

Trinity: A Language for Multi-View Architecture Description and Control

Abstract

This is the text of the abstract.

1. Multiple Views of Software Architecture

The software architecture of a system is the set of structures needed to reason about the system, each consisting of elements of the system, relations among them, and properties of both [2]. Software architecture enables the system designers to analyze the system's ability to meet quality objectives like performance, reliability, etc. Thus, it facilitates the early detection of design errors and leads to improved software quality. It also provides a blueprint for system implementation and evolution.

To facilitate the different uses of software architecture in system development activities, a number of architecture description languages (ADLs) have been developed [1]. ADLs are formal languages that can be used to represent the architecture of a software system in an unambiguous way. Thus, they provide a rigorous basis for the analysis of system designs. ADLs also enable the various stakeholders to understand the system better and thus aid in system implementation, maintenance and evolution.

System designers often organize the architecture description of a software system as a set of views. An architectural view is a representation of those elements of the system that are needed to show how the architecture addresses a concern held by one or more stakeholders. For example, to devise an implementation plan, the system designers might create a (module) view that shows the module decomposition of the system and the dependencies among the modules (i.e., implementation units). Moreover, this view would not include the system's runtime entities or elements from the system's deployment environment. The separation of concerns achieved by splitting the architectural description into differ-

ent views helps keep the architecture description cognitively tractable.

The vast majority of existing ADLs, however, lack support for multiple views. This hampers the use of ADLs for the specification of software architecture in practical settings. A survey conducted by Malavolta et al. [6] found that better support for multiple views is one of the features most desired by industrial practitioners in ADLs.

2. Architectural Control

Software architects design systems to meet quality attribute requirements like performance, reliability, security, etc. To guarantee that the implemented system exhibits the desired quality attributes, it is necessary to ensure that the implementation adheres to the design principles and constraints prescribed by the architecture. Architectural control is the ability of software architects to enforce architectural constraints on the implementation so that the system meets its design goals.

Luckham and Vera have identified *communication integrity* as a key aspect of maintaining consistency between the architecture and implementation of a software system [5]. Communication integrity is the property that each component in the implementation may communicate directly only with the components to which it is connected in the architecture.

In prior work, ArchJava used a custom type system to verify communication integrity statically [3]. A limitation of ArchJava is its inability to ensure that an application communicates over the network using only the connections shown in the architecture. Since programs in ArchJava have unrestricted access to system libraries, a component can use the network library directly to communicate in ways not specified in the architecture.

3. Trinity

We are developing Trinity, an ADL that would allow software architects to ...

Multi-View Architecture Description in Trinity. Clements et al. [4] identify three basic types of views: ...

Trinity provides support for these three views.

Module view:

```
1 component Test
2   port port1
3   port port2
```

Listing 1. Caption

Component-and-connector view:

Deployment view:

Listing 1

Architectural Control in Trinity.

References

- [1] Architectural Languages Today. URL <http://www.di.univaq.it/malavolta/al/>. Accessed August 2, 2017.
- [2] Modern Software Architecture Definitions. URL <http://www.sei.cmu.edu/architecture/start/glossary/moderndefs.cfm>. Accessed August 2, 2017.
- [3] J. Aldrich, C. Chambers, and D. Notkin. ArchJava: Connecting Software Architecture to Implementation. In *Proceedings of the 24th International Conference on Software Engineering*, pages 187–197, 2002.
- [4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Second edition, October 2010.
- [5] D. C. Luckham and J. Vera. An Event-Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, 21(9):717–734, September 1995.
- [6] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang. What Industry Needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, June 2013.