On the Role of Quality Attributes in Specifying Software/System Architecture for Intelligent Systems

Hessam S. Sarjoughian
Computer Science & Engineering
Arizona State University
Tempe, AZ 85287-5406
sarjoughian@asu.edu
www.acims.arizona.edu

Abstract

In recent years, researchers in software engineering have been exploring a range of issues on software and system architectures. One area that directly affects intelligent system is software and system architecture. Recent findings offer evidence that software architecture can be developed based on a set of well-defined concepts and methods. In this paper, we examine this line of research and its role in the field of intelligent systems. We suggest using *architecture lifecycle* as an approach to deriving architectures for intelligent systems. We introduce *intelligence* quality attribute and propose using it with a set of existing quality attributes to support incremental, evolutionary specification and evaluation of *intelligent software/system architectures*.

1 Introduction

The architecture of a computing system can be defined to be one or more structures composed of software/hardware components and their connections. A variety of architectural structures – conceptual, physical, control flow, data flow, module, call, process or coordination, and class – serve a collection of complementary, and sometime necessary, blueprints of complex systems. For example, a data flow structure allows one to trace satisfiability of functional requirement based on its constituting components and connections which represent programs and send data relationships respectively. These structures are of particular importance since they can serve as the first set of formal specifications suitable for "predicting" the desired behavior of intelligent systems. That is, a subset of such structures capture a system's behavior in terms of primitive artifacts that can be specified and qualified/quantified over a system's lifetime.

2 Software/System Architectures

The role of an architecture is to put in place a set of blueprints guiding the development of a system that can satisfy some desired behavior. The system's desired behavior can be formally specified in terms of a set of *quality attributes* such as performance and reusability. Many system architecture definitions, descriptions, and specifications have been proposed and employed from fields such as control theory, artificial intelligence, and software engineering (e.g., [1-9]). Bass, Clemens, and Kazman [4] offer a comprehensive account for the software/system architecture. Their treatment is founded on fundamental concepts (e.g., hierarchical structure) and techniques (e.g., modularization) proposed by Fred Brooks, Edsger Dijkstra, Kevin Iverson, and David Parnas among others. They present the architecture of a computing system as a set of specifications capturing its key elements in a systematic fashion. The approach allows one to specify and assess an architecture in a step-wise fashion – i.e., moving from domain independent to application-specific. The architecture, therefore, can be said to provide a set of artifacts that can be evaluated in terms of how well it supports or enables a system's desired external (e.g., availability and performance) and internal behavior (e.g., testability and modifiability).

More formally, an architectural specification of a computing system is defined as a structure or structures composed of software components, the externally visible properties of the components, and the relationships among them [4]. Each structure can be viewed as an skeleton of a system in terms of its components and connections. Each component has a type (software, hardware, or mixed), behavior (static, dynamic, or mixed), and roles (processing, control, mediating, etc.). The connections specify the type (communication, control, send/receive) and nature (e.g., asynchronous) of connections among the components. The structure itself specifies the topology of components and the kinds of interactions enabled – e.g., hierarchical, flat, layered, and mixed such as layered hierarchical.

The software/system architecture lifecycle [1] is shown in Figure 1a. These architectures can be derived in terms of three more primitive artifacts: architectural style, reference model, and reference architecture (see Figure 1). These artifacts provide "early design decisions". An architectural style specifies which quality attributes it supports based on generic description of component types and their patterns of runtime control and/or data transfer. The architectural style also specifies general constraints placed among components and types of interactions among them. For example, layered architecture defines how a layer may interact with layers below or above it and supports reusability quality attribute.

