General Purpose ADL (Architecture Description Language)

COMS W4115

Alan Khara

Ask2206

February 11, 2014

1.0 Introduction

In the design of system architecture[1], a blueprint of the system typically comes second, after requirements are gathered in the process of architecture development. This is followed by design verification and, later, implementation. Initially, the domain of Architecture Description Languages (ADLs) was confined to the design phase and was highly motivated by Object Oriented paradigm [1] [2] [3]. ADLs were mainly graphical in nature, with numerous Line and Box representations that were later standardized by Object Management Group (OMG) in Unified Model Language[2] (UML) [4][5]. Being quickly adopted by industry, UML was infested with ambiguous features: a given relationship between components of a system could be ambiguous and yield two different interpretations of the same drawing [2]. Research during the last decade has worked to solve this problem on two different fronts. One line of research seeks to extend UML in order to make it less ambiguous [6] [7] [8] [9]. Another kind of research is finding new ADLs (graphical and text based) that have more rigorous and well defined semantics [10] [11] [12][13]. Both of these trends are important with respect to defining the problem scope of the proposed project.

Because they are mainly driven by industry, efforts to extend UML are inherently domain-specific [14]. The primary motivation behind the development of UML is to overcome ambiguity and reduced the communication gap that exists among various stakeholders. This is tackled by either introducing constraints or adding new vocabulary. Object Constraint Language (OCL), the first such effort, was introduced by IBM and became part of UML since its inception. The main purpose of OCL was to set pre and post conditions on operations and make UML more precise [15] [16]. Even though OCL was able to reduce ambiguity in UML, it was unsuccessful in tackling the problematic communication gap among stakeholders. While developing a system, there are many stakeholders: project managers, application developers, system analysts, and domain engineers. This requires a common platform where the different viewpoints of these groups can be incorporated and clearly communicated. This failure to deal with diverse viewpoints is the essence of the communication gap. The efforts that have been to reduce this gap can be explained by the research that introduced Semantics for Business Vocabulary and Rules (SBVR). In this research, UML/OCL is transformed to include business policies and rules [8]. It is important to note that extending UML for architecture development is considered to be more practical and is currently the dominant trend within the industry. However, this approach has the serious limitation of being used only for description of the system, and not in any other phases of development, like verification, implementation, and testing [17] [18]. In addition, there

[1] Representation system having components, their behaviour and interaction
[2] Latest version is UML 2.4.1 published in August 2011 by Object Management Group (OMG)

is no provision for checking whether the final model is faithful to the original design. It also does not provide any analysis functions [19].

New ADLs with more rigorous semantics were built when it became clear that Line and Box representations cannot be extended enough to achieve the complex architecture, and that graphical representation has its own limitations. Some of the prominent ADLs proposed during the last two decades include ADAGE (1993) [20], Meta-H (1993) [21], Aesop (1994) [22], Rapide (1995) [23], Darwin (1995) [24], C2 (1996) [25], Unicon (1996) [26], Wright (1996) [27], Software Architecture Description Language SADL (1997) [28], Acme (2000) [29], xADL (2001) [30], π-ADL (2004) [31], Architecture Analysis and Design Language AADL (2006) [32]. All these languages have different capabilities. The need to categorize and classify the existing ADLs became greater as the area matured and became larger, and several classification systems have been proposed [18] [33] [34]. These classifications, in conjunction with the development of Component Based Software Engineering (CBSE), resulted in some generalized principles of Architecture Description [35] [36], which in turn started blurring the difference among requirement languages, modelling languages and programming languages [36]. Even though most of the new ADLs are called description languages, they have the capacity to model, design and verify the architectures. Despite ADLs achieving Turing completeness and being capable of modelling complex architecture, industry is still hesitant in adopting these technologies [18] [31] [37]. Only the domain-specific versions of ADLs are in use [20] [29] [32].

In a nutshell, research in ADLs is advancing in two different dimensions. The first dimension includes extensions to UML with the objective of making it more useful and less ambiguous. Due to UML's inherent graphical limitation, it cannot be used in other aspects (verification, implementation, analysis) of system development. In the second dimension, there is the development of formalized ADLs capable of fulfilling industry needs. Only a language from the domain of formalized ADLs can promise to be comprehensive and usable in all aspects of system development. There is not yet an ADL that successfully fulfills these requirements, and understanding this issue will help clarify the problem scope of the proposed project.

