An Architecture-Centric Approach for Systems Design

Colin J. Neill and Raghvinder S. Sangwan
Engineering Division
School of Graduate Professional Studies
The Pennsylvania State University
Malvern, PA 19355
cjneill | rsangwan@psu.edu

Daniel J. Paulish
Distinguished Member of Technical Staff
Siemens Corporate Research, Inc.
Princeton, NJ 08540
daniel.paulish@siemens.com

Abstract. The most critical requirements for the lifetime value of a system are its non-functional requirements such as reliability, security, maintainability, changeability, etc. These are collectively known as the ‘ilities’ and are typically addressed in system design once the functional architecture has been developed. In this paper we propose the use of architecture-centric design that modifies this standard workflow so that those non-functional requirements, which actually reflect the true business needs, are addressed first. This ensures that the final system better reflects and embodies those architecturally-significant requirements rather than having them addressed secondarily. This is an important change since the ‘ilities’ are systemic properties (properties of the system as a whole) rather than systematic properties (properties of individual components or sub-systems) and are therefore difficult to address once the functional architecture has been determined and the separation of concerns is already somewhat completed. We provide an example of the approach based around a simplified case study of an online, on-board health monitoring system for shipboard gas turbine electricity generators that collects, filters, analyzes, transmits, and mines sensor data from the generators subsystems.

Introduction

A system is considered successful if it meets stakeholder needs. On the surface this would imply that designing the system to meet the requirements specification developed at project inception would be sufficient, but in reality the stakeholder needs change continuously throughout the development process, and in fact, throughout a system’s lifetime. Correspondingly, the value the system provides to its users will diminish during its lifetime unless action is taken to ensure that the system evolves to meet those changing needs (Browning and Honor 2008; Sangwan et al. 2008a).

In order for the system to evolve the architecture must be designed accordingly. This has typically been achieved by adapting the functional architecture – the architecture developed to meet the functional requirements of the system – through design for reliability, maintainability, usability, etc. (Blanchard and Fabrycky 2006). To this end, significant strides have been made in recent years in the understanding of changeability, flexibility, and adaptability, and how they impact architecture and architectural decisions (Fricke and Shulz 2005; Engel and Browning 2008) to aid in such processes. It is clear, then, that the *ilities*, as they are known, are gaining importance in systems design. In fact, it could even be said that they are now even more important to the lifetime value of a system than the functional requirements – the functional requirements will change, but only if the architecture can endure such change (Ross et al. 2008; Sangwan et al. 2008b).

Given this change in priorities, we are left with the challenge of "how to create systems with the desired behaviors and to predict and suppress the undesired ones" (Crawley et al 2004). In that case, might an alternative approach to architectural design, one that focuses primarily on the non-functional requirements of the system, be advantageous? Instead of first decomposing a system according to its gross functionality, the architectural design is driven by the important quality attributes that ensure its longtime survival (Sangwan and Neill 2007). One approach that has been used successfully in software engineering, and shows promise in systems engineering, is the combination of the Quality Attribute Workshops (Bachmann et al., 2002; Barbacci et al., 2000) and Attribute Driven Design (Bass et al., 2003). In this paper we will demonstrate architecture-centric design in systems engineering using the combination of QAW and ADD through a simplified case study of a generator health monitoring system.

Case Study

GenHealth is an online, real-time health and condition monitoring system for Ship Service Gas Turbine Generators. The SSGTG consists of a gas turbine engine that turns a generator that supplies electrical power to the ship and its subsystems through the ship's switchboards. A typical ship would use three such generators each supplying over 2.5 MW of electrical power.

In such a hostile environment, equipment failure is frequent, with a mean time between failure as low as 600 hours (25 days of round-the-clock operation) and any unscheduled downtime can be very costly, not only financially, but also in terms of safety, and in military applications, security. It is, therefore, important that faults are diagnosed early so that breakdown can be avoided. Such predictive monitoring is both a selling feature of the equipment, but also a cost-savings to the operator and maintainer. Additionally, on-board, on-line monitoring could provide valuable insight into the turbine and generator operation and reveal potential design improvements for future models by mining the data streams from the fuel (rate, pressure), oil (temperature, pressure), power (RMS, harmonics, peak), torque, load, and control sensors. To accomplish the manufacturer is seeking to develop an integrated sensor-data gathering and storage system to deploy onto the ship which can then filter, encrypt, and transmit this data to the ship-board operator's systems for local monitoring and maintenance as well as to the system support central offices for diagnostics and analysis and the regional maintenance and modernization coordination offices for preventive repairs.

In order to meet these objectives the system must incorporate the following broad functional requirements:

- Real-time access to the turbine, control system and generator data
- Statistical modeling of data for early fault prediction.
- Online monitoring of the equipment supported by a visualization of the data
- Ticketing service and event notification related to fault detection.

Figure 1 indicates the context for the GenHealth system. It receives sensor data for diagnostics and analysis from the key subsystems of the SSGTG and interacts with many different users that include the maintenance, support and design engineers.