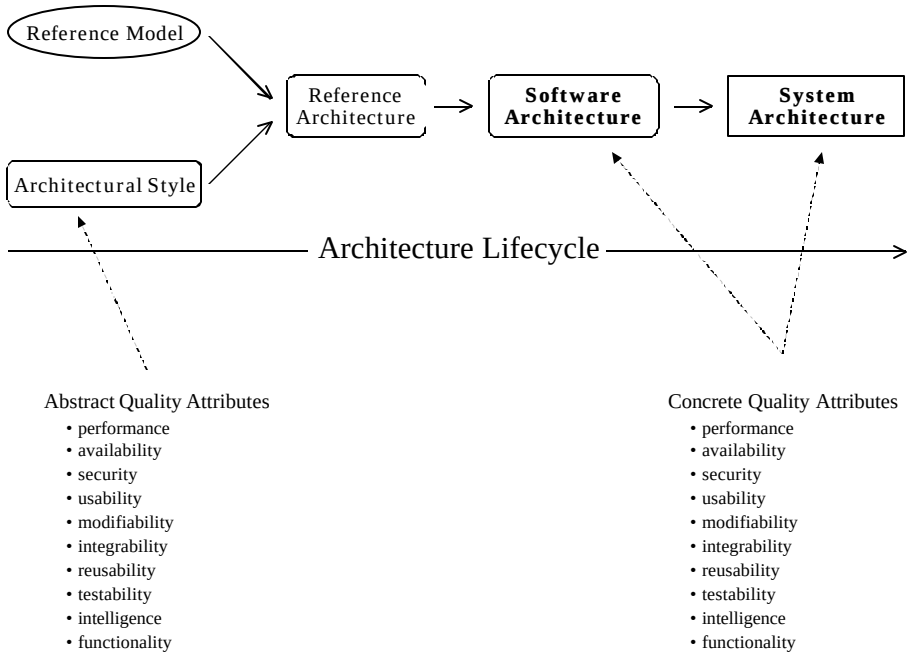

Figure 1: Architecture Lifecycle and Quality Attributes

The reference model, unlike architectural style, specifies a standard decomposition of a known problem (domain) into parts that cooperatively provide a suitable solution (e.g., Model-Viewer-Controller). A reference model represents a division of functionality together with data and control flow between them. The reference architecture is a reference model mapped onto components [2] along with their data and control flow where components cooperatively implement the functionality defined in the reference model. Next, we describe in some detail architectural style and quality attributes.

2.1 Architectural Styles

Architectural styles play a major role in the development lifecycle of intelligent systems since they can serve as the first step toward specifying and examining quality attributes (see Section 2.2). Unlike a software or system architecture, an architectural style specifies a family of architectures [3] – i.e., a set of components, and connectors along with a topology and its semantic constraints. In an architectural style, the topology specifies the layout of components and connectors as well as their collective interactions. Each component is designated to carry out some functions (e.g., a semi-autonomous vehicle processing sensory inputs). The connectors (e.g., procedure call, multicast, etc) are responsible for mediating communication, coordination, or cooperation among components. The architectural style,

[1] Lifecycle is iterative and evolutionary.
[2] Mapping of reference model to reference architecture is not necessarily one-to-one.
[3] It is important to note that architectural style not a kind of (software or system) architecture. Architectural style (a) does not specify the number of components, (b) dos not provide domain-specific, detailed functionality of the components and (c) does not give precise details for the interactions that may take place among components.

therefore, is an abstraction of a set of software/system architectures capturing their abilities and limitations from a domain-independent viewpoint. The importance of an architectural style is that it can be examined in terms of how well it enables some quality attributes (e.g., performance and integrability) and possibly impeding others (e.g., modifiability and security) in early stages of design.

2.1.1 Quality Attributes

Establishing quality attributes is an essential step toward specifying a system architecture which can facilitate or hinder some desired/necessary behavior. Quality attributes can serve as elementary artifacts for specifying and measuring a software/system architecture from distinct and complementary viewpoints. The quality attributes are categorized as runtime observable and non-runtime observable [4]. Common runtime observable attributes are performance, availability, security, and usability. Modifiability, integrability, reusability, portability, and testability quality attributes are common non-runtime observable. To account for system's behavior, there exist also the functionality quality attribute. The collection of runtime and non-runtime quality attributes serve as a basis toward specifying alternative architectures where each architecture is an instantiation of quality attributes tradeoffs. The tradeoffs capture interaction among quality attributes and their ranking with an underlying assumption that there can be no architectural specification that can satisfy all quality attributes[4].

