new sales channels in the form of value-added resellers can expand the market. VARs sell software under their own brand to support hardware devices from many different manufacturers.

This example illustrates how business goals can significantly impact a software management system's architecture without necessarily affecting its functionality. These goals include

- supporting hardware devices from different manufacturers;
- considering language, culture, and regulations of different markets;
- assessing tradeoffs and risks to determine how the product should support these goals; and
- refining goals such as scaling back on intended markets, depending on the company's comfort level with the tradeoffs and risks.

More importantly, these business goals correspond to quality attributes the end system must exhibit. The system must be modifiable to support a multitude of hardware devices and consider different languages and cultures. Supporting different regulations in different geographic markets requires the system to respond to life-threatening events in a timely manner—a performance requirement.

Understanding business goals and their implied quality concerns is there-

Table 1. Business goals and quality attribute scenarios for the software management system.

Business goal	Goal refinement	Quality attribute	Quality attribute scenario	Priority
Open new sales channels in the form of VARs	Support hardware devices from different manufacturers	Modifiability	Two developers integrate a new device into the system in 320 person-hours.	High, High
	Support conversions of nonstandard units the different devices use	Modifiability	A system administrator configures the system to handle units from a newly plugged-in field device in less than three hours.	High, Medium
Expand by entering new and emerging geographic	Support international languages	Modifiability	A developer packages a version of the system with new language support in 80 person-hours.	High, Medium
	Support regulations that require life-critical systems, such as fire alarms, to operate within specific latency constraints	Performance	A life-critical alarm reports to concerned users within three seconds of the event.	High, High

fore critical. We can't simply jump at using a service-oriented architecture, J2EE, or some other tempting solution. Instead, these goals must drive architectural design.

One way to do this is to employ the *quality attribute workshop* (M. Barbacci et al., *Quality Attribute Workshop Participants Handbook*, special report CMU/SEI-2000-SR-001, Software Engineering Institute, Carnegie Mellon Univ., 2000). QAW elicits quality attribute requirements mapped to business goals.

Through workshops, participants use goals that management and technical stakeholders provide to generate scenarios for the quality attributes corresponding to the goals. These scenarios must be specific enough to determine whether a system satisfies a given scenario.

Table 1 shows a mapping of the business goals to quality attribute scenarios for the software management system. The table also shows prioritized scenarios for the software management system.

QUALITY ATTRIBUTE REQUIREMENTS

While QAW provides a way to capture a system's quality requirements in the form of scenarios, documenting these quality attributes doesn't directly provide an architecture that reflects them. To accomplish this, we employ *attribute-driven design* (L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 2003).

ADD first prioritizes the quality attribute scenarios by soliciting input from both business and technical stakeholders. Business stakeholders prioritize scenarios based on their business value—H indicates high; M, medium; and L, low. Technical stakeholders prioritize based on the difficulty associated with achieving a given scenario during system design. The resulting nine combinations are in the following order of precedence: HH, HM, HL, MH, MM, ML, LH, LM, and LL.

We then decompose the system by applying a series of architectural tactics that correspond to each quality attribute. Figure 1 shows the result of applying these tactics to the software management system. The sequence of decomposition reflects the priority order of the quality attribute scenarios in Table 1.

Starting with a monolithic system, shown in Figure 1a, ADD applies modifiability tactics to limit the impact of change and minimize the dependencies on the part of the system responsible for integrating new hardware devices.

As Figure 1b shows, introducing an adapter for each field system demonstrates an *anticipation of changes* tactic, with each adapter exposing a standard interface—a *maintain existing interface* tactic. A virtual field system further limits the ripple effect when removing or adding field systems—a *hiding information* tactic. The adapters shown in Figure 1b also convert nonstandard units used by various devices to standard ones—an *intermediary modifiability* tactic.

As Figure 1c shows, applying a *performance* tactic (concurrency) is next. This step adds support for critical systems operating within specific latency constraints. The components responsible for evaluating rules and generating alarms for life-threatening situations separate into an alarms module. This module can now move to a dedicated execution node reducing latency—introducing multithreading within the module further enhances its performance.

Applying the modifiability (*anticipation of changes*) tactic again in Figure 1d creates a separate presentation module to support multiple international languages.

More often than not, this type of process is not followed and architecture discussions focus on a particular technology without the full understanding of the design problem at hand. Only after the technology is operational do the neglected critical mission goals and unsuitable technology-driven solution become apparent.

