Design and Evaluation Criteria for Layered Architectures

A. J. Gerber[1], A. Barnard[2] and A. J. van der Merwe[2]

[1] Meraka Institute
Pretoria
Gauteng
South Africa

[2] School of Computing, University of South Africa,
Muckleneuk, Pretoria
Gauteng
South Africa

Abstract. The architecture of a system is an indispensable mechanism required to map business processes to information systems. The terms *architecture, layered architecture* and *system architecture* are often used by researchers, as well as system architects and business process analysts inconsistently. Furthermore, the concept *architecture* is commonplace in discussions of software engineering topics such as business process management and system engineering, but agreed-upon design and evaluation criteria are lacking in literature. Such criteria are on the one hand valuable for the determination of system architectures during the design phase, and on the other hand, provides a valuable tool for the evaluation of already existing architectures. The goal of this paper is thus to extract from literature and best-practices such a list of criteria. We applied these findings to two prominent examples of layered architectures, notably the ISO/OSI network model and the Semantic Web language architecture.

1 Introduction

Currently, the architecture of a system is an indispensable mechanism used to map business processes to the required information system [1]. The term 'architecture' seems to defy the creation of a common, agreed definition within the information system application domain. Although the concept *architecture in software systems* was not formally defined with the introduction of structured programming, it was implied in the work of pioneers such as Parnas and Dijkstra [2–4]. These pioneers derived techniques to model a system as consisting of components. Dijkstra was mainly concerned with program clarity and correctness, and hence a program's structure. Parnas introduced the concept of 'a family of programs' rather than a single program [2], as well as 'modularization as a mechanism for improving flexibility and comprehensibility of a system while allowing shortening of its development time' [5]. Computer programs became increasingly complex that forced separation of related functionality into sub-programs or modules, enabling management of complexity and verification of correctness [6].

The implementation of software as modules invoked related issues such as 'low coupling','high-cohesiveness' and 'the separation of concerns' that are adopted today as best practices within system design [6, 7]. The abstraction of software system functionality into modules, together with its interfaces is in essence *the determination of the architecture of the system*.

Furthermore, the term *layered architecture* is often used in the vernacular of IT researchers and system architects in the information systems application domain. In the early 80's, the concept layered architecture was used by [8] to model the proposed underlying network architecture of the International Standards Organisation's Open Systems Interconnection (ISO/OSI model). This model are widely integrated into network protocols and the Internet protocol TCP/IP, for instance, operates on the network and transport layers of the OSI model[9]. [10] used a layered architecture to describe a 'real-time distributed computing system from the functional, design, distribution and execution viewpoints'. To facilitate the vision of the Semantic Web, Berners-Lee, Hendler and Lassila [11] described the underlying language architecture as layers. Recently, Jeckle and Wilde [12] used the ISO/OSI layered architecture to describe a web services protocol stack. [13] introduced a layered framework for classifying and organising the descriptive models of an enterprise's architecture.

As the complexity of software systems grow, there is consensus among researchers and system architects that the determination of the architecture of a system is crucial to the successful understanding and development thereof, especially when the system envisaged is intricate and multifaceted. The concept *architecture* is commonplace in discussions of software engineering topics such as business process management and system engineering, but agreed-upon design and evaluation criteria are lacking in literature. Such criteria are on the one hand valuable for the determination of system architectures during the design phase, and on the other hand, provides a valuable tool for the evaluation of already existing architectures. The goal of this paper is thus to extract from literature and best-practices such a list of criteria.

In sections 2 and 3 we provide the reader with an overview of the terms architecture and layered architecture respectively, within the information system application domain. A list of evaluation and design criteria for layered architectures are compiled in section 4. We applied this evaluation criteria to the ISO/OSI Network model [8], as well as the proposed language architecture for the Semantic Web [11] in section 5. In conclusion it is our contention that the use of these evaluation criteria provide insight into the architectural requirements of systems based on layered architectures.

2 Architecture

Bass, Clements and Kazman [1] define the software architecture of a program or computing system as the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. In this context, an architecture describes elements and how elements relate to each other. It omits non-relevant information and is therefore an *abstraction* of the system. 'External visible properties' are the assumptions other elements can make of an element, such as services it provides or performance characteristics. Generally, systems comprise more than one structure, and no structure alone is *the* architecture. All systems encompass an architecture as any system can be viewed as a composition of elements with relationships among them.