To exemplify the interplay of architectural style and quality attributes, we consider a university library and its on-line system where patrons have access to hardcopy materials such as books and archival periodicals as well as softcopy materials and other on-line information via a web portal. Suppose we are asked to recommend an architectural style[5] satisfying availability and portability quality attributes. One recommendation may suggest the base architectural style to be a layered and transactional database-repository with call-and-return and client-server architecture flavor (see Table 2). The layered style provides portability for the client-side and if necessary for the server-side. To support high degree of availability (e.g., > 99.9%), we can use multiple servers either in centralized or decentralized fashion. The centralized approach suggests having two or more identical servers in one physical location where the system can automatically switch from one server to another as necessary (replace data storage unit.). In a decentralized setting, servers are dispersed in different locations thus ensuring availability – for example network (e.g., router/switch) or power failure. Client/server fits the general pattern of use of services offered by university library on-line system and dispersed patrons. It supports the main scheme of many users accessing the on-line system. Data-centered style is useful and important for satisfying the functionality requirement. The server-side data (e.g., library holdings) needs to be stored in a centralized database for frequent access by large number of patrons.

Architectural Styles	Control Issues			Data Issues		
	Topology	Synchronicity	Binding Time	Topology	Synchronicity	Mode
Call-based client/server	star	synchronous	wt, ct, rt	star	spor., lvol.	passed
Layered	hierarchical	any	wt, ct, it, rt	hierarchical	spor., lvol., cont.	any
Decentralized servers	arbitrary	asynchronous	wt, ct, rt	arbitrary	spor., lvol.	passed

Table 1: Candidate Architectural Style for University Library On-line Example

It is important to note the data/control issues as they show how static and dynamic aspects of each architectural style relates to satisfying some desired quality attributes. The control topology is principally of star pattern between library patrons and the library on-line system. Given the layered architectural style, control topology is hierarchical for a single server and arbitrary if multiple servers are used (set of decentralized servers). Control is asynchronous since there is no need to synchronize users – users interactions with the system are independent of one another. The overall data topology is arbitrary and hierarchical with multiple servers. Data continuity is also frequently sporadic and low volume (small amount of data is transmitted from client to server). Binding time is of all types – wt(write-time),

[4] There are three complementary higher level categories of quality attributes referred to as system, architecture, and business [4].
[5] It is important to note that the above recommendation is subject to business quality attributes – use of cost, market, tools, etc. may result in other architectural styles!

ct(compile-time), it(invocation-time) and rt(run-time). Data to and from clients are primarily "passed" although the database content is "shared". Table 1 shows typical settings for data and control for the architectural styles [4].

3 Intelligent Systems Lifecycle Architecture

Given the foregoing discussions, we can consider the role of architectural style as the first step toward specifying the architecture of an intelligent system. Before we proceed further, we briefly discuss performance and intelligence attributes which are considered as necessary for any intelligent system[6].

3.1 Performance Attribute

Performance attribute represents how well (response time) a system responds to stimulus both internally and externally (e.g., a robot decision to abandon a subgoal and pursue another; heed a warning send by a remote sensor). An architecture specification can satisfy performance quality by controlling the communications among components. For example, the number of interactions and the size and number of data/control can be minimized to increase performance. Of course, performance also depends on how fast each component can carry out its responsibilities, interdependencies of a component's processing on other components, hardware components (e.g., processing speed of a router and network bandwidth) and architectural topology (e.g., physical separation between computing nodes), etc. The key point is that the architectural style is the first design artifact which can be used to specify and assess performance attribute in a formal setting. Furthermore, since architectural style design decisions feed into the software and system architecture, we are able to track it from its abstract to its concrete specifications. This ability has a profound consequence – i.e., formally trace the satisfiability of performance attribute from its inception to its realization within the architecture as shown in Figure 1.

