3rd INTERNATIONAL CONFERENCE ON RESEARCH IN SCIENCE AND TECHNOLOGY
BERLIN .GERMANY 9 July, 2016
RSTCONF
سومین کنفرانس بین المللی پژوهش در علوم و تکنولوژی
برلین - آلمان ۱۹ تیرماه ۱۳۹۵

Architecture Description Language (ADL): a survey

Zeynab Salamati
Department of Computer Engineering, Islamic Azad University, Birjand, Iran
Zeynab.salamat@gmail.com
Kazem Nikfarjam
Department of Computer Engineering, Azad University, Birjand, Iran
nikfarjam@iaubir.ac.ir
Mohammad Mehmandoost
Department Computer Engineering, Azad University, Ghazvin, Iran
Mohammad_mehmandoost@hotmail.com

Abstract:

Architecture Description Languages (ADLs) are emerging as viable tools for formally representing the architectures of systems. Software architectures shift the focus of developers from lines-of-code to coarser-grained architectural elements and their overall interconnection structure. Architecture description languages (ADLs) have been proposed as modelling notations to support architecture-based development. In this paper we analyse ADLs and classify kind of the ADLs that have been proposed recently. Preliminary results allow us to draw conclusions about what constitutes an ADL, and how contemporary ADLs differ from each other and will try as much as possible to introduce visual studio any language.

KEYWORDS: SOFTWARE ARCHITECTURE, (ADL) ARCHITECTURE DESCRIPTION LANGUAGE, NEW VERSION OF ADLS, VISUAL STUDIO

1. **INTRODUCTION:** Architecture description languages (ADLs) are the means by which software architectures are defined. ADLs enable software architects to express high level system structure by describing its coarse-grained components and connections among them. They are languages designed to model a system. They have often graphical as well as plain text syntax. The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. They shift focus from lines-of-code to software components and their overall interconnection structure. We should consider ADL not a programming language or object-oriented modeling notation or formal specification language, ADL is composed of several simple constructs: components, roles, connectors, ports, and protocols [1]. Types of Architectural elements are: GUI, WebUI, command line, message driven server, server, batch program, data loader, data stores, system database, file, external entities, subsystem, external system, external data source [2].*A* component is a reusable black/grey-box entity (a piece of code) with well-defined interface and specified behaviour which is intended to be combined with other components to form a software system (an application). A component can usually have multiple interfaces, some to provide services to the component's clients, others to require services from the environment. Components can be nested to form hierarchies; a higher-level component can be composed of several mutually interconnected, cooperating subcomponents. The basic architecture of a component is defined by a component model. A component model specifies the structure of component interfaces, the mechanisms by which a component interacts with its environment, component internal structuring, etc. Basically, the component model provides guidelines as to how to create and implement components and how to assemble them into a larger application. Interfaces are the means by which components connect. A connector frame is

represented by a set of named roles. In principle, a role is a generic interface of the connector intended to be tied to a component interface. In the context of the frame, a role is either in the provides role or the requires role position. A provides role serves as an entry point to the component interaction represented by the connector type instance and it is intended to be connected to a requires interface of a component (or to a requires role of another connector). Similarly, a requires role serves as an outlet point of the component interaction represented by the connector type instance and it is intended to be connected to a provides interface of a component (or to a provides role of another connector) [3]. Types of Architectural connectors are: RPC, direct invocation, database data flow, file data flow, system messaging [2].The combination of component and connector is configuration. Architectural configurations, or topologies, are connected graphs of components and connectors that describe architectural structure. This information is needed to determine whether appropriate components are connected, their inter-faces match, connectors enable proper communication, and their combined semantics result in desired behaviour. [4]. The resulting configurations are indeed generic, compositional and reusable. An interface provided by a component can be viewed as a set of operations implemented by the component. It corresponds to a procedural interface of a traditional library or to an object interface. In addition to being a set of operations, an interface also serves as the contract between a component and its environment which separates providers implementing the component interface from clients using this interface. (By publishing its interface, a component tells the outside world about the services it provides. By invoking operations upon the published interface, clients access the component's services as a black-box.) [3]. A common concept among ADLs is the division of a component into interface and implementation. The interface is the connection of the component to other component, such as ports. The implementation takes care of the intern parts of the component. In papers that have been presented, Authors have focused on the overall classification of architecture description languages. In this paper we will try to classify ADLs in more details and small classifications, we will classify ADLs and new or improved versions of them in separate categories.

2. Relation between ADL and language programming

Unfortunately, it's not any specified criteria that differ ADLs from programing languages, requirements languages, modelling languages. In principle, ADLs differ from requirements languages because the latter describe problem spaces whereas the former are rooted in the solution space and they differ from programming languages because the latter bind all architectural abstractions to specific point solutions whereas ADLs intentionally suppress or vary such binding. ADLs differ from modelling languages because the latter are more concerned with the behaviors of the whole rather than of the parts, whereas ADLs concentrate on representation of components [5].

We can say ADL is a Visual Programming Language (VPL) and its incorporation by Trading Technologies marks a new and potentially important programming innovation for the algorithmic trading community [6].

3. Strengths and Weaknesses of ADLs

Strengths

- ADLs represent an unambiguous formal way of representing software architecture.
- As majority of ADLs are textual therefore machine read-able and suitable for automation.
- ADLs support describing a system at higher level of abstraction.

- Because of its formal representation characteristics they allow analysis of architecture's correctness, complete-ness, consistency, ambiguity and performance.
- ADLs support automatic generation of software systems.

Weaknesses:

- Many of the ADLs are textual and such less appealing for the software architects of other domains than only those for which ADL was created.
- Most of the ADLs were created for domain specific applications like avionics etc. and therefore suitable only for those domains.
- They lack features which are highly desirable by the persons (software architects) who use them; one of them may be being graphical perhaps.
- One of the biggest drawbacks of ADLs is they lack supporting tools barring except a few [7].