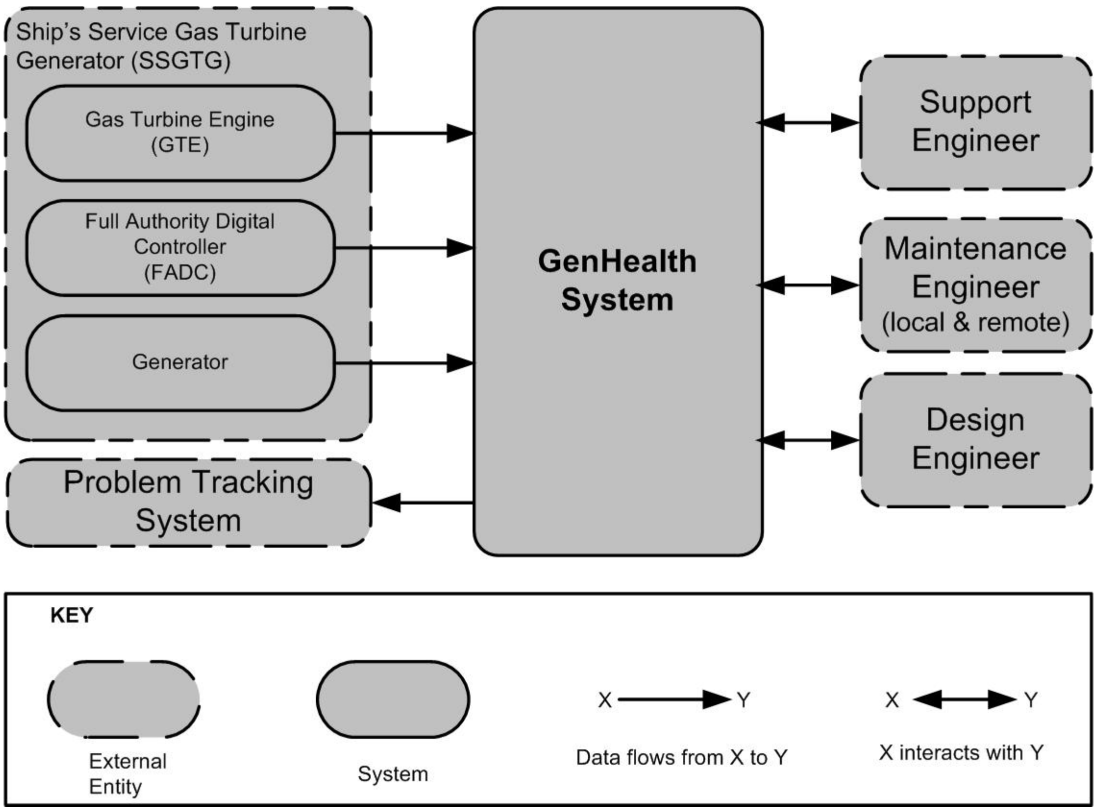

Figure 1: GenHealth context diagram

Of course, this operational context of the SSGTG imposes considerable constraints on the final system that must be accommodated. First and foremost is the reliability and throughput of a data network within the generator and engine rooms of a ship. The heavy equipment and the potentially turbulent, hot, and humid conditions make for a noisy electrical and physical environment in which to transmit large volumes of sensor data from the SSGTG subsystems. Correspondingly, reconciling this sensor data, synchronizing it to a consistent clock, storing it when the network is unreliable, filtering it for upload to the system command's central office, and analyzing it both locally and centrally for diagnosis and evaluation are all made more difficult. In addition, the data must be encrypted to protect the ship's operators and an audit trail is also necessary for non-repudiation.

Architecture-Centric Design

The first step in architecture-centric design is to determine the most critical systemic properties that the architecture must embody. On the surface we may consider all systemic properties important (the system must be secure, reliable, maintainable, robust, modifiable, etc.) but in practice the relative importance of each property varies from system to system. Furthermore, such coarse-grained requirements are not particularly useful in design. We may desire the system be modifiable, but the real question is modifiable with respect to which aspects, when, and with how much effort? (Sangwan et al 2008a)

To answer such questions we employ Quality Attribute Workshops. These are facilitated focus groups of stakeholders which establish a prioritized set of architecturally significant requirements in the form of quality attribute scenarios – "short stories that describe an interaction with the system that exercises a particular quality [with] at the very least, a clear stimulus and response" (Barbacci et al 2000)

The workshop begins with a stakeholder presentation on the business and mission goals for the system under consideration followed by identification of key architectural drivers reflecting those core goals. The GenHealth system has the following business goals:

BG1: *Improve support efficiency*: Provide remote capability to troubleshoot SSGTG problems.

BG2: *Improve availability*: Monitor the health of SSGTG to assess its state in order to predict potential problems and allow the early scheduling of maintenance services, i.e. reduce machine shutdowns and shift maintenance activity from reactive to planned by identifying conditions which contribute to machine loss of life or machine stoppage.