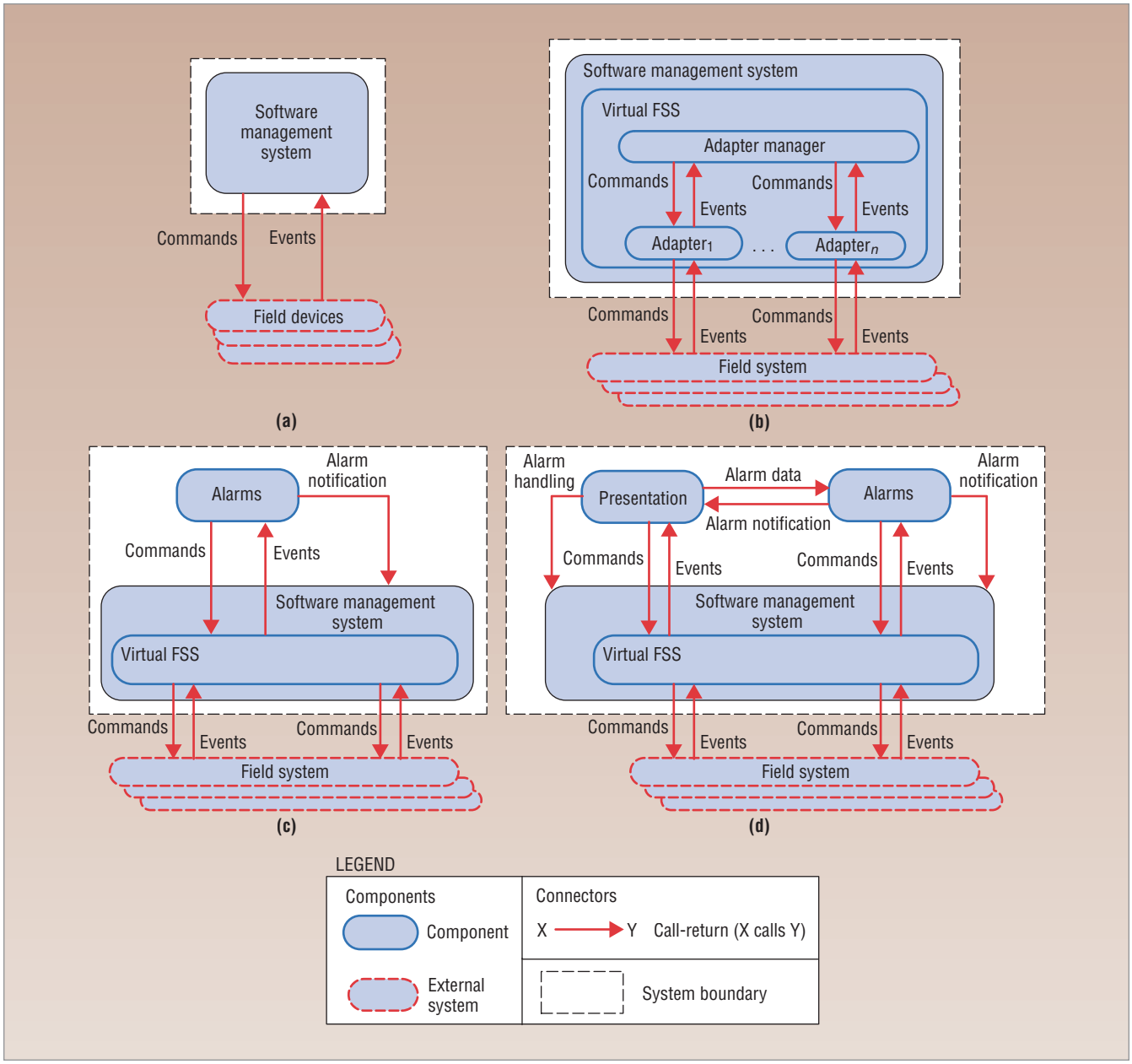


Figure 1. Results of applying architectural tactics to the software management system. (a) Monolithic system; (b) support for adding a new hardware device; (c) support for life-critical systems to operate within specific latency constraints; and (d) support for internationalization.

An organization's business goals and associated quality attribute requirements are the critical forces in determining its system architecture. Failure to clearly understand these forces is likely to result in ill-informed decisions that could fail to meet the sponsoring organization's objectives, disappoint customers, or worse, ruin Christmas!

Raghvinder S. Sangwan is an assistant professor of information science in the Engineering Division of the Great Valley School of Graduate Professional Studies at the Pennsylvania State University. Contact him at rsangwan@psu.edu.

Colin J. Neill is an associate professor of software engineering and assistant head of the Engineering Division of the Great Valley School of Graduate Professional Studies at the Pennsylvania State University. Contact him at cjn6@gv.psu.edu.

Editor: Michael G. Hinchey, Loyola College in Maryland; mhinchey@loyola.edu