4. Types of Architecture Description Language

4.1. AADL: The Architecture Analysis & Design Language (AADL) is a large and complete language intended for design both the hardware and the software of a system. It is an industrial modeling standard used in avionics, aerospace, automotive, medical devices, and robotics to describe an embedded real-time system as an assembly of soft-ware components mapped onto an execution platform. In AADL, a component type specifies the component's interface and properties, and a component implementation specifies its internal structure as a set of subcomponents and a set of connections linking their ports. An AADL construct may have properties describing its parameters, declared in property sets [8]. It supports processors, buses, devices, and ports as well as processes, threads, and data. It is possible to define physical port-to-port connections as well as logical flows through chains of ports. Component definitions are divided into component types that define the features visible to other components and component implementations that define the inner parts of the component.

The AADL language provides the means to express the basic information required to control a multi-processor scheduling tool. The availability of multiples processing units extend the design space and engineers need help at the early stages of the design to check their choice about their assignment to the tasks. An approach to extend the AADL standard properties to support the modeling and specification of embedded multi-core system [9].

4.2.1. ACME: Acme is a simple, generic software architecture description language (ADL) that can be used as a common interchange format for architecture design tools and/or as a foundation for developing new architectural design and analysis tools. The goal of it is providing a common language that could be used to support the interchange of architectural descriptions between a variety of architectural design tools. Although it is still useful as an architectural interchange language, since the project's inception the Acme language and its supporting toolkit have grown into a solid foundation upon which new software architecture design and analysis tools can be built without the need to rebuild standard infrastructure. Currently, the Acme Language and the Acme Tool Developer's Library (Acmelib) provide a generic, extensible infrastructure for describing, representing, generating, and analyzing software architecture descriptions. It provides three fundamental capabilities: architectural interchange, Extensible foundation for new architecture design and analysis tools, architectural description. Acme is built on a core ontology of seven types of entities for architectural representation: components, connectors, systems, ports, roles, representations, and rep-maps [10].it's a small and rather simple language. A system is constituted by components connected by connectors; the ports are end-points of the connectors. Actually, ACME can be considered a subset of AADL. One peculiar thing about ACME is that its representation can vary depending on the underlying model. Component represents the primary

computational elements and data stores of a system. They correspond to the boxes in box-and-line descriptions of software architectures [10].

4.2.2. PL-Aspectual ACME: In one paper a method proposed which called PL-Aspectual ACME, a flexible and extensible ADL for modeling SPL architectures with AO abstractions. PL-Aspectual ACME seamlessly adapts concepts of ACME to represent SPL variability's at the architectural level [11].

4.2.3. LightPL-ACME: It's an ADL proposed as an ACME extension that aims to provide a lightweight, simple language for SPL architecture description, so that it is possible to associate such description with the artefacts related to the do-main engineering and application engineering activities in the SPL development process. It supports the separation of these SPL activities by creating specific abstractions for features (related to the domain engineering activity) and products (related to the application engineering activity). Moreover, LightPL-ACME was designed envisioning the representation of the architecture and its relationship with the features. Three essential elements of LightPL-**ACME** are: ProductLine, Feature, and Product. It's is able to express important elements of an SPL (such as features and the products that can be generated from them) in a simple, lightweight, clear, and objective way [12].

4.3. ArchC: It is an open-source SystemC-based language that is specialized for processor architecture description. Its main goal is to provide enough information, at the right level of abstraction, in order to al-low users to explore and verify new architectures, by automatically generating software tools like simulators and co-verification interfaces [13]. SystemC is among a group of design languages and extensions being proposed to raise the abstraction level for hardware design and verification. An architecture description in ArchC is divided in two parts: the Architecture Resources (AC ARCH) description and the Instruction Set Architecture (AC ISA) description. In the AC ARCH description, the designer provides ArchC with information about storage devices, pipeline structure, memory hierarchy and all the processor resource information available in the ISA manual. In the AC ISA description, the designer provides ArchC with details about each instruction, like: (a) format, size and assembly language syntax; (b) opcode information required to decode it; and (c) instruction behavior. Based on these two descriptions, ArchC automatically generates a simulator and an assembler for the architecture ArchC has its syntax totally based on C++ and SystemC and Designers can choose between interpreted and compiled simulators, they can also use ArchC models to simulate/evaluate more sophisticated memory hierarchies with several cache and memory levels [14]. AcSynth is an ArchC framework for power characterization and simulation. This tool brings a whole new consumption analysis aspect into ArchC allowing power reports and energy consumption to be generated in a very short time frame. It use of PowerSC, acPower and acSim tools elaborating a unified system bringing power consumption analysis into the ArchC ADL[15].

4.4. Aesope: Aesop is a system for developing style-specific architectural development environments. Each of these environments supports (1) a palette of design element types (i.e., style-specific components and connectors) corresponding to the vocabulary of the style; (2) checks that compositions of design elements satisfy the topological constraints of the style; (3) optional semantic specifications of the elements; (4) an interface that allows external tools to analyze and manipulate architectural descriptions; and (5) multiple style-specific visualizations of architectural information together with a graphical editor for manipulating them [16]. It's a set of tools designed to develop a system model and based on the UNIX environment; it has pipe and filter style extensions in order to model those futures. It has a generic kernel, suitable for all environments and a generic real-time

extension. A tool does not provide a plain text description of the model; all modelling is done in the graphic editor of the tool.