2.0 Problem Scope

Several Surveys have been conducted to find the incapacity of current formalized ADLs; that is, to determine why they are not fulfilling industry needs [18] [37]. Industry architectural requirements, gathered from surveys and ISO/IEC/IEEE 42010 standard, can be described as follows:

1. Architectural Description (AD) can be easily communicated to all concerned stakeholders

2. ADL helps creating, refining and validating system architecture

3. ADL can handle real-time constructs at the design level

4. ADL can handle static, dynamic and mobile architectures

5. ADL supports the representation of non-architectural information (like scenarios and requirements)

6. ADL should have a consistency check and completeness rules

7. ADL can capture design rationale or history of the design decision

8. ADL can handle data and control flow

9. ADL allow domain-specific extensions to satisfy architectural issues specific to the concerned domain

10. ADL can present implementation-related information and develop prototypes

11. ADL must have analytical capacity and the ability to generate various viewpoints

12. ADL provides extensive toolset support

Many ADLs are capable of meeting a large subset of the above requirements. For example, AADL and Meta-H can model real-time embedded systems in avionics [21] [32]. Using event-based style, C2 can describe the user interface system [25]. Rapide can simulate and analyze architectural designs [23] [39]. Π-ADL can support dynamic and mobile architectures becoming a second generation ADL Language [31]. Yet, no ADL is capable of meet all the above-noted requirements. Several attempts have been made to develop an ADL that can fulfill industry requirements [40], but most of these attempts were domain-specific and resulted in rewriting semantics of ADL to comply with changing domain requirements [41] [42]. In fact, it has been suggested that a general purpose ADL may not exist [37]. Therefore, research focus must return to domain-specific ADLs. Contrary to the recommendation, this proposal is attempting to introduce semantics and description of general purpose ADL that can fulfill all of the requirements.

<u>3.0 Language Description</u>

To fulfill the requirements noted in the problem scope, the underlying semantics of ADL needs to be based on some general theory of Architecture Description. There are proposals for such a theory that contain first class design entities like components, connectors, styles, representations, visualizations and extra functional properties [18] [22] [29][31] [43]. Semantics of a general purpose ADL requires theory that is more general, such that it can be accepted in all domains. One such candidate is Systems Theory (ST), which is common to all disciplines of engineering [44]. An abstraction of ST can provide a common language for all stakeholders while also allowing the capacity needed for domain-specific extensions. In addition, it is capable of fulfilling the requirements noted in the problem scope of this proposal, as will be explained further. Moreover, current principles in Architecture Description can, upon close examination, be viewed as a subset of Systems Theory.

In Systems Theory, structure [45] [46] contains three parts: system, boundary and surroundings. Based on the nature of the boundary, there can be three types of systems:

1) Open System: energy and material can transfer across the boundary;

2) Closed System: only energy can flow across the boundary; and,

3) Isolated System: no interaction between system and surroundings take place.

At the level of abstraction, these types can be defined by data flow. Another important classification of a system is based on its behavior: whether it is controlled or uncontrolled.

Control flow can easily be abstracted out of this classification. It is data flow and control flow that make the system static or dynamic. Ultimately, all kinds of architectures are developed by using abstractions for modelling system structure and behavior. This proposal attempts to present the semantics of a general purpose ADL based on Systems Theory, which is explained as a superset of current Architectural Description Theories. Only a subset of the proposed language will be implemented, so as to comply with the realistic expectations of the project within the proposed timeframe.

Current ADLs constructs are based on two different kinds of architecture: Interface Connection Architecture (ICA) and Object Connection Architecture (OCA) [10] [12]. Former is more common among this domain of languages than latter. The commonality between these two architectures is the concept of Modules or Components, which are linked with connections representing the whole structure. The main difference in these architectures is that ICA emphasizes more on constraints in connections, where OCA does not. These architectures will be discussed in greater detail in the Final Report of this Project.

