SA Assignment 3

Connectors are architectural building blocks used to model interactions among components and rules that govern those interactions. Unlike components, connectors may not correspond to compilation units in an implemented system. They may be implemented as separately compliable message routing devices, but may also manifest themselves as shared variables, table entries, and buffers, instructions to a linker, dynamic data structures, and sequences of procedure calls embedded in code, initialization parameters, client-server protocols, pipes, SQL links between a database and an application, and so on.

Connectors can be classified on the following basis:

* **Interface**- A connector’s interface is a set of interaction points between the connector and the components and other connectors attached to it. Since a connector does not perform any application-specific computations, it exports as its interface those services it expects of its attached components. Connector interfaces enable proper connectivity of components and their interaction in an architecture and, thereby, reasoning about architectural configurations.
* **Types-** Connector types are abstractions that encapsulate component communication, coordination, and mediation decisions. Architecture-level interactions may be characterized by complex protocols. Making these protocols reusable both within and across architectures requires that ADLs model connectors as types. This is typically done in two ways: as extensible type systems, defined in terms of interaction protocols, or as built-in, enumerated types, based on particular implementation mechanisms.
* **Semantics -** Similarly to components, connector semantics is defined as a high-level model of a connector’s behavior. Unlike components, whose semantics express application-level functionality, connector semantics entail specifications of interaction protocols. ADLs may support modeling of connector semantics in order to enable component interaction analysis, consistent refinement of architectures across levels of abstraction, and enforcement