4.5. Rapide: It's a kind of architecture description language (ADL in short), provides a method for testing the consistency of component transaction. An event driven ADL, architecture defines it as a development framework that supports component-based development. The language provides modeling, analysis, simulation and code generation capabilities, but no child will be connected explicitly represented as first-order entities [1]. The type language is intended to provide interfaces for the definition language, which defines the architecture. The constraint language defines requirements for timing and other pattern events. The executable language is concurrent and reactive. Its main purpose is to construct behavior of components and connections between components.

4.6. Wright: Software architecture is receiving increasing attention as a level of software design. However, the current practice of software architecture description is largely informal and ad hoc. This has the consequence of weakening the effectiveness of architecture as a vehicle for communication about and analysis of a software system.

Wright addresses this issue by providing a formal basis for architectural description. As an architecture description language, Wright can be used to provide a precise, abstract, meaning to an architectural specification and to analyze both the architecture of individual software systems and of families of systems.

Wright defines a set of standard consistency and completeness checks that can be used to increase the designer's confidence in the design of a system. These checks are defined precisely in terms of Wright's underlying model in CSP, and can be checked using standard model checking technology. Wright is built upon the abstractions components, connectors, and configurations. The configurations can be divided into instances: attachments (describes the topology of the system) hierarchy (a component may hold other components).

4.7. Darwin: Darwin is a language for describing software structures which has been around, in various syntactic guises, since 1991. It was originally developed as a configuration language for the REX project building on experience obtained from the earlier CONIC configuration language. Darwin supports hierarchical composition. One component is composed of other components or of primitive components; that is, built-in features of the language. Darwin also supports the structure of parallel programs and modeling of network topologies.

It is a typical representative of ADLs that use implicit connections. The connections among components are specified in terms of direct bindings of requires and provides interfaces. The semantics of a connection is defined by the underlying environment (programming language, operating system, etc.), and the communicating components should be aware of it (to communicate, Darwin components directly use ports in the underlying Regis environment) [3].

4.7.1. Darwin/FSP: Darwin and FSP are two complementary efforts for describing the structure (Darwin) and behavior (FSP) of software architecture. Darwin is a structural ADL for describing a system in terms of hierarchically structured components, interfaces (which can be required or provided) and interconnections among components. FSP models behavioral aspects of a software system in terms of concurrent processes; the LTSA tool in turn generates a labeled transition system from the FSP description. An FSP specification is attached to a Darwin one by specifying the behavior

of Darwin components via FSP processes. The Darwin/FSP metamodel contains the union of all the constructs defined in the specification of both the Darwin and FSP languages [17].

4.8. MetaH: It is an ADL and toolset originally developed to meet the strict requirements of flight control and avionics, including hard real-time, safety, security, fault-tolerance and multi- processing. However, an increasingly broad range of applications shares these requirements**.**

The essential concepts of MetaH are components and connections. Each component has a set of attributes, an interface and zero or more implementations. Connections link components together to form architecture. MetaH specifications can also refer to a series of components called a path [18].

4.9. Durra: Durra is a language designed to support the development of large-grained parallel programming applications. These applications are often computation-intensive, or have real-time requirements that require efficient concurrent execution of multiple tasks, devoted to specific pieces of the application. During execution time the application tasks run on possibly separate processors, and communicate with each other by sending messages of different types across various communication links. The application developer is responsible for prescribing a way to manage all of these resources. We call this prescription a task-level application description. It describes the tasks to be executed, the possible assignments of processes to processors, the data paths between the processors, and the intermediate queues required to store the data as they move from source to destination processes. Durra is a task-level description language, a notation in which to write these application [19]. There are a number of architecture description languages have already done a lot of research on them and they are very similar to durra: Maruti [20], ABE[21], and OnikdChimera [22] in its use of an architectural specification language for composing source modules written in a traditional programming language.

4.10. SADL: A new architecture description language called SADL, intended for the expression of software architecture hierarchies that are to be analyzed formally. The entire SADL framework is formally defined and can be used to support formal reasoning about individual and multiple architectures. It's architectural elements are 1.Architectur consist of components, connector and configuration that can be contain two kinds of elements: connections and constraints. 2. Mapping.3. Architectural style 3. Tenement pattern. SADL intended for both abstract and concrete modelling of system architectures. Sadl has been used to formalize the X/Open Distributed Transaction Processing (DTP) [23].

4.11. C2: C2 is UCI's component- and message-based architectural style for constructing flexible and extensible software systems. A C2 architecture is a hierarchical network of concurrent components linked together by connectors (or message routing devices) in accordance with a set of style rules. C2 communication rules require that all communication between C2 components be achieved via message passing. The C2 style focuses on the conceptual architecture of a system, independently of particular implementation architecture. UCI's Java and C++ class frameworks for C2 concepts, such as components, connectors, and messages, provide development support for implementing C2 architectures in Java and C++. Components, connectors, and messages are explicitly represented as objects.

4.11.1. C2 SADL: It is the language for defining architectures built according to the C2 style. C2 SADL draws its influences from the strengths and shortcomings of existing ADLs. It is currently only a prototype language and its needed support tools are under construction. C2 SADL consists

of three parts: IDN (interface definition notation), ADN (architecture description notation), ACN (architecture construction notation) [24]. a simplified version of C2 SADL, an ADL (with its accompanying ACN) for architectures built according to the C2 style, Components and connectors in this architecture can be "rewired" simply by using the Weld and Unweld functions of the CAN. This architecture improved The problem of dynamically changing architecture, Currently existing ADLs only provide means for declaratively specifying the structure of an architecture. As such, they are not well suited to support dynamic architectural changes. One possible solution is to provide an architecture construction facility (ACN) in an ADL. Coupled with ACN interpretation and appropriate code generation tools [25]. This kind of ADLs solves the problem dynamically changing architecture. We should contemplate Dynamic Architecture is trying to establish under what circumstances it is safe to remove and/or add a component to an architecture, change the filtering policy on a connector port, and "rewire" the architecture the role of a particular architectural style in facilitating dynamic architecture changes. On the one hand, the ability to express such changes in an ACN is independent of a style [25].