[14] states that the architecture of a system is a comprehensive framework that describes its form and structure, including its components and their organization. Furthermore, 'architectural design represents the structure of data and program components that are required to build a computer-based system. It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of the system' [14, p.255].

Jacobson, Booch and Rumbaugh [15] define software architecture in the Unified Process methodology as: encompassing the significant decisions about the organization of a software system; the structural elements and their interfaces that will comprise the system together with their behaviour as specified in the collaborations among those elements; the composition of the structural and behavioural elements into progressively larger subsystems; as well as the architectural style that guides this organization.

[16] identifies two common elements of system architecture, viz. the highest-level breakdown of a system into its parts and the decisions that are hard to change. He also states that a system comprises of multiple architectures, and that the view of *what is architecturally significant*, can change over a system's lifetime.

From these definitions, we maintain that an architecture is a description of the components that encompass a system. The description of the components must include its organization or structure, its defining features or properties, as well as their relationships together with available interfaces that allows interaction with it. With these definitions in mind, we argue that the following are important criteria during evaluation of existing and development of new architectures:

– Architecture is defined within a *certain context*, where this context determines the important aspects of a system, the components necessary to realize the system, the properties of components as well as the relationships between components and external entities. This relates to the notion of multiple architectures or structures defined by views as introduced by [1], [16] and [15].
– An architecture is a *model* of the system in the given context, where a model is an abstraction of a real-world representation[17]. It is important to realize that a model provides a means to view *only* the significant aspects of the entire system.

Due to the progression of the design of architectural models, some architectural recurrences evolved. These are described as *architectural patterns*[1], also referred to as *architectural styles* [14]. A pattern is the description of a problem that occurs repetitively within a specific environment, as well as the core of the solution to that problem in such a way that the proposed solution can be reused. Patterns are rooted in practice and are referred to as *best practice descriptions* [16]. Examples of the best known architectural patterns include, but are not limited to, the client/server architectural pattern [18, 15], Peer-to-Peer architectural pattern [18, 15], three-tier architectural pattern [18, 15, 16], and the layered architecture or layers pattern [15].

From the above it suffice to note that the *layered architecture* is regarded as an architectural pattern or style that organizes functionality into layers. It can thus be regarded as an instance of an architecture and has to conform to the definitions of an architecture. In addition, layered architectures as a pattern provide best-practice solutions to recurring problems and is discussed in the next section.

3 Layered Architecture

One of the first examples of a layered architecture is Dijkstra's experimental layered operating system, developed at the Technische Hogeschool Eindnoven (THE). The goal of THE was to design and implement a provably correct operating system, by means of isolating various aspects of the operating system, into distinct layers [19]. Layers were isolated so that a specific layer only access its immediate neighbours [20, 14, 21].

The most well-known example of a layered architecture is probably the definition of network protocols found in the ISO/OSI model. This model defines all the methods and protocols required to connect computers by means of a network [8]. It separates the methods and protocols needed for network connectivity into seven different layers and each higher layer relies on services provided by a lower-level layer. The OSI model is an example of a closed layered architecture with low coupling because a layer may only access the layer immediately below it. However, each level introduces a speed and storage overhead [18, 20, 9, 22]. An example of an open layered architecture (where a layer can access the layer immediately below it, but also deeper layers) is the Swing user interface for Java [23].

Several other examples of layered architecture usage exist. The ISO/OSI network model comprises a formal specification. In contrast, the meaning of layered architectures in most other cases are implied, for instance [10] uses a layered architecture as a top-level organization to describe different viewpoints of real-time distributed computing systems, and [24] use a layered approach to assist with data interoperability on the Semantic Web.

Arguably, layering is a common best practice pattern used by software architects to break apart complex systems. In a layered architecture, the principal elements or components are arranged in the form of a layer cake where each layer rests on a lower layer. Generally, a layer represents a grouping of elements that provides related services. A higher layer might either use various services defined by the immediate lower layer only (closed architecture) or services by all the lower layers (open architecture). However, the lower layers are unaware of higher layers [16, 18, 25, 20].