Systems theory considers system as a set of sub-systems, which are smaller systems itself. System is dynamic when it is changing its states. It accepts input, and provides output, which is observable state of the system. In a strict mathematical sense, system can be represented as a tuple, or ordered set as [48] [49] [50]:

System $< T, U, Y, Q, \Omega, \delta, \lambda >$

- T is Time Set
- U is Input Set
- Y is Output Set
- Q is States Set
- Ω is admissible Input function Set U(T)
- δ is defined as $\delta: Q \times \Omega \rightarrow Y$, this function derives dynamism in the system
- λ is defined as $\lambda: Q \rightarrow Y$, function that maps State Set to Output Set

If one replaces component in architectural description theory to sub-system, one can analyze system to any level of complexity. This definition will add richness to the concept of structure description, and does not contradict the current architectural frameworks. On the contrary, current architectural frameworks can be reduced from the systems theory. One of the criticisms of present architectural frameworks is that they do not have concept of hierarchy and class inheritance built in semantics [18]. In systems approach, these concepts are built in the definitions. For example, System composed of sub-systems is classic representation of hierarchy. Final report will contain all the terms related to the Systems Theory. As part of the proposal, only syntax is described here.

3.1 Syntax

3.1.1 Comments

To comply with other language constructs, comments will be written between two double colons. For example -

:: This is a Comment ::

3.1.2 Variables Declaration

Variables types are declared as follows:

>>Energy (float) :: Variable Name followed by data type in parenthesis ::

3.1.3 Data Types

3.1.3.1 Integers

Integer Values are assigned in the same way as other variable assignments

>> X (Integer)

>> X = 50

3.1.3.2 Booleans

Four-state logic will be implemented. This is very important in the study of complex systems. As complexity increases Multi-value logic can be implemented. This is based on Belnap's Logic Scheme [51].

>> State (Boolean)

>> State = TRUE

>> State = FALSE

>> State = BOTH

>> State = NEITHER

3.1.3.3 Floating-Point Numbers

Floating-Point Values are assigned in the same way as other variable assignments

>> X (float)

>> X = 50.00 :: Decimal Value will declare the variable as float ::

3.1.3.4 Alpha Numeric Strings

Strings will be declared in double quotes. For Example –

>> Language (String)

>>Language = "ADL is Architecture Description Language"

Quotes inside quote will be declared with single quote. For example –

>>Language = “ADL is ‘Architecture Description Language’”

3.1.3.5 Constants

Constants will be declared with adjective to the data type in parenthesis, as follows:

>> TIME_MAX (constant integer) = 56

3.1.3.6 Formula

In Systems theory, admissible Input Values and Output is often defined as a parameterized or non-parameterized function. Moreover, there are pre and post conditions which can be functions as well. To capture that abstraction, data type formula is introduced. This will allow building the system, without worrying about the functions at the start.

>> Energy (Formula)

>> …

>> Energy = Temperature_Change * specific heat * mass :: Right hand side are other declared variables in the program ::

3.1.3.7 System

Language is based on System oriented approach, and system will follow the strict mathematical definition under System theory:

>> System <= Boiler :: This is somewhat like Class declaration under Object Oriented Paradigm ::

System is composed of sub-systems, called Components, and hierarchy is built in:

>> Boiler <= Burner, Flask, Water, Steam, Heat_Sensor :: Right side is the set of sub-systems in a system::

3.1.3.8 Tuple

Sub-System, as a system itself will be defined as a set of variables, called Tuple (as advanced data type, much like array or list). Like below:

>> Heat_Source :=> <[Energy Given] (float), Cost (formula), [Burner Factor] (Constant)>

Access to tuple point will be based on the index, which starts with 1 or with the name of the variable.

```
>> Real_Cost = Heat_Source![2]

>> Real_Cost = Heat_Source!Cost
```

3.1.3.9 Higher Order Tuple

A tuple can contain a set of tuples, becoming higher order tuple. This concept will help adding layers of complexity.

```
>> Boiler <Heat_Source<>, Flask<>, Water<>, Steam<>>
```

Important difference of above concept from double arrays is that a tuple can be of any dimension, and different dimensional tuples can be contained in higher order tuple.

3.1.4 Control Flow

For pre and post conditions, logic plays an important part. Standard Logical operators will be used.

```
>> if Boiler!Heat_Source!Cost > MAX_Cost AND Boiler!Heat_Source!Energy < MAX_ENERGY

>>      Set State = BOTH

>> Otherwise

>>      State = NEITHER .
```

If / Otherwise end with '.' Period. Indentation plays important role in this language as readability is very central to the whole concept.

3.1.5 Functions

The parallel to function in this language is the construct of sub-system. It accepts an input and throws an output.