BG3: *Product design optimization*: Analyze data collected as a part of the monitoring effort to improve future design of the SSGTG.

The workshop then proceeds with collaborative brainstorming of quality attribute scenarios that describe operationally the key architectural drivers. The scenarios are subsequently consolidated, refined, and prioritized. The business goals, the corresponding quality attributes, and the quality attribute scenarios from the workshop are summarized in Table 1.

Table 1: Prioritized set of business goals and quality attribute scenarios

Business Goal	**Goal Refinement**	**Quality Attribute**	**Quality Attribute Scenario**	**Priority**
BG 1	Ensure data confidentiality and integrity	Security	*Sensor data shall be accessible and handled only by the authorized engineers, including during on-board storage and transmission.*	*H*
BG 2	Provide early warning to avoid or minimize SSGTG outage	Availability / Security / Performance	*Sensor data from SSGTG begins to indicate some malfunction. GenHealth shall predict a fault and notify the support engineers on-board the ship and maintenance engineers in the central office within 30 minutes of the notification to prevent serious breakdown or stoppage.*	*H*
BG 3	Support statistical modeling capabilities	Performance	*For future design optimization, GenHealth shall provide statistical modeling capabilities on sensor data collected from subsystems within SSGTG*	*M*

Once the quality attribute scenarios are finalized we can proceed with the design of the architecture through attribute driven design (Bass et al 2003). ADD begins by identifying the architectural tactics that match the quality attributes of the system. These tactics codify design knowledge, and in that regard are similar to architectural patterns, but differ in that they are derived from analytic models of quality attributes rather than personal, or anecdotal, experience (Bass et al 2003). The tactics that apply to the quality attributes in the GenHealth system are shown in Table 2.

Table 2: Quality attributes and corresponding architectural tactics

Business Goal	**Quality Attribute**	**Tactics and Tactic Categories**
BG 1 BG2	Security	*Resist attacks:* – Authenticate users – Authorize users – Maintain data confidentiality – Maintain data integrity
BG 2	Availability	*Fault preparation* – Passive redundancy *Fault detection* – Heartbeat *Fault recovery* – State resynchronization
BG 2 BG 3	Performance	*Resource demand:* – Increase computational efficiency – Manage event rate – Control frequency of sampling *Resource management:* – Increase available resources – Maintain multiple copies of data

Architecture Elaboration

Architecture elaboration using ADD is an iterative process that starts with a single monolithic component responsible for all of the system functionality and then recursively decomposes this component by successively applying architectural tactics corresponding to a prioritized set of quality attribute requirements for the system. The final outcome is an architecture that is a tradeoff between conflicting quality requirements and reflects the priority order of the quality attribute requirements. We show the single component for GenHealth system in Figure 2 along with the key for all the component and connector diagrams that will be produced during the architecture elaboration process.

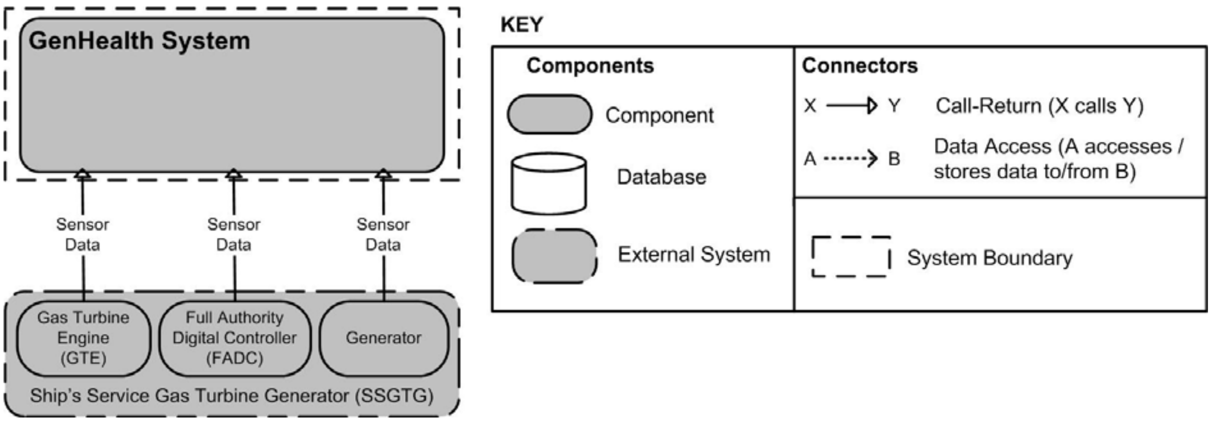
GenHealth System
Sensor Data
Sensor Data
Sensor Data
Gas Turbine Engine (GTE)
Full Authority Digital Controller (FADC)
Generator
Ship's Service Gas Turbine Generator (SSGTG)
KEY
Components
Component
Database
External System
Connectors
X → Y Call-Return (X calls Y)
A → B Data Access (A accesses / stores data to/from B)
System Boundary