4.12. Cheddar: is a free real time scheduling tool composed of a graphical editor used to describe a real-time applications, a framework which includes most of classical real time scheduling/feasibility algorithms/tests. It is designed for checking temporal constraints of real-time applications. To perform this type of scheduling analysis with Cheddar, systems to analyse can be described with AADL or with a dedicated ADL, the Cheddar Architecture Design Language, called Cheddar ADL. Cheddar ADL aims to write, analyse and validate real-time applications handled in the context of Cheddar. The particularity of this ADL, as compared to other ADLs, is that it allows to capture all required aspects for the schedulability analysis of real-time systems. The particularity of this ADL, as compared to other ADLs, is that it allows to capture all required aspects for the schedulability analysis of real-time systems. It's elements classified into two categories software part, which contain Address space , Task, Buffer, Resource, Message and Dependency , and hardware part, which contain C or e , Cache , Processor and Network. In compare with other ALs cheddar provides a framework which is used to classify and compare several existing ADLs [26].

4.13. LISA: The language LISA is aiming at the formalized description of Programmable architectures, their peripherals and interfaces. It was developed to close the gap between purely structural oriented languages (VHDL, Verilog) and instruction set languages for architecture exploration purposes. The language syntax provides a high flexibility to describe the instruction set of various processors, such as SIMD, MIMD and VLIW-type architectures. Its model components is consisting of memory model, resource model, the instructions model, behavioral model, timing model and micro-architecture model [27].

4.13.1. In a case study it was shown that a real-world ASIP, the ICORE architecture, was completely realized using the LISA based HDL code generation. The results concerning maximum frequency, die size and power consumption were comparable to those of the hand optimized version of the same architecture [28].

4.13.2. LISPARC: It's a processor model that was described using an architecture description language called Language for Instruction-Set Architectures (LISA). It is advantageous to use a high level model instead of a hardware processor model [29].

4.13.3. In a paper has been presented an ASIC (Application-Specific Integrated Circuits) synthesis flow based on Architectural Description Language LISA. Based on such ASIC model in LISA, alternative design choices can be quickly explored by applying techniques such as state merging and splitting, resource sharing and multi-mode synthesis [30].

4.14. LImbiC: It's designed using a 32-bit Harvard architecture. One read-only memory is used to store the program and one read-write memory is dedicated to data. The data memory is optimized for image processing applications by supporting two-byte accesses to address individual pixels. LImbiC has 13 32-bit general-purpose registers, as well as a 32-bit stack pointer, link register and program counter. The program status register was scaled down to an application-specific status register just representing the ALU flags and the processor mode. No additional registers are added to maintain machine-code compatibility to the Thumb ISA. The LImbiC processor leverages a 3-stage pipeline comprised of fetch decode and execute stages. A LISA model in Processor Designer can be used to produce three different types of outputs: software development tools, simulation models, and synthesizable HDL models. The ADL LISA is well suited to develop an application-specific processor with a small number of instructions [31].

4.15. XML-BASED Architecture Description Languages:

4.15.1. DAPNA: A method have been introduced for documenting, analyzing and realizing DPNs. DPN architecture is specified with an XML-based architecture description language (ADL) and DAPNA comprises a reusable library of communication primitives, data source and sink definitions, and data filters. This library can be further extended with user defined, composable data filter. It makes the development of DPNs less effort-consuming and less error-prone [32].

4.15.2. ADML: The Architecture Description Mark up Language (ADML) is an XML-based representation language for architecture. ADML is based on ACME, an architecture description language. ADML adds to ACME a standardized representation (parsable by ordinary XML parsers), the ability to define links to objects outside the architecture (such as rationale, designs, components, etc.), straightforward ability to interface with commercial repositories, and transparent extensibility. There are other languages that have been proposed previously: ABC/ADL [33], XSSA/ADL [34]

4.15.3. xARCH: is an XML-based representation for building ADLs. It consists of a core of basic architectural elements, defined in an XML schema called the "instances" schema. The xArch instances schema provides definitions for the following elements typically found in an ADL. Component, connector, interface, and link (connection between interfaces). xArch can be extended by writing new XML schemas that augment the core xArch schema with additional information about the architecture by modifying existing tags and attributes or adding new ones. This property allows architecture researchers to add their own modelling constructs to xArch. In addition to the many XML [35].

4.15.4. Breeze/ADL: It uses the Breeze Graph Grammar formalism (BGG) that proposed by Huang as a theoretical basis [36] and adopts XML as meta-level language which enhances its inter-operability with other XML-based ADLs, which represents software architecture in XML format according to BGG. The basic elements of it are node (component, connector), port and edge which match the BGG definitions. Breeze/ADL not only translates the software architecture graph into a textual format but also captures change during both initial development and subsequent evolution. The main benefit is that it uses the graph grammar to support Breeze/ADL and can provide the following benefits:

- Inter-operability: It uses the XML as its textual form, thus, general XML tools may easily parse or modify the Breeze/ADL file.

- Preciseness: The breeze graph grammar formalism provides a precise definition of architecture elements and operations, and the Breeze/ADL supports refinement and evolution of the system [37].

4.15.5. xADL: It provides five XML (Extensible Markup Language) based tags to represent architecture elements, namely Architecture, Component, Connector, ComponentType and ConnectorType. xADL contains the inherent features of XML, which allow to extend tags for expressing pattern elements. Each tag can be enforced with pattern elements specific constraints**.** xADL supports type of connections using XML DTDs (Document Type Definitions), which means different kinds of connections to express pattern elements can be used by specifying DTDs. Furthermore, these DTDs can be used to constrain the behavior of interacting pattern elements [38]. xADL forms the basis for the ArchStudio 4 [39].