3.1.6 Loops

To facilitate analysis of systems design, loops play the central role. One can define, cycle of states and real time variables can be computed. This will be used as bellow:

```
>> while Burner!Energy!Cost >= MAX_COST

>>      continue Burner!Burning_State.
```

Loops will end with '.' Period. Indentation plays important role in this language as readability is very central to the whole concept.

3.1.7 Other Proposed Constructs

Some other constructs are planned, which include if variable name is two words with space, then square brackets are used Heat_Sink can also be [Heat Sink], and this can be done interchangeably. The reason to keep this notation is to comply with existing ADLs.

4.0 Representative Program

Here a dynamic physical system is constructed. More complexity will be added to this program at later date to show how complexity can be added and constructs can be reused. The purpose of this program is to construct a Boiler Operation, and check all vulnerabilities to come up with a design decision. It should be noted that how history of design decision is codified in the whole program and how hierarchy is proven.

System <=: Boiler
Components <=: [Heat Source], Flask, Water, Steam
Environment <=: [Heat Sink]
:: Basic concepts of System Model common to all stakeholders, recognized in problem scope::

::Description Environment::

[Heat Sink] :=> Humidity (Float), [Energy lost] (formula), cost (constant), [Reliability Range] (constant)

::Description System::

Heat_Source :=> <Energy_Given (float), Cost (formula), Burner_Factor (Constant)>

Flask :=> <Energy_received (formula), Internal_radius (float), External_Radius (float), Pressure_Capacity (formula), Volume (formula), Cost (formula), Thickness (formula), Material_Security_Factor (constant), Material_Cost_Factor (constant), State (formula)>

Water :=> <Volume (formula), Energy_Transferred (formula), Cost (formula), Density (Constant)>

Steam :=> <Volume (Formula), Pressure (Formula)>

::Interconnections between system components::

[Heat Sink]![Energy Lost] = [Humidity]*.5

[Heat Source]!Cost = [Average Energy Given]* [Burner Factor]

Flask![Energy Received] = ([Heat source]![Energy Given] – [Heat Sink]![Energy lost]) – [Heat Sink]![Reliability Range]
Flask![Thickness] = [External Radius] – [Internal Radius]
Flask![Pressure Capacity] = Thickness*[Material Security Factor]
Flask!Volume = 4/3*Pi (Thickness)^3
Flask![Cost] = Flask!Volume * [Material Cost Factor]

Water!Volume = ½*Flask!Volume
Water![Energy Transferred] = Flask![Energy Received]/Flask!Thickness
Water!Cost = Water!Volume*6

Steam!Volume = Water!Volume
Steam!Pressure = Steam!Volume*[Quality Factor]

::Vulnerability::
::Control Loop Logic::

```
If Steam!Pressure > Flask![Pressure Capacity]
        Set Flask!State = FALSE.
```

:: Constraints - Consistency and completeness rules, recognized in problem scope::

[Heat Source]!Cost =< 5000
Flask!Cost =< 20000
Water!Cost =< 500

:: Materials, similar to object instantiation::

[Env Open] !! [Heat Sink]
{Humidity = 78
[Reliability Range] = 5
Cost = 0}

[Env Controlled] !! [Heat Sink]
{Humidity = 78
[Reliability Range] = 0.5
Cost = 400}

[Burner V1] !! [Heat Source]
{[Average Energy Given] = 890
[Burner Factor] = 9}

[Burner V2] !! [Heat Source]
{[Average Energy Given] = 920
[Burner Factor] = 10}

[Green Glass] !! [Flask]
{[Material Security Factor] = 9
[Material Cost Factor] = 10}

[Red Glass] !! [Flask]
{[Material Security Factor] = 10
[Material Cost Factor] = 7}

[Lec Glass] !! [Flask]
{[Material Security Factor] = 18
[Material Cost Factor] = 15}

[Saline] !! [water]

{ Density = 1 }

[Saline Steam] !! [Steam]
{[Quality Factor] = 6}

:: Run Tests, prototypes formation recognized in problem scope::

::Option 1:: Boiler<Burner V1, Green Glass, Saline, Saline Steam>< [Env Open]>

:: Above option will check the consistency of the system, based on the components used and explain if this proposal meet all constraints::

Option 2: Boiler<Burner V2, Red Glass, Saline, Saline Steam)>

:: See the history of design proposals is codified here ::

Option 3: Boiler<Burner V2, Green Glass, Saline, Saline Steam)>

Show Tuple<1> ::kind of analysis output, analytical capacity, recognized in problem scope::

5.0 References

[1] Vahid Garousi. 2012. Classification and trend analysis of UML books (1997---2009). *Softw. Syst. Model.* 11, 2 (May 2012), 273-285.

