Critique_3

By Anvesh Tummala (16173144)

Critique on “Software Reflexion Models: Bridging the Gap between Design and Implementation”:

Software reflexion model is used to map engineer’s high-level model and source model by summarizing a source model from the viewpoint of engineer’s high-level model. (The work we are doing our project is just building the source model without considering any existing engineers model, so finally our architecture model might conflict with the engineer’s model.). But this model is a two way approach where it tries to build a model using bottom-up approach, such a way that it will converge with top-level model or at least it will find where it is diverging or absence from the high-level model. This reflexion model is unambiguous and precise as it is more formalized using Z specification language, it is very important for conformity of the final map. As it is an iterative and lightweight(less effort needed) model, it can be easily usable by developers. This model helps in viewing the system specific to the task and also reduces the worry about keeping documentation (can get on-demand documentation using this).

Critique on “A Classification and Comparison Framework for Software Architecture Description Languages”:

This paper classifies some existing Architecture Description Languages and highlights some common important aspects between them. My assessments are as follows

- While contradicting Clements research, “Modechart can’t be just considered as an ADL because of its tool support” is not convincing. The author might have given some properties missing in Modechart for not to be an ADL (like Shaw and Garlan’s properties for ADL, composition, abstraction, configuration, heterogeneity and analysis). But contradicting the use of criteria “ability to model requirements and algorithms” in ADL’s is convincing as he stated in the beginning of paper that ADL’s scope is mainly on structural not implementation level.
- In all ADL components and connectors studied Aesop is the only Language that support all classification categories. It implies that satisfying all the classification categories is not complementary. Also In future we should focus on currently sparsely provided categories like evolution and nonfunctional properties to increase level of support for developers (like extensibility).
- Surprisingly, some ADL’s doesn’t consider Connectors as the basic elements and provide them as inline. These inline connectors cannot be used at design level and cannot be reused. This emphasis that an ADL to be complete, should have connector as a basic element.
- Most interesting point is System generation (with each code module linked to each architectural element) from architectural specification may not always be a positive sign. They cause problems in the refinement of architecture. The future changes to the code modules may not be possible to reflect on the architecture or vice versa. From this we can conclude that the architecture should be at more abstract level than the code modules, this allows the flexibility for the refinements in both architectural and code.
- From the comparison tables from the paper, I map some factors to the following table. (Scope-focus; Basic Elements- Components, Connectors, Interfaces; Static and Dynamic Aspects-Dynamism in configuration, tools; Ambiguity-Understandable specification; Accuracy and Precision- Semantics, Constraints, Nonfunctional Properties; Viewpoint-Multiple Viewpoints)

	ACME	Aesop	C2	Darwin	MetaH	Rapide	SADL	UniCon	Weaves	Wright
Scope and Purpose	Architectural interchange mainly at structural level	Specification of architectures in specific styles	Architectures of highly-distrusted, evolvable and dynamic systems	Structure of distributed systems that communicates with well-defined interfaces	Architectures in the guidance, navigation, and control (GN&C) domain	Modeling and simulation of the dynamic behavior described by an architecture	Formal refinement of architectures across levels of details	Glue code generation for interconnecting existing components using common interaction protocols	Data-flow architectures , characterized by high-volume of data and real-time requirements on its processing	Modeling and analysis of the dynamic behavior of concurrent systems
Basic Elements	Components, Ports, Connector	Components, Ports (in & out), Connector	Components, Ports (provided and required), Connector	Components, services (provided and required), binding, hierarchical composition	Processes, Ports, Connection	Interfaces, Constituents(Provided, Required, action & service) , Connection	Components, input and output ports, Connector	Components, Players, Connector	Tool Fragments, Objects, Transport Services	Components, Ports (semantics specified in CSP), Connector
Static & Dynamic Aspects	Static structural views	Static structural views	Static and Dynamic structural views and Dynamic architectures (unanticipated)	Static structural views, Dynamic architectures through lazy and dynamic instantiation and binding	Static structural views	Static structural views, Dynamic architectures through dynamic event generation	Static structural views	Static structural views	Static and Dynamic structural views and Dynamic architectures (unanticipated)	Static structural views and Dynamic architectures(constrained)
Ambiguity	Less ambiguity using explicit textual specification	Less ambiguity using explicit textual specification and parallel type hierarchy for visualization	Less ambiguity using explicit textual and graphical specification	The external meaning of elements is ambiguous (component, interface)	The external meaning of elements is ambiguous (component, interface)	The external meaning of elements is ambiguous (component, interface)	Less ambiguity using explicit textual specification	Less ambiguity using explicit textual and graphical specification	Less ambiguity using explicit textual specification	Less ambiguity using explicit textual specification
Accuracy	There are no semantics, uses other ADL's semantic models in property lists. Below Avg	Semantics are optional using style specific languages. Average or worst (optional)	Semantics defined for component invariants, operation pre –post-conditions in 1st order logic Above Avg	Formalism by Pi-Calculus, helps in checking internal consistency. Best	Uses ControlH for modeling algorithms in GN&C domain. Implements semantics via paths. Average	Formalized by Posets (Partial Ordered Sets) Best	Not formalized. Worst	Not formalized, but have event traces in property list. Below Avg	Uses Partial Ordering over input and Output. Above Avg	No semantics. Semantics specified in CSP. Below Avg
Precision	Less	Less	High	Limited to structural elements and interconnection	High	High	Worst	Less	High	Less
Viewpoints	Textual, weblets, architectural view (high-level, basic constructs)	Textual, Graphical, style-specific visualization, component and connectors distinguished iconically.	Textual, graphical view of development process	Structural, development viewpoints using hierarchical composition	Textual, graphical, component types distinguished iconically	Textual, graphical, Execution behavior by animating simulations.	Textual only.	Textual, graphical, component and connectors distinguished iconically.	Graphical only, Component and connector types distinguished iconically.	Textual only, model checker provides a textual equivalent of CSP symbols.

Critique on "The Architecture Analysis & Design Language (AADL): An Introduction":

This ADL differs from others in few aspects, it represents both architectural and runtime environments of system, it provides elements for data flows and interconnections like bus. It provides elements at Software, Hardware and Composite level, which helps in representing complete abstract representation of system in detail. It even has scope to represent non-functional properties like synchronization, timing requirements, deadlines, space requirements, arrival rates, constraints etc.

Constraints limit binding between process and processors and collocation of components that support fault tolerance. Extensibility can be possible by property sets and Annex libraries and helps developers to alter the specifications to meet domain specific requirements. The use of component types and components is from OOPs concepts. We can think types as he abstract classes, features and flows as interfaces, properties as parameters, extends as inheritance, refines type (adds new features in runtime) as overloading (Polymorphism), Packages as packages, property sets templates. Viewpoints include textual, Graphical, XML representations. Less ambiguity because of its well defined features, specifications. Moderately Accurate as it does not use any formalized model under it, but it has well defined semantics for AADL declarations. In conclusion I can say AADL is an object oriented language in ADL's that helps in capturing both static and dynamic behaviors of system.