4.15.6. XYZ/ADL: It is a software architecture description language which Based on the executable temporal logical language XYZ/E (is an executable temporal logic language based on Manna-Pnueli's Linear Time Temporal Logic, which combines both static and dynamic semantics in a unified framework and supports the whole procedure of stepwise refinement, from the abstract specifications to executable codes.). XYZ/ADL can not only represent the system description at different abstract levels from formal specification to executable program which under a unified logical framework, but also can represent both dynamic semantics and static semantics of software architecture. XYZ/ADL, suitable for the formal description and helpful for the refinement to the software architecture at different abstract level, can verify the semantic consistency of the process of refinement with tools in the XYZ system. The following are architectural units of XYZ/ADL [40].

4.15.6.1. In one paper have been used the concept of AOP at coding phase up to software architecture by adding Aspect into XYZ/ADL and adding aspect role in connector for dealing with the interaction between aspect and component, and proposes the related composition mechanism so as to form the Aspect-Oriented Architecture Description Language (AO-ADL) [41].

4..15.6.2. BYADL (Build Your ADL) is a framework that supports a software architect in defining its own ADL, which is optimal according to specific stakeholder's concerns, starting from an existent ADL. BYADL provides extensibility mechanisms for: (i) adding domain specificities, new architectural views, or analysis aspects, (ii) integrate ADLs with development processes and methodologies, (iii) customize ADLs by fine tuning them. It supports the generation of textual and graphical editors for the newly created ADL [42].

4.16. FPGA: logic block architecture description language (FPGA) permits the modelling of far more complex soft logic blocks and hard logic blocks than was previously possible. The key features of the language are its ability to describe hierarchy, modes and arbitrary interconnect between atomic elements in the block. It presented a packing algorithm that begins to address the complexities of the FPGAs that use the new language. Complex logic blocks require a new CAD flow that permits the expression of that complexity and the ability to synthesize to it. The new language uses XML syntax; readers unfamiliar with XML At the highest-level, the language contains two categories of construct:

1) physical blocks, and 2) interconnect. The basic physical block type in the language is specified using the XML element physical blocks type, so can we say FPGA is a XML-based ADL [43].

4.17. UML: Unified Modelling Language (UML) is an ADL and has become a de facto standard notation for documenting the architecture of software systems. The UML has well defined formal syntax and semantics and can be machine checked and processed. UML includes a set of graphical notation techniques to create abstract models of specific systems. It is capable to describe and model even software architectures too [44]. We can say the UML is a graphical language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system. The UML supports multiple views of a system both structural and behavioral especially those included in Kruchten's 4+1 model of view of architecture.

Strengths

- UML provides a graphical representation to the software architecture.
- UML supports multiple views which are very helpful for all the stakeholders.
- Numerous tools are available for UML.
- UML is a general purpose modeling language and have been used effectively in almost all the domains of software engineering.

Weaknesses

- Not suitable for automated analysis of verification and validation etc. of architecture.
- UML constructs lack in formal semantics and therefore may become a source of ambiguity, inconsistency in some cases [7].

4.17.1. UML/MARTE: UML/Modelling and Analysis of Real-Time and Embedded System (MARTE) profile currently supports mono and multiprocessor scheduling algorithms, but only for a partitioned approach has proposed various updates for MARTE metamodels of specialization and generalization stereotype in order to support global scheduling approaches, allowing task migrations. Those changes allow a schedulable resource to be executed on different computing resources in the same period [9].

4.17.2. Grasp: It is a textual ADL capable of specifying rationales and associating them with elements of architecture. It implements the conceptual model. In one paper Grasp ADL attempts to address the adoption of rationale techniques as part of architecture. The Architecture Rationale Element Linkage (AREL) model attempts to capture architecture rationale with traceability. AREL promotes architecture rationale to a first-class entity and establishes relationships between rationale and elements in the architecture. The AREL approach uses UML profiles to model rationale, AEs and their associations. These UML elements in the AREL example were manually translated into their equivalent Grasp constructs [45].

4.17.3. SysADL: a SysML (is a customized version of UML for systems engineering, and it is being increasingly used by systems engineers and inheriting the popularity of UML.) profile for

expressing architecture descriptions using the well-known and consolidated abstractions from the ADL community. A profile is a lightweight extension of a Language that allows specializing its syntax using stereotypes that represents both a well-defined syntactic element and a set of additional semantic constraints for each stereotyped metaclass and it uses SysML as it is an OMG standard that focuses on systems and software.

Components; A SysADL component is a stereotype that specializes a block with a subset of its properties.

Connectors; A Connector in SysADL is a stereotype that specializes a SysADL component to represent the interactions between components.

Configurations; A configuration defines the structure of a system as a composition of components and connectors.

Ports; A Port is an interface that specifies services in both components and connectors [46].

4.17.4. EAST-ADL: It provides a comprehensive approach for describing automotive electronic systems through an information model that captures engineering information in a standardized form. Aspects covered include vehicle features, requirements, analysis functions, software and hardware components, communication and their dependencies are refinement, allocation, composition, communication, etc. The representation of the system's implementation is not defined in EAST-ADL itself but by AUTOSAR. EAST-ADL2 and AUTOSAR in concert provide means for efficient development and management of the complexity of automotive embedded systems from early analysis right down to implementation. Concepts from model based development and component based development reinforce one another [47].