According to [16], some of the benefits of breaking a system into layers include: (1) a single layer is viewed as a coherent whole without knowledge of the other layers, (2) the implementation of a specific layer can be substituted with alternative implementations of the same basic services, (3) dependencies between layers are minimized, (4) layers support standardization because they define how layers, as well as their interfaces, should operate, and (5) several higher-level services may reference a service provided by a lower-level layer.

Since a layered architecture is an instance of an architecture, for completeness it is necessary to map the elements of a layered architecture to concepts discussed in section

2. The *layers* of a layered architecture map to components, or a grouping of components, as referred to in section 2. In a layered architecture, the stacking and sequencing of layers are determined by relationships and organization of the architectural components.

4 Design and Evaluation Criteria for Layered Architectures

From the discussion of architecture concepts in section 2, and the description of layered architectures in section 3, we extract the following design and evaluation criteria. In the table below we indicate a possible question to be asked for evaluation purposes by 'Q'.

Criteria	**Description**
Clearly defined context	The context used to analyze the system determines its important aspects, assisting in the identification of the main components required to realize the system, its properties, its organization, as well as the relationships between components. *Q* Is it possible to identify the context from the description of the architecture?
Appropriate level of abstraction	The architecture model should be at a sufficiently high level of abstraction so that the system or subsystem under review can be viewed as a whole. Only the aspects of the system that are relevant at a certain level of abstraction should be visible at that level. *Q* Can the system within the context be viewed as a whole? *Q* Are there any components/properties/relationships in the architecture model that could be removed without losing important information at this level of abstraction?
Hiding of implementation details	This criterium supports the above criterium regarding the level of abstraction. Implementation details should be hidden in an architectural model. *Q* Are any implementation details visible in the description of the components/properties/relationships/structures of the architecture?

Clearly defined functional layers	This criterium relates to the determination of the architectural components and their grouping into the appropriate layers. Q Does the layer description specify a *function* of the layer within the system? Q Is the layer's function clear from its description and position in the architecture? Q Could the layer be removed without compromising the integrity of the system?
Appropriate layering, including well defined interfaces and dependencies	This criterium relates to the organization of the layers. The layers must clearly build on one another and its relationships and dependencies should be distinguishable, where any layer only accesses layers below it. This criterium also includes the specification of dependencies or access rules between layers, which is used to determine whether the architecture is open or closed. Q Do the layers clearly build on one another? Q Does a specific layer only require functionality defined by lower layers and not those of upper layers? Q Is it possible to determine whether the layered architecture is open or close?
Modularity	Components and hence layers should be modular. It should be possible to change the implementation of a layer as long as interfaces and functionality remains the same. Q Is it possible to replace the implementation of a layer with another implementation of the same functionality and interfaces without compromising the integrity of the layered architecture?

5 The Evaluation of Layered Architectures

As mentioned, this criteria can be used to design new architectures or evaluate existing ones. In order to establish and demonstrate the usefulness of this criteria, we evaluate two existing architectures in this section. We consider the ISO/OSI Model as it is a conceptual model or layered architecture that is used for the visualization and design of network functionality. It is furthermore considered to be an established specification, commonly used for network design. In addition, we consider the more recent proposed Semantic Web language architecture of [11].

The *ISO/OSI model* is a layered architecture that separates the methods and protocols required for network connectivity into seven different layers. Each higher layer relies on services provided by a lower-level layer [8, 9]. The layers are ordered from

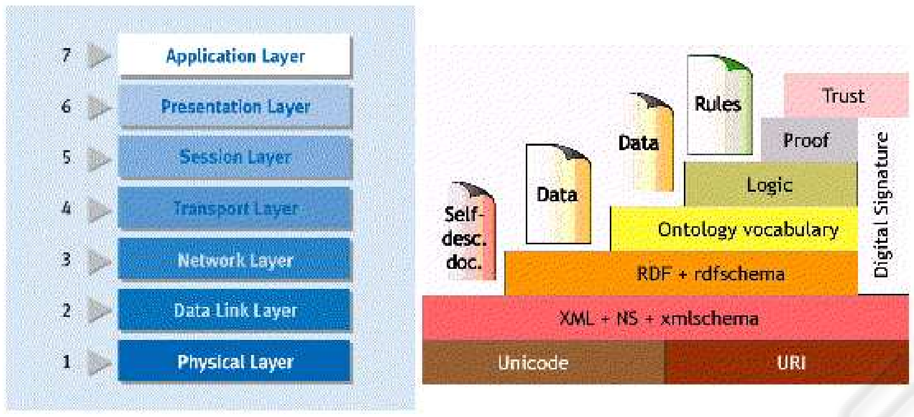

Fig. 1. The ISO/OSI Network Model and Semantic Web Architecture.