Figure 2: The monolithic system

The prioritized set of business goals and quality attribute requirements in Table 1 drive the subsequent elaboration process. BG1 related to security is concerned with data confidentiality and integrity. As shown in Figure 3, these responsibilities are moved to an access control component. Access control uses the *authenticate user* tactic to ensure the user is who he/she purports to be, and the *authorize user* tactic to ensure the authenticated user has the rights to access or modify either data or services. In addition, the data concentrator component uses the *maintain data confidentiality* tactic to protect data from unauthorized access over exposed communication links. This tactic makes use of encryption and secure sockets layer (SSL) to achieve this objective. The data concentrator also uses the *maintain integrity* tactic to ensure data is delivered as intended. This tactic can be implemented using checksums.

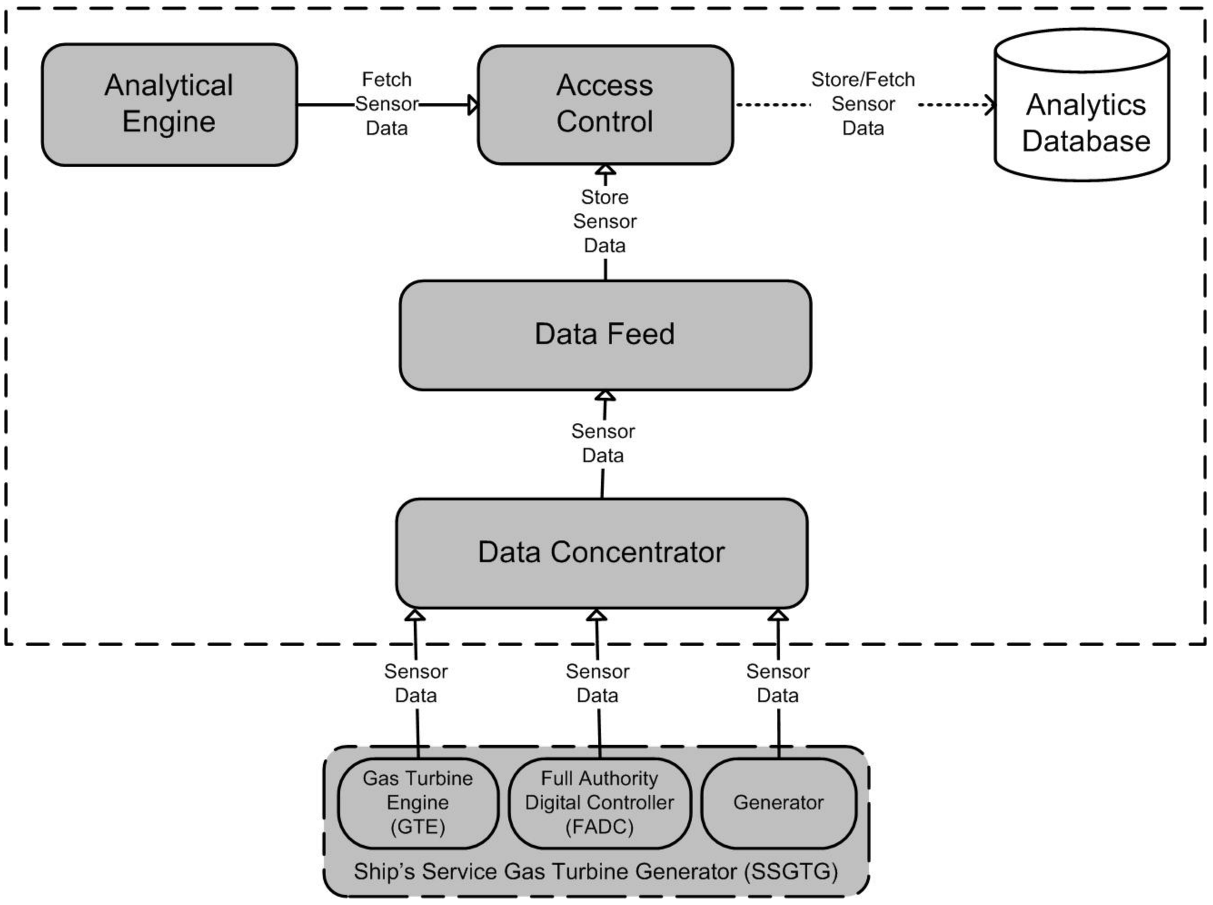

Figure 3: Applying security tactics

The next priorities are the availability, security and performance concerns related to BG2. The primary concern related to availability and performance is that when sensors onboard the SSGTG sense some fault, it is usually within 30 minutes of such an event that most serious faults would lead to machine stoppage or loss. Therefore, it is critical that action be taken within this 30 minute window to avoid significant damage and consequently a prolonged outage. The security concern is that notification be sent to concerned engineers in a manner that does not compromise a military mission, for instance.