3.2 Intelligence Attribute

Now, we introduce *intelligence* as one additional quality attribute to those given in [4]. Treating intelligence as a quality attribute is based on the concept that it is distinct from the *functionality* quality attribute in the sense that a system may produce some desired behavior without necessarily using its intelligence faculty[7]. Consider the definition of intelligence as proposed by [7] – "intelligence is the ability of a system to act appropriately in an uncertain environment, where an appropriate action is that which increases the probability of success, and success in the achievement of behavioral subgoals that support the system's ultimate goal". This definition suggests, among other things, the notion that an intelligent system may achieve its goal but not necessarily with the same success rate as compared with one that is considered intelligent. Thus, we suggest the intelligence quality attribute to be distinct from the other quality attributes. Clearly, intelligence considered as a quality attribute implies that it plays a key role in the architecture lifecycle – i.e., some architectural styles may provide for intelligence while others may not and therefore some software/system architecture may or may not be categorized as "intelligent" by examining its intelligence quality attribute. Similar to other quality attributes, intelligence quality attribute interacts with other quality attributes (e.g., intelligence and security quality attributes may interact with another).

3.3 Use of Quality Attributes

As we have discussed, software/system architectures increasingly play a central role in the development of large-scale complex intelligent systems. This is primarily due to the fact that architectural specifications collectively contain the necessary knowledge required for describing a system's performance, availability, testability, functionality, etc. Intuitively, performance and intelligence of an intelligent system should therefore be traceable to its architecture. For example, for an semi-autonomous vehicle to be able to restructure itself in pursuit of achieving its objectives, it may need to account for quality attributes such as intelligence. The architecture of a semi-autonomous vehicle navigating itself to its destination through an unknown environment must account for the intelligence and performance quality attributes. Concrete realization of these quality attributes may be arriving at the destination within 5% of a specified time period with high-degree of intelligent decision-making to avoid collision – e.g., 98% collision avoidance rate.

[6] Reference to performance and intelligence as "attributes" may not necessarily be in agreement with the characterizations given in [6] and [7].
[7] Similar arguments can be given for other quality attributes.

This view suggests *abstract* and *concrete* quality attributes where architectural style accounts for the former and software/system architecture accounts for the latter. Given abstract and concrete quality attributes, we may ask the following questions in order to substantiate the premise that intelligent systems can be specified with the architecture lifecycle using quality attributes (see Figure 1).

1. do abstract quality attributes form a primitive basis for specifying the architecture of intelligent systems?
2. can abstract quality attributes be evolved to their concrete counterparts through the architecture specification lifecycle?

We answer our questions informally. An affirmative response to the first question implies that architectural style is the most suitable specification to capture quality attributes. Architectural style by definition is void of domain-specific knowledge. Furthermore, architectural style is sufficiently rich to account for quality attributes – its structure, components, and interactions can account for quality attributes in an abstract setting – independent of explicit knowledge for a system under consideration. Consider, a layered architectural style. This style supports dividing tasks into those aimed for a specific application and those generic to many applications. This accounts for the hierarchical decision-making. The layering approach also allows lower levels to be specialized for distinct computing platforms and thus supports the portability quality attribute. Intelligence and portability attributes are possible without domain specific knowledge for a specific application. The satisfiability of these attributes can be shown by examining the constituent parts, control issues, and data issues of the architectural style[8] (refer to Table 2). For example, hierarchical data and control topologies play major roles for satisfying the modifiability and portability attributes – i.e., the control and data issues are of direct importance to allow components interact via their connectors in an abstract setting.

Constituents Parts		Control Issues			Data Issues		
Components	Connectors	Topology	Synchronicity	Binding time	Topology	Synchronicity	Mode
various	various	hierarchical	any	wt, ct, it, rt	hierarchical	spor., lvol., cont.	any

Table 2: Layered Architectural Style Specification

With an affirmative response to the first question, we now turn to the second question. The answer to this question is also affirmative but perhaps less evident. In the architectural lifecycle, reference model accounts for known (empirically proven) ways of partitioning a class of systems (e.g., Model-View-Controller) into components and interactions using a set of unit operations [4] such as abstraction, compression, and part-whole and is-a decompositions. These unit operations, of course, underlie the quality attributes – for example, the degree to which performance and availability quality attributes can be supported by the reference model. Furthermore, unit operations support reasoning about architectural styles and thus allowing measurements in qualitative/quantitative forms. This suggests that reference model serves as a vehicle to transform abstract quality attributes to those that are grounded in a particular class of systems. Therefore, the combination of the reference model and architectural style (i.e., reference architecture) serve as the intermediary for deriving the concrete quality attributes for a software/system architecture.