[2] Robert Allen and David Garlan. 1997. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.* 6, 3 (July 1997), 213-249.

[3] Perry, Dewayne E., and Alexander L. Wolf. "Foundations for the study of software architecture." *ACM SIGSOFT Software Engineering Notes* 17.4 (1992): 40-52.

[4] Hamilton, Marc. *Software Development: A Guide to Building Reliable Systems*. Prentice Hall Professional, 1999.

[5] *Unified Modeling Language Specification*. International Standard ISO/IEC 19501:2005 (E).

[6] Pérez-Martínez, Jorge Enrique, and Almudena Sierra-Alonso. "UML 1.4 versus UML 2.0 as languages to describe software architectures." *Software Architecture*. Springer Berlin Heidelberg, 2004. 88-102.

[7] Ivers, James, et al. *Documenting component and connector views with UML 2.0*. No. CMU/SEI-2004-TR-008. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2004.

[8] Cabot, Jordi, Raquel Pau, and Ruth Raventós. "From UML/OCL to SBVR specifications: A challenging transformation." *Information Systems* 35.4 (2010): 417-440.

[9] Manfred Broy and María Victoria Cengarle. 2011. UML formal semantics: lessons learned. Softw. Syst. Model. 10, 4 (October 2011), 441-446.

[10] José Meseguer and Grigore Roşu. 2013. The rewriting logic semantics project: A progress report.*Inf. Comput.* 231 (October 2013), 38-69.

[11] Feiler, Peter H., and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[12] Alexandre Rademaker, Christiano Braga, and Alexandre Sztajnberg. 2005. A Rewriting Semantics for a Software Architecture Description Language. *Electron. Notes Theor. Comput. Sci.* 130 (May 2005), 345-377.

[13] Forget, Julien, et al. "A real-time architecture design language for multi-rate embedded control systems." *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 2010.

[14] François Pinet, Magali Duboisset, and Vincent Soulignac. 2007. Short communication: Using UML and OCL to maintain the consistency of spatial data in environmental information systems. *Environ. Model. Softw.* 22, 8 (August 2007), 1217-1220.

[15] R. K. Pandey. 2011. Object constraint language (OCL): past, present and future. *SIGSOFT Softw. Eng. Notes* 36, 1 (January 2011), 1-4.

[16] Warmer, Jos, and Anneke Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[17] Pandey, R. K. "Architectural description languages (adls) vs uml: a review."*ACM SIGSOFT Software Engineering Notes* 35.3 (2010): 1-5.

[18] Clements, Paul C. "A survey of architecture description languages."*Proceedings of the 8th international workshop on software specification and design*. IEEE Computer Society, 1996.

[19] Manfred Broy and María Victoria Cengarle. 2011. UML formal semantics: lessons learned. Softw. Syst. Model. 10, 4 (October 2011), 441-446.

[20] COGLIANESE, LOUISH, and Raymond Szymanski. "DSSA-ADAGE: an environment for architecture-based avionics development." *AGARD, Aerospace Software Engineering for Advanced Systems Architectures 8 p(SEE N 94-29315 08-61)* (1993).

[21] Binns, Pam, and Steve Vestal. "Formal real-time architecture specification and analysis." *IEEE Real-Time Systems Newsletter* 9.1-2 (1993): 104-108.

[22] Garlan, David, Robert Allen, and John Ockerbloom. "Exploiting style in architectural design environments." *ACM SIGSOFT Software Engineering Notes*. Vol. 19. No. 5. ACM, 1994.

[23] Luckham, David C., et al. "Specification and analysis of system architecture using Rapide." *Software Engineering, IEEE Transactions on* 21.4 (1995): 336-354.

[24] Magee, Jeff, et al. "Specifying distributed software architectures." *Software Engineering—ESEC'95*. Springer Berlin Heidelberg, 1995. 137-153.

[25] Medvidovic, Nenad, et al. "Using object-oriented typing to support architectural design in the C2 style." *ACM SIGSOFT Software Engineering Notes*. Vol. 21. No. 6. ACM, 1996.

[26] Zelesnik, Gregory. "The unicon language reference manual." *School of Computer Science Carnegie Mellon, Pittsburgh, Pennsylvania (May 1996)*(1996).

[27] Allen, Robert, and David Garlan. "The Wright architectural specification language." *Rapport technique CMU-CS-96-TBD, Carnegie Mellon University, School of Computer Science* (1996).

[28] Moriconi, Mark, and Robert A. Riemenschneider. *Introduction to SADL 1.0: A language for specifying software architecture hierarchies*. Technical Report SRI-CSL-97-01, SRI International, 1997.

[29] Garlan, David, Robert T. Monroe, and David Wile. "Acme: Architectural description of component-based systems." *Foundations of component-based systems* 68 (2000): 47-68.

[30] Khare, Rohit, et al. "xADL: enabling architecture-centric tool integration with XML." *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE, 2001.

[31] Oquendo, Flavio. "π-ADL: an Architecture Description Language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures." *ACM SIGSOFT Software Engineering Notes* 29.3 (2004): 1-14.

[32] Feiler, Peter., Gluch, David., & Hudak, John. 2006.*The Architecture Analysis & Design Language (AADL): An Introduction* (Technical Report CMU/SEI-2006-TN-011). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

[33] Medvidovic, Nenad, and Richard N. Taylor. "A framework for classifying and comparing architecture description languages." *ACM SIGSOFT Software Engineering Notes*. Vol. 22. No. 6. Springer-Verlag New York, Inc., 1997.

[34] Medvidovic, Nenad, and Richard N. Taylor. "A classification and comparison framework for software architecture description languages." *Software Engineering, IEEE Transactions on* 26.1 (2000): 70-93.

[35] Robert Allen and David Garlan. 1997. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.* 6, 3 (July 1997), 213-249.

[36] Crnkovic, Ivica, et al. "A classification framework for software component models." *Software Engineering, IEEE Transactions on* 37.5 (2011): 593-615.

[37] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. 2013. What Industry Needs from Architectural Languages: A Survey. *IEEE Trans. Softw. Eng.* 39, 6 (June 2013), 869-891.

[38]

[39] Frederic D. Mckenzie, Mikel D. Petty, and Qingwen Xu. 2004. Usefulness of Software Architecture Description Languages for Modeling and Analysis of Federates and Federation Architectures.Simulation 80, 11 (November 2004), 559-576

[40] Qian Wang. 2005. Integrating architecture description languages: a semantics-based approach. In *Proceedings of the Second international conference on Distributed Computing and*

Internet Technology (ICDCIT'05), Goutam Chakraborty (Ed.). Springer-Verlag, Berlin, Heidelberg, 434-445.

[41] Braga, Christiano, and Alexandre Sztajnberg. "Towards a rewriting semantics for a software architecture description language." *Electronic Notes in Theoretical Computer Science* 95 (2004): 149-168.

[42] Rademaker, Alexandre, Christiano Braga, and Alexandre Sztajnberg. "A rewriting semantics for a software architecture description language." *Electronic Notes in Theoretical Computer Science* 130 (2005): 345-377.

[43] Garlan, David. "Software architecture and object-oriented systems."*Proceedings of the IPSJ Object-Oriented Symposium*. 2000.

[44] Von Bertalanffy, Ludwig. "{General System Theory}." *General systems* 1 (1956): 1-10.

[45] Austin M.A., Modelling Systems Structure and Behaviour, Presented at Institute for Systems Research, University of Maryland, College Park, 2012.

[46] Selberg S. and Austin M., Toward An Evolutionary SoS Architecture, Presented to the INCOSE 2008 Symposium, Utrecht, The Netherlands, June 19, 2008.

[47] Skyttner, Lars. *General systems theory: ideas & applications*. World Scientific, 2001.

[48] Polderman, Jan Willem, and Jan C. Willems. *Introduction to mathematical systems theory: a behavorial approach*. Vol. 26. Springer, 1998.

[49] Hinrichsen, Diederich, and Anthony J. Pritchard. *Modelling, State Space Analysis, Stability and Robustness*. Vol. 1. Springer, 2004.

[50] Lin, Yi, ed. *General systems theory: A mathematical approach*. Vol. 12. Springer, 1999.

[51] Belnap Jr, Nuel D. "A useful four-valued logic." *Modern uses of multiple-valued logic*. Springer Netherlands, 1977. 5-37.