The main role of EAST-ADL2 is that of providing an integrated system model. As such, EAST-ADL2 must address multiple aspects of a system including:

- Documentation, in terms of an integrated system model.
- Communication, by providing predefined views as well as the information sufficient for generating a number of other views.
- Analysis of a complete embedded system through the description of system structure and properties. Special emphasis has been placed on modeling support for analysis of component interfaces, timing correctness and safety analysis [48].
- A method has been proposed that describe how the model-based design and analysis capabilities of the EAST-ADL language can be combine with metaheuristic algorithms such as genetic algorithms to allow automatic architectural optimization to take place [49].

4.17.4.1. EAST-ADL2/MARTE: There is an approach to combine MARTE and EAST-ADL2 to overcome EAST-ADL2 limitation of notions for modeling the timing features. EAST-ADL2 is an architecture description language defined as a domain specific language

for the development of automotive electronic systems. The MAST toolset is integrated in the MDE process to perform the scheduling analysis [9].

4.17.4.2. VITAL: a method for integrating architectural description languages and verification techniques, tailored for EAST-ADL models and implemented in the tool ViTAL (A Verification Tool for EAST-ADL Models using UPPAAL PORT). ViTAL provides model-checking of EAST-ADL descriptions with respect to timing and functional behavioral requirements [50].

4.18. LePUS3 and Class-Z are formal object-oriented Design Description Languages. They are formal specification languages for modelling non-functional specifications representing the design of object-oriented class libraries, design patterns, and object-oriented application frameworks. LePUS3 diagrams 'Codecharts' or simply charts are axiomatiezed as decidable statements in the first-order predicate logic. LePUS3 is tailored for tool support in fully automated design verification (checking conformance to Codecharts) and program visualization (design recovery from source code), as demonstrated by the Two-Tier programming toolkit.LePUS3 and Class-Z are formal Design Description Languages tailored for the following purposes:

- **Scalability**: To model industrial-scale programs using small Codecharts with only few symbols
- **Automated design verifiability**: To allow programmers to continuously keep the design in synch with the implementation
- **Visualization** (only LePUS3): To allow tools to reverse-engineer legible Codecharts from plain source code modelling their design
- **Pattern verification**: To allow tools to determine automatically whether your program implements a design pattern
- **Abstraction in early design**: To specify unimplemented programs without committing prematurely to implementation minutia
- **Genericity**: To model a design pattern not as a specific implementation but as a design motif

- **Rigour**: To be rigorous and allow software designers to be sure exactly what Codecharts mean and reason rigorously about them.

4.19. SMADL: The Social Machines Architecture Description Language – as an attempt to be a different way to program the Web, mixing concepts from Architecture Description Languages (ADLs) and Domain-Specific Languages (DSLs). SMADL presents a simple textual syntax in favor of expressiveness. It was developed in Xtext language Workbench, making it fully integrated to the Java Virtual Machine and Eclipse IDE. In SMADL, a relationship is represented by a single keyword, so composition possibilities for several SMs can be infinite, making it possible to create a network of SMs. Each SM establishes a relationship with others, just like an import mechanism in Java [51].

4.20. ABACUS: in order to reduce the risks associated with complex systems is through an architecture-based approach. The central function of Architecture-Based analysis of Complex Systems (ABACUS) is to allow enterprise architects, strategists and designers to model, predict and control the emergent properties of their respective systems from an architectural point of view. It's elements are components, connectors and properties [52]. In a paper by ArchiMate language have been reviewed the challenges confronting an architecture description language for enterprise architecture [53].

4.21. EADL: Embedded architecture description language (EADL) captures both hardware and software components and their interactions, and gains its flexibility from its support to platform-oriented instantiation. It has demonstrated its effectiveness in serving as the vehicle for integrating component-based co-design, co-simulation, co-verification, and co-synthesis in ESIDE. It has key features include:

- EADL is based on a unified component model for embedded systems that unifies hardware and software components and bridges the HW/SW semantic gap.
- EADL does not dictate execution and interface semantics of hardware and software components while supporting platform-oriented semantics instantiation.
- EADL supports concise specification of embedded system architectures and also formulation of architectural patterns of embedded systems.
- EADL integrates architectural design with assertion-based verification (ABV). It supports association of properties with components and property templates with architectural patterns, to facilitate HW/SW co-verification using formal methods such as model checking [54].

4.22. LILEANNA: LILEANNA (LIL Extended with Anna (Annotated Ada) is an implementation of Goguen's LIL (Library Interconnect Language). LILEANNA is a language for formally specifying and generating Ado packages. LILEANNA extends Ada by introducing two entities: theories and views, and enhancing a third, package specifications. A LILEANNA package, with semantics specified either formally or informally, represents a template for actual Ada package specifications. It is used as the common parent for families of implementations and for version control. A theory is a higher-level abstraction, a concept (or content), that describes a module's syntactical and semantic interface. A view is a mapping between types, options, and exceptions [55]. In a paper was presented the ALMA (Architecture oriented paraLlelization for high performance embedded Multicore systems using Scilab) toolset that aims to deliver an end-to-end solution for semi-automatic parallelization of Scilab code to embedded multicore architectures. Two distinct phases are identified, the parallel code production and the parallel platform code generation. Two important tools integrate all the parts of the toolset, (1) the ALMA Intermediate Representation (ALMA IR) representing the Scilab code during the parallelization steps and (2) the Architecture Description Language (ADL) enabling the architecture independence of the toolset by providing an abstract specification of the target architectures. The ADL makes use of a hierarchical architecture description, a specialized ADL compiler and a modular SystemC based simulation environment. Simulation flexibility is gained through the ability of extracting multiple abstraction levels from the ADL. The evaluation of the simulation environment has shown, that the hierarchical approach provides flexibility in simulation accuracy and simulation performance [56].