Quality attributes as defined within the architecture lifecycle can be used as a means to formalize varying levels of intelligence from the architectural specification point of view. For example, intelligence can be said to range from selection rules (most primitive) to synthesis of paradigm (most advanced) [7]. The intermediary levels of intelligence are combinations of selection rules, new rules, grouping of rules, synthesis of states, and synthesis of context. That is, the ability to begin with abstract quality attributes and arrive at their concrete counterparts can serve a means to formalize increasingly higher levels of intelligence.

4 Other Approaches for Developing Intelligent System Architectures

A large body of research has been devoted to the study of intelligence and intelligent systems. Many algorithms, techniques, and methodologies have been developed for intelligent systems (e.g., [7, 10-13]). Some of these have

[8] Control/data interaction for the layered architectural style is specified as having often isomorphic shapes with flow directions either being the same or opposite [4].

become the foundations for system architectures such as NIST-RCS [2], subsumption [3, 5], InteRRAP [14]. Some architectures (e.g., subsumption) emphasize lower-level information processing and evaluation (e.g., obstacle detection and motion control) while others (e.g., InteRRAP) focus on higher-level knowledge processing and decision-making (e.g., planning and model-based prediction). The latest NIST-RCS reference architecture described by Meystel and Albus [7] account for both low- and high-level processing and decision-making. They suggest sensors, actuators, sensory processing, behavior generation, world model, and value judgment to account for the RCS class of reference architectures.

5 Conclusions

We have presented an architecture lifecycle for the design of intelligent systems. We described the importance of architectural styles as the basis toward specification and development of software and architectures. We showed the key role quality attributes play and their importance in specifying and measuring an intelligent system's performance, intelligence, modifiability, etc.. An important aspect of the quality attributes is how they evolve from their abstract form (captured in an architectural style) to their concrete realizations (represented in a system architecture). Our proposition rests on the premise that an incremental and evolutionary architectural specification lifecycle can lend itself best for selecting, specifying, developing, and measuring quality measures and ultimately system architecture. Furthermore, with this approach, tradeoffs among quality attributes can be systematically accounted for and analyzed.

References

1. Albus, J.S., M. H.G., and L. R., *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*. 1987, Robot Systems Division, Center for Manufacturing Engineering, National Technical Information Service: Gaithersburg, MD.
2. Albus, J.S., *A Reference Model Architecture for Intelligent Systems Design*. 1994, NIST: Gaithersburg, MD.
3. Brooks, R.A., *Cambrian Intelligence: the early history of the New AI*. 1999: MIT Press.
4. Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*. The SEI Series in Software Engineering. 1998: Addison Wesley.
5. Brooks, R.A., *A Robust Layered Control System for A Mobile Agent.* IEEE Journal of Robotics and Automation, 1986. **2**(1): p. 14-23.
6. Meystel, A.M. and E.R. Messina. *Measuring the Performance and Intelligence of Systems*. *Proceedings of PerMIS*. 2000. Gaithersburg, MD: NIST.
7. Meystel, A.M. and J.S. Albus, *Intelligent Systems Architecture, Design, and Control*. 2001: John Wiley & Sons, Inc.
8. Müller, J.P. and M. Pischel, *An Architecture for Dynamically Interacting Agents.* International Journal of Intelligent and Cooperative Information Systems, 1994. **3**(1): p. 25-45.
9. Sarjoughian, H.S., B.P. Zeigler, and S.B. Hall, *A Layered Architecture for Agent-based System Development.* IEEE Proceeding, 2001. **89**(2): p. 201-213.
10. Genesereth, M., N.J., Nilsson, *Logical Foundation of Artificial Intelligence*. 1987, San Mateo, CA: Morgan Kufmann. xviii+405.
11. Saridis, G.N., *Intelligent Robotic Control.* IEEE Trans. Auto. Cont., 1983. **28**(5): p. 547-557.
12. Zadeh, L.A., *Fuzzy Logic, Neural Networks and Soft Computing.* Comm. of ACM, 1994. **37**(3): p. 77-84.
13. Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. 1990, New York: Academic Press.
14. Müller, J.P., *The Design of Intelligent Agents: A Layered Approach*. Lecture Notes in Artificial Intelligence. 1996: Springer.