bottom to top and include: physical layer, data link layer, network layer, transport layer, session layer, presentation layer and application layer. See Figure 1.

In 2001 Tim Berners-Lee introduced the vision of the *Semantic Web* [11]. The proposed Semantic Web is an information space usable by machines rather than humans as is the case with the current Web. In addition, Tim Berners-Lee proposed a language tower or layered architecture depicted in Figure 1 [26]. The higher level languages use the syntax and semantics of the lower layers. Several Semantic Web authors have referred to and adopted this figure (also referred to as the Semantic Web layer cake) [27–31].

In the table below we evaluate these architectures against the criteria of section 4.

Criteria	**ISO/OSI Model**	**Semantic Web**
Clearly defined context	**Conform to:** The context of the ISO/OSI model is clearly defined as the protocol stack required for network interaction between computers.	**Conform to:** The context is stated to be the Semantic Web language architecture.

Appropriate level of abstraction	**Conform to:** All the layers required for network interaction are identified and the network is represented as a whole and no unnecessary information is displayed on any layer.	**Does not conform to:** It is possible to argue that the whole Semantic Web language architecture is visible. However, it is appropriate and commendable to remove information from the model. The top three layers define functionality, but the rest of the layers specify *existing technologies* rather than functionalities. It is not clear what are the function and interface to 'Digital Signatures' as a vertical layer, neither why 'Unicode' and 'URI' appear as two sections of the bottom layer.
Hiding of implementation details	**Conform to:** No unnecessary or implementation detail is visible on the architecture description.	**Does not conform to:** The bottom three layers as well as 'Digital Signatures' are implementation specifications or existing technologies.
Clearly defined functional layers	**Conform to:** All the layers have well-defined functionality descriptions, and their position within the architecture supports this functionality.	**Does not conform to:** The top three layers define functionality. It is not clear whether these are in relation to 'languages' as the architecture context specify, or applied functionality required for the implementation of the Semantic Web. The bottom layers specify existing technologies (such as 'XML' and 'RDF') rather than the functions embodied by these layers. The function of 'Digital Signatures', 'Unicode' and 'URI' are also not clear from their position on the architecture.
Appropriate layering, including well defined interfaces and dependencies	**Conform to:** Each layer build on the layer immediately below, implying that the architecture is closed. Each layer has an interface specification.	**Does not conform to:** The layers do not clearly build on one another. It is not clearly specified what the requirements of upper layers with regard to their lower layers are, and it is not possible to establish whether this is an open or closed architecture.
Modularity	**Conform to:** Different implementations of the layers exist and can be interchanged without negatively influencing the integrity of the architecture.	**Undefined:** It is not possible to determine the modularity of this architecture since the functionality and interfaces of the layers are not defined.

Using our proposed criteria, we established that the ISO/OSI model clearly conforms to all specified criteria. We can therefore conclude that this existing layered architecture is well designed. In contrast, the proposed Semantic Web layered architecture does not comply with the majority of established criteria. Further research might include an adaption of this architecture to conform to the criteria.

6 Conclusion

In this paper we presented an overview of different definitions and use of the term architecture, as well as features of layered architectures. Following from this investigation, theory and best-practices, a list of architectural design and evaluation criteria were derived. In order to demonstrate the efficacy of this criteria list, we evaluated two layered architectures obtained from literature. We contend that this criteria list can assist researchers and system architects to evaluate and design an architecture in general, and a layered architecture in particular.