The availability architectural tactics that can be used to address this concern belong to *fault preparation, detection and recovery* categories (see Table 2). We show these in Figure 4 for the data logger component only but they can be easily applied to any other component of the system.

Fault preparation tactics are performed routinely during normal operation to ensure that when a fault occurs, recovery can take place. We show this for the data logger component that uses a *passive redundancy* tactic. The primary data logger is responsible for receiving the sensor data from SSGTG. If it fails, a standby secondary can be promoted to the primary and take on the responsibility. The failed instance can be removed and a new instance of a secondary logger can be started.

Fault detection tactics are associated with detecting and dealing with the fault. The data concentrator component periodically receives a *heartbeat* from the data loggers; failure to do so indicates a fault. In case of the primary data logger, the heartbeat would be the periodic delivery of the sensor data itself. Note that this is detection of a fault in the GenHealth system itself, rather than faults in the generator system – the architectural quality refers to the systems availability to notify local and remote maintenance engineers of the status of the generator system; not how the fault prediction algorithms themselves operate.

Fault recovery tactics are concerned with restoring normal operations. When a primary data logger fails, the data concentrator promotes the secondary which then uses a *state resynchronization* tactic to upgrade its state before it can start operating as a primary. The primary data logger always maintains a log of its current state that is used by a secondary for state resynchronization in case of a failure.

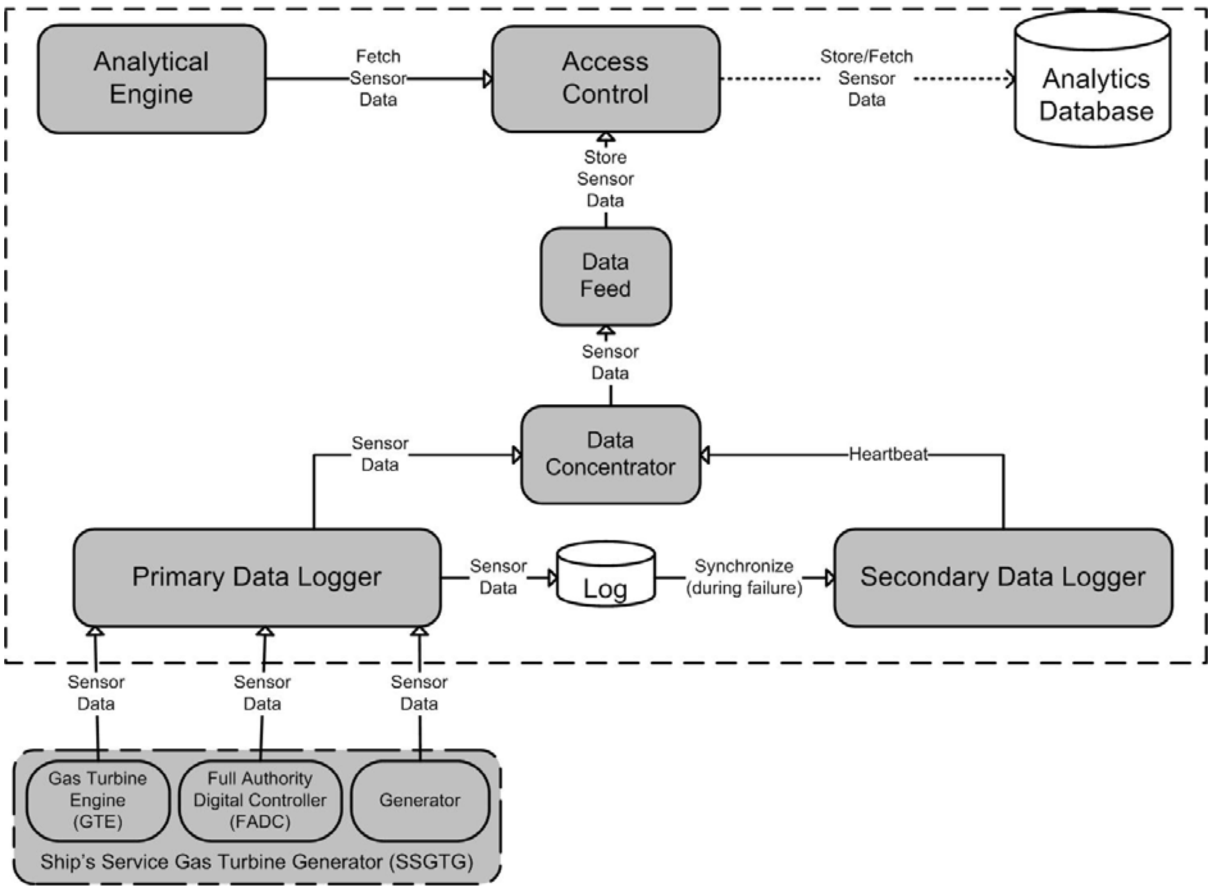

Figure 4: Applying availability tactics

By breaking the system into these distinct components, the individual components can also address their specific latency (or performance) concerns using appropriate tactics (see Table 2). For instance, the data logger component can reduce latency through the use of *manage event rate* and *control sampling frequency* tactics that manage the number of sensor data samples and the sampling frequency. The data concentrator and analytics engine can use the *increase computational efficiency* tactic to manage the performance of the data encryption/compression and predictive analysis algorithms respectively. All of these components can use *increase available resources* tactic as they can all be independently deployed.