3. CONCLUSIONS

In this paper we have classified ADLs and tried to describe properties each of them. In several papers have been evaluated number of ADLs and have been used of features them. For example in a paper by review ADLs have perused what industry needs from Architectural Languages [57] or have been perused role of ADLs in Dynamic Architectural Changes [25]. In a paper have been evaluated the most popular or commonly used ADLs, with respect to four of the most significant architecture patterns (Syntax, Visualization, Variability, Extensibility) and modelling patterns for software architecture [38]. In a paper has been reviewed number of ADLs with features, next reviewed analysis and evaluation methods: SAAM(Scenario-Based Architecture Analysis Method), ATAM (Architecture Trade off Analysis Method), ALPSM(Architecture Level Prediction of Software Maintenance),

ALMA(Architecture Level Modifiability Analysis), SACAM(Software Architecture Comparison Analysis Method) [1]. For specifying software architecture and validate quality requirements, In an approach have been used of two different research methodologies: a qualitative investigation followed by a quantitative validation survey which triangulates the results of the first, It has been proposed some ideas for specifying quality requirements and for reducing the cost of validation [58]. We classified ADLs and new version of them in separate and small categories. In next step we will review an evaluation of new ADLs and their versions.

REFERENCES

1. Lingling Y, Wei Z, "An overview of software architecture description language and evaluation method. International Conference on Communication, Electronics and Automation Engineering Advances in Intelligent Systems and Computing. vol. 181, pp 895-901, 2013
2. Woods E, Bashroush R. Modeling large-scale information systems using ADLs – An industrial experience report. in journal of systems and software. vol. 99, pages 97–108, 2015
3. Balek, D. Connectors in software architectures. Department of distributed and dependable, 2002
4. Medvidovic N, Taylor R.N. A classification and comparison framework for software architecture description languages. Software Engineering, IEEE Transactions .vol. 26, pp. 70-93. 2000
5. Clements P.C. A survey of architecture description languages. IEEE computer society Washington, DC, USA, pp.16, 1996.
6. Gutmann M. Software review: ADL / X_trader. http://www.futuresmag.com, 2012
7. Pandey R.K. Architectural Description Languages (ADLs) VS UML: A Review. ACM Sigsoft software engineering note, vol. 35 issue 3, pp. 1-5, 2010
8. Bae K, Olveczky P.C, Meseguer J. Definition, semantics, and analysis of multirate synchronous AADL. 19th international symposium, Singapore, pp. 94-109, 2014
9. Rubini S. Scheduling analysis from architectural models of embedded multi-processor systems. ACM sigbed review-special issue on the 3rd embedded operating system workshop. vol. 11 issue 1, pp. 68-73, 2014
10. Garlan D, Monroe R.Wile D. ACME: An architecture description interchange language. Cascon first decade high impact papers, pp. 159-173, 2010
11. Barbosa E.A, Batista T, Garsia A & Silva E. Pl-Aspectual ACME: An aspect-oriented architectural description language for software product lines. 5th European conference, ECSA, pp.139-146, 2011
12. Silva E, Medeiros A.L, Cavalcante E & Batista A. A lightweight language for software product lines architecture description. 7th European conference, ECSA, pp. 114-121, 2013
13. Rigo S, Araujo G, Bartholomeu M & Azevedo R. Archc: a system c-based architecture description language. 16th symposium computer architecture and high performance computing, 2004
14. Azevedo R, Rigo S, Bartholomeu M, Araujo G, Araujo C, Barros E, The ArchC architecture description language and tools, on international journal of parallel programming. vol. 33, pp. 453-484, 2005
15. Guedes M, Auler R, Borin E, Azevedo R. An automatic energy consumption characterization of processors using ArchC. on journal of systems architecture. vol. 59, issue 8, 2013
16. Garlan D, Allen R, Ockerbloom R. Exploiting style in architectural design environments. on ACM sigsoft software engineering notes.vol. 19, pp. 175-188, 1994
17. Eramo R,I. Malavolta I, Muccini H, Pelliccione P, Pierantonio A. A model-driven approach to automate the propagation of changes among architecture description languages on software & systems modeling. vol. 11, issue 1 , pp. 29-53, 2012
18. Krueger J.K, Vestal S, Lewis B. Fitting the pieces together: system/software analysis and code integration using METAH. digital avionics systems conference. vol. 1, 1998
19. Barbacci M.R, Wing J.M. DURRA: A task-level description language reference manual. software for heterogeneous machines project, 1989
20. Saksena, J, Silva A & Agrawala, A. Design and implementation of Maruti-ii. principles of real-time, 1994