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison Wesley Professional (2003)
2. Parnas, D.L.: Designing software for ease of extension and contraction. In: ICSE '78: Proceedings of the 3rd international conference on Software engineering, IEEE Press (1978) 264–277
3. Weiner, L.H.: The roots of structured programming. In: Papers of the SIGCSE/CSA technical symposium on Computer science education, New York, NY, USA, ACM Press (1978) 243–254
4. Dijkstra, E.W.: The end of computing science? Commun. ACM **44** (2001) 92
5. Parnas, D.: On the criteria to be used in decomposing systems into modules. Communications of the ACM **15** (1972) 1053 – 1058 accessed 19 October 2005.
6. Parnas, D.L., Clements, P.C., Weiss, D.M.: The modular structure of complex systems. In: ICSE '84: Proceedings of the 7th international conference on Software engineering, IEEE Press (1984) 408–417
7. Schach, S.R.: Introduction to object-oriented analysis and design with UML and the Unified Process. Irwin McGrawhill (2004)
8. Zimmermann, H.: Os1 reference model-the is0 model of architecture for open systems interconnection. IEEE TRANSACTIONS .ON COMMUNICATIONS (1980) accessed 19 October 2005.
9. Hallberg, B.: Networking: A Beginner's Guide. Second Edition. Osborne / McGraw-Hill (2001)
10. Simpson, H.R.: Layered architecture(s) : Principles and practice in concurrent and distributed systems. In: 1997 Workshop on Engineering of Computer-Based Systems (ECBS '97). (1997) 312
11. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. The Scientific American **May 17, 2001** (2001) accessed 20 February 2004.
12. Jeckle, M., Wilde, E.: Identical principles, higher layers: Modeling web services as protocol stack. In: XML Eurpe 2004, Amterdam. (2004) accessed 15 October 2005.
13. Zachman, J.: The framework for enterprise architecture: background, description and utility. Zachman International Website (2003)
14. Pressman, R.S.: Software Engineering: A Practitioner's Approach. sixth edition edn. McGraw-Hill (2005)

15. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999)
16. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley (2003)
17. Avison, D., Fitzgerald, G.: Information Systems Development: Methodologies, Techniques and Tools. Third edition. McGraw-Hill (2003)
18. Bruegge, B., Dutoit, A.H.: Object-oriented Software Engineering using UML, Patterns, and Java. Second edition edn. Prentice-Hall (2004)
19. Dijkstra, E.W.: The structure of the the-multiprogramming system. Commun. ACM **11** (1968) 341–346
20. Nutt, G.J.: Centralized and Distributed Operating Systems. Prentice-Hall International Editions (1992)
21. Brooks, F.P.: The Mystical Man-month. Addison-Wesley Publishing Company (1975)
22. Popescu-Zeletin, R.: Implementing the iso-osi reference model. In: SIGCOMM '83: Proceedings of the eighth symposium on Data communications, ACM Press (1983) 56–66
23. Sun: Swing package for java. Sun Website http://sun.java.com (2003)
24. Cruz, I.F., Xiao, H.: Using a layered approach for interoperability on the semantic web. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE03). (2003)
25. Bachman, C.: Personal chronicle: Creating better information systems, with some guiding principles. IEEE Transactions on Knowledge and Data Engineering (1989) 17–32
26. Berners-Lee, T.: Semantic web - xml2000. W3C Website (2000) accessed 11 August 2004.
27. Fensel, D.: Language standardization for the semantic web: The long way from oil to owl. In: Distributed Communities on the Web: 4th International Workshop, DCW 2002, Sydney, Australia. Volume 2468 / 2002. (2002) 215–227 accessed 15 March 2005.
28. Hendler, J.: Agents and the semantic web. IEEE Intelligent Systems **16** (2001) 30–37
29. Oberle, D., Staab, S., Studer, R., Volz, R.: Supporting application development in the semantic web. ACM Trans. Inter. Tech. **5** (2005) 328–358
30. Patel-Schneider, P.F., Fensel, D.: Layering the semantic web: Problems and directions. In: Proceedings of The Semantic Web - ISWC 2002: First International Semantic Web Conference, Sardinia, Italy. Volume 2342 / 2002., Springer-Verlag GmbH (2002) 16 accessed 15 March 2005.
31. Thuraisingham, B.: Security issues for the semantic web. In: Proceedings of the 27th Annual International Computer Software and Applications Conference, IEEE (2003) 632 accessed 31 March 2005.