Finally for BG3 related to performance, the primary concern is to provide statistical modeling capability for future design optimization. This is a computationally intensive process and would impose a significant load on the processor and memory resources. The architectural tactics that can be used to address this concern belong to the *resource management* category (see Table 2).

To manage this load we further decompose the system into a data mining engine with its corresponding data warehouse that can be used for offline analysis of historical data for future SSGTG design improvements. The data mining engine can now be allocated its own resources, an *increase available resources* tactic, and the data warehouse is populated by replicating the sensor data from the analytics database, a *maintain multiple copies of data* tactic. This is shown in Figure 5.

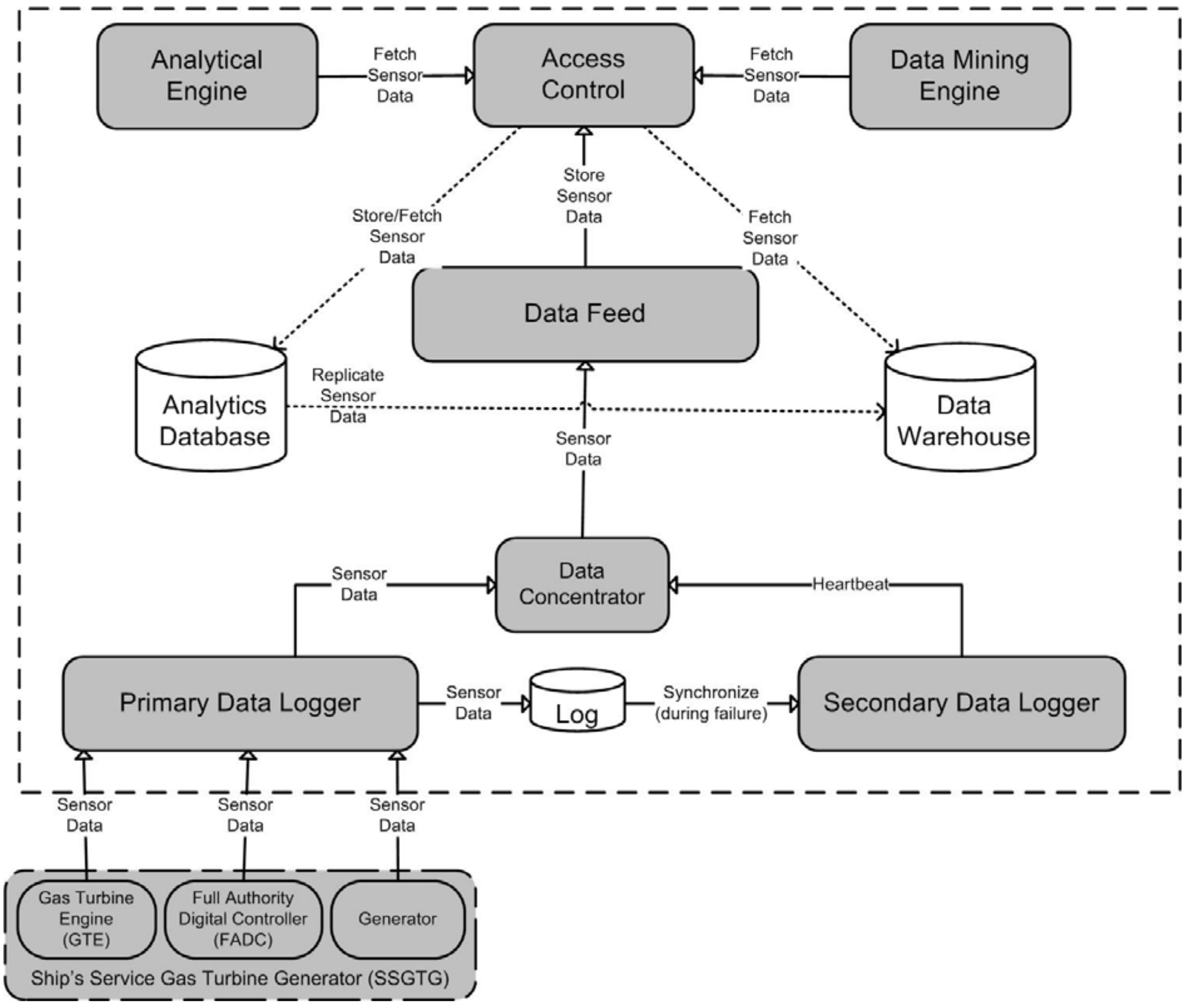

Figure 5: Applying performance tactics

Related Work

It is clear from the recent literature that the quality of the architecture of systems is becoming of increasing interest and the need to understand the design choices that affect that quality is critical to the success of the system and its value over its lifetime. To help architects visualize these choices, and their consequences, several investigators have explored the formal definitions of some of the 'ilities.' Nilchani and Hastings (2007) examined system flexibility and state that it is such an important attribute that it requires separate analysis in terms of its value to the system. This work was then extended in an examination of changeability that incorporated flexibility along with adaptability, scalability, modifiability, and robustness in multi-attribute trade space exploration in an attempt to redress the "inability in the community to design and assess systems on the basis of these 'ilities,' especially in terms of concrete and unambiguous definitions."(Ross et al. 2008). Fricke and Schulz (2004) had earlier defined changeability in these terms while expressing the importance of flexibility, agility, and adaptability to the success of a long-lived system. They proposed a set of principles for design for changeability including simplicity, modularity, independence, and autonomy that is entirely consistent with the architecture-centric approach here described. Engel and Browning (2007) addressed the same issues of design for adaptability but with a novel interpretation of financial options theory giving rise to their design for dynamic value using architecture options. For the most part, however, these studies tackle the thorny problem of weighing the choices of different design decisions with regards to the 'ilities,' rather than prescribing how to create a design that reflects those properties.

Other authors have explored the practices and principles that help guide architects in developing high-quality architectures, however. Maier (2006), while addressing the need to reconcile software and system architectures, observed that systems engineering typically took a function-first approach and suggested that "functional and behavioral analysis techniques should emphasize the identification and documentation of the critical scenarios early in the process, and defer detailed functional specification." To achieve this he suggests the use of the 4+1 model (Kruchten 1995) of architectural views so that the software and system architectures are consistent. Indeed, this cross-over of techniques from software to systems engineering has provided a rich ground for investigators. Neill and Holt (2002) proposed a combination of the UML and a formal model of temporal behavior for systems design, and in a series of articles, the use of Structured Analysis and Object-Oriented Analysis and Design were described for Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) architectures (Levis and Wagenhals 2000; Wagenhals et al. 2000; Bienvenu et al. 2000). With the emergence of SysML, such consistency between the software and systems domains has become considerably easier to achieve and has allowed for still more software engineering design approaches to cross-over into systems engineering. Rao et al (2008) propose combining SysML with colored petri-nets for system-of-system design using standard object-oriented analysis and design techniques, for example. With all these approaches, however, it is increasingly accepted that they do not handle systemic properties, such as quality attributes, well (Garlan 2000; Kazman et al. 2004; Sangwan et al. 2008a), which is precisely the objective of the approach presented in this work.

Conclusions

In this paper we have presented an architecture-centric approach to systems design based on quality attribute workshops and attribute-driven design that has been used successfully in software-centric systems and shows great promise in the systems domain. Unlike most system design approaches, the architecture-centric approach is driven from systemic quality attributes that are determined from both the business and technical goals of the system rather than just its functional requirements. This ensures that the architecture of the final system clearly, and traceably, reflects the most important goals for the system. While superficially the most important features of any system are its functions, in practicality it is the non-functional requirements that have the greatest impact on a systems lifetime value since it is these requirements that determine how easily the system accepts future change, and how well the system meets the reliability and security needs of its operators and owners. By making these requirements "first class citizens" the architecture meets these needs first. Furthermore, most non-functional requirements are systemic properties: they are properties that the entire system must reflect rather than just one component or subsystem. They, therefore, cannot easily be built into an existing architecture as is common in secondary design for reliability or maintainability efforts. In essence, these properties must be designed in from the beginning. Architecture-centric design addresses this directly by utilizing analytically-derived tactics to inform the architects design choices and shape the architecture.

We demonstrated the technique in the design of a health-monitoring system for ship-board gas turbine electrical generators where the architecturally-significant requirements were reliability, performance, and security and produced a first-pass architecture that meets those requirements directly.

References

Bachmann, F., Bass, L., Klein, M. 2002. *Illuminating the fundamental contributors to software architecture quality* (CMU/SEI-2002-TR-025), Pittsburgh, PA: Software Engineering Institute Carnegie Mellon University, August 2002.

Barbacci, M., Ellison, R., Weinstock, C., Wood, W. 2000. *Quality attribute workshop participants handbook* (CMU/SEI-2000-SR-001), Pittsburgh, PA: Software Engineering Institute Carnegie Mellon University, July 2000.

Bass, L., Clements, P., and Kazman, R. 2003. *Software architecture in practice*, Second Edition, Boston, MA: Addison-Wesley, 2003.

Bienvenu, M.P., Shin, I. and Levis, A.J. 2000. C4ISR Architectures: III. An object-oriented approach for architecture design. *Systems Engineering*, Vol. 3, No. 4, 2000 pp 288-312.

Blanchard, B.S. and Fabrycky, W.J. 2006. *Systems engineering and analysis*, 4th Ed. Pearson Prentice Hall, 2006.

Browning, T.R. and Honour, E.C. 2008. Measuring the Life-cycle Value of Enduring Systems. *Systems Engineering*, Vol. 11, No. 3, 2008, pp 187-202.

Crawley, E., deWeck, O., Eppinger, S., Magee, C., Moses, J., Seering, W., Schindall, J., Wallace, D. and Whitney, D. 2004. The influence of architecture on engineering systems, MIT Eng Syst Symp, March 29–31, Cambridge, MA, 2004.

Engel, A. and Browning, T.R. 2008. Designing systems for adaptability by means of architecture options. *Systems Engineering*, Vol. 11, No. 2, May 2008, pp 125-146.

Fricke, E. and Schulz A.P. 2005. Design for Changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle. *Systems Engineering*, Vol. 8, No. 4, 2005 pp 342-359.

Garlan, D., 2000. Software architecture and object-oriented systems, in Proceedings of the IPSJ Symposium 2000, Tokyo, Japan, August 2000.

Kazman, R., Kruchten, P., Nord, R., and Tomayko, J. 2004. Integrating software-architecture-centric methods into the Rational Unified Process (CMU/SEI-2004-TR-011), Pittsburgh, PA: Software Engineering Institute Carnegie Mellon University, 2004.

Kruchten, P.B. 1995. A 4+1 view model of software architecture, *IEEE Software*, 12(6), November 1995, pp. 42–50.

Levis, A.H and Wagenhals, L.W. 2000. C4ISR Architectures: I. Developing a process for C4ISR architecture design. *Systems Engineering*, Vol. 3, No. 4, 2000 pp 225-247.

Maier, M.W. 2006. System and software architecture reconciliation. *Systems Engineering*, Vol. 9, No. 2, 2006. pp 146-149.

Neill, C.J. and Holt, J.D. 2002. Adding temporal modeling to the UML to support systems design. *Systems Engineering*, Vol. 5, No. 3, 2002, pp. 213-222

Nilchiani, R. and Hastings, D.E. 2007. Measuring the value of flexibility in space systems: A six-element framework. *Systems Engineering*, Vol. 10, No. 1, 2007 pp 26-44.

Rao, M., Ramakrishnan, S. and Dagli, C. 2008. Modeling and simulation of net centric system of systems using systems modeling language and colored petri-nets: A demonstration using the Global Earth Observation System of Systems. *Systems Engineering*, Vol. 11, No. 3, 2008 pp 203-220.

Ross, A.R, Rhodes, D.H., and Hastings, D.E. 2008. Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. *Systems Engineering*, Vol. 11, No. 3, 2008 pp 246-262.

Sangwan, R.S. and Neill, C.J. 2007. How business goals drive architectural design. *Computer*. Vol. 40, No. 8, August 2007. pp 101-103.

Sangwan, R.S., Neill, C.J., Bass, M, and El Houda, Z. 2008a. Integrating software architecture-centric methods into object-oriented analysis and design, *Journal or Systems and Software*. Vol. 81, Iss. 5, May 2008, Pages 727-746.

Sangwan, R.S., Li-Ping Lin, and Neill, C.J. 2008b. Structural complexity in architecture-centric software evolution, *Computer*, Vol. 41, No. 10, October 2008, pp. 99-102.

Wagenhals, L.W., Shin, I., Kim, D., and Levis, A.H. 2000. C4ISR Architectures: II. A structured analysis approach for architecture design. *Systems Engineering*, Vol. 3, No. 4, 2000 pp 248-287.

Author Biographies

Dr. Colin J. Neill is associate professor of software engineering at the Penn State School of Graduate Professional Studies. He teaches in the graduate systems engineering, software engineering, and engineering management programs. He holds degrees in Electrical and Electronic Engineering, Communication Systems, and Software and Systems Engineering from the University of Wales Swansea. He is the author of over 60 articles on software and system design, architectural quality and complexity, and requirements engineering.

Dr. Raghvinder S. Sangwan is assistant professor of software engineering at the Penn State School of Graduate Professional Studies. He holds a Ph.D. in computer and information sciences from Temple University. His research and teaching involves analysis, design, and development of system architectures, and automatic and semi-automatic approaches to assessment of their design. Prior to joining Penn State Great Valley, he worked as a lead architect for Siemens on geographically distributed development projects, building information systems for integrated health networks. He is a technical consultant for Siemens Corporate Research in Princeton.

Dr. Daniel J. Paulish is a Distinguished Member of Technical Staff at Siemens Corporate Research in Princeton, New Jersey with over twenty years experience in software engineering management and is currently the leader of the Siemens Software Initiative in the Americas. He is the author of *Architecture-Centric Software Project Management* and a co-author of *Global Software Development Handbook*, and *Software Metrics: A Practitioner's Guide to Improved Product Development*. Dr. Paulish is formerly an industrial resident affiliate at the Software Engineering Institute of Carnegie Mellon University, and he holds a Ph.D. in Electrical Engineering from the Polytechnic Institute of New York.