21. Hayes-Roth F, E. Davidson J.E, Erman L.D, Lark J.S. Frameworks for developing intelligent systems: the ABE systems engineering environment. IEEE expert, vol. 6 , 1991
22. Gertz M.W, Stewart D.B, Khosla P.K. A human machine interface for distributed virtual laboratories. on robotics & automation magazine, IEEE, vol. 1, pp. 5-13, 1994
23. Moriconi M, Riemenschneider R.A. Introduction to SADL 1.0: a language for specifying software architecture hierarchies. 1997
24. Taylor R.N, Redmiles D.F. The c2 software architecture description language (SADL). information and computer science university of California
25. Medvidovic N. ADLs and dynamic architecture changes. in ISAW '96 joint proceedings of the second international software architecture workshop. on sigsoft '96 workshops, pp. 24-27, 1996
26. Fotsing C, Singhoff F, Plantec A, Gaudel V, Rubini S, Li, H.S , Lemarchand T. Cheddar architecture description language. http://www.univ-brest.fr, 2014
27. Schliebusch O, Hoffmann A, Nohl A, Braun G, Meyr H. Architecture implementation using the machine description language LISA. design automation conference, pp. 239-244, 2002
28. Tradowsky C, Thoma F, Hubner M, Becker J. Lisparc: Using an architecture description language approach for modeling an adaptive processor microarchitecture,.industrial embedded systems (SIES), pp. 279-282, 2012
29. Wang Z, Wang X, Chattopadhyay A, Rakosi Z.E. ASIC synthesis using architecture description language. VLSI design, automation, pp. 1-4, 2012
30. Tradowsky C, Harbaum T, Deyerle S, Becker J. LIMBIC: An adaptable architecture description language model for developing an application-specific image processor. in IEEE computer society annual symposium, pp. 34-39, 2013
31. Sözer H, Nouta S, Wombacher A, Perona P. DAPNA: An architectural framework for data processing networks. information sciences and systems, pp. 221-229, 2013
32. Guang W.X, Dong F.Y & Hong M. ABC/ADL: An XML-based software architecture description language.in journal of computer research and development, 2004.
33. Liang T, Mingtian Z. XSSA/ADL: An XML-based security requirement architecture description language. in journal of computer research and development, 2007
34. Dashofy E.M, Vanderhoek A, Taylor R.N. A highly-extensible, xml-based architecture description language. in proceedings working IEEE/IFIP conference, pp. 103-112, 2001
35. Li C, Chen L & Yu C. BGG: A graph grammar approach for software architecture verification and reconfiguration. in International Conference on Complex, Intelligent, and Software Intensive System, 2013
36. Li C, Huang L, Chen L, Yu C. Breeze/ADL: graph grammar support for an xml-based software architecture description language. in IEEE 37th annual computer software and applications conference, 2013
37. Kamal A.W, Avjeriou P. An evaluation of ADLs on modeling patterns for software architecture. on workshop on rapid integration of software, 2007
38. Archstudio 4.0 tool set for the XADL language. on http://www.isr.uci.edu/projects/archstudio
39. Zhang G, Shi H, Rong M, Di H. Model checking for asynchronous web service composition and l.n.i.c.s. Based on XYZ/ADL. vol. 6988, pp. 428-435, 2011
40. Cao Y, Rong M & Zhang G. An aspect-oriented software architecture description language AO-ADL based on XYZ. on international journal of hybrid information technology vol. 7, pp. 365-376, 2014
41. Di Ruscio D, Malavolta I, Muccini H, Pelliccione P, Pierantonio A. BYADL: An mode framework for building extensible architecture description languages," lecture notes in computer science. vol. 6285, pp. 527-531, 2012
42. Luu J, Anderson J.H, Rose J.S. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. in 19th ACM/sigda international symposium on field programmable gate arrays, pp. 227-236, 2011
43. Bharathi B, Sridharan D. UML as an architecture description language. in international journal of recent trends in engineering. vol. 1, no. 2, 2009.

44. Silva L, Balasubramaniam D. A model for specifying rationale using an architecture description language. lecture notes in computer science. vol. 6903, pp. 319-327, 2011
45. Leite J, Oquendo F, Batista T. SysADL: A sysML profile for software architecture description. lecture notes in computer science. vol. 7957, pp. 106-113, 2013
46. Cuenot P, Frey P, Johansson R, Lönn H, Papadopoulos Y, Reiser M.O, A. Sandberg, D. Servat, R. T. Kolagari, M. Törngren, M. Weber. 11 the east-ADL architecture description language for automotive embedded software. lecture notes in computer science. vol. 6100, pp. 297-307, 2010
47. Törngren M, Chen D, Malvius D, Axelsson D. Model-based development of automotive embedded systems. OAI: diva.org:kth-80302, 2008
48. Walker M, Reiser M.O, Piergiovanni S. Papadopoulos T, Lönn H, Mraidha C, Parker D, Chen D, Servat D. Automatic optimization of system architectures using east-ADL," journal of systems and software. vol. 86, issue 10,pp. 2467–2487, 2013
49. Enoiu E.P, Marinescu R, Seceleanu C, Pettersson P. VITAL: a verification tool for east-ADL models using uppaal port. engineering of complex computer systems, pp. 328 – 337, 2012
50. Do Nascimento L.M, Burégio V.A.A, Garcia V.C, Meira S.R.L. A new architecture description language for social machines. International world wide web conferences steering, pp. 873-874, 2014
51. Dunsire K, O'neill T, Denford M, Leaney J. The abacus architectural approach to computer-based system and enterprise evolution. engineering of computer-based systems, pp. 62 – 69, 2005
52. Lankhorst M.M, Proper H.A, Jonkers H. The architecture of the architect language. enterprise, business-process and information systems modeling, vol. 29, pp. 367-380, 2009
53. Pilkington J.L.N.T, Xie F, Liu Q. Embedded architecture description language. journal of systems and software. vol. 83, issue 2, pp. 235–252, 2010
54. Tracz W, "LILEANNA: a parameterized programming language. in proceedings advances in software reuse. selected papers from the second international workshop, pp. 66 – 78, 1993
55. Bruckschloegl T, Oey O, Ruckauer M, Stripf T, Becker J. A hierarchical architecture description for flexible multicore system simulation. in parallel and distributed processing with applications (ISPA), IEEE international symposium, pp. 190 – 196, 2014
56. Bruckschloegl T, Oey O, Ruckauer M, Stripf T, Becker J. A hierarchical architecture description for flexible multicore system simulation. in parallel and distributed processing with applications (ISPA), IEEE international symposium, pp. 190 – 196, 2014
57. Malavolta I, Lago P, Muccini H, Pelliccione P & Tang A. What industry needs from architectural languages: a survey.on software engineering, IEEE transactions, vol. 39, pp. 869 – 891, 2013
58. Caracciolo M, Lungu F & Nierstrasz O. How do software architects specify and validate quality requirements?. on software architecture lecture notes in computer science volume 8627, pp 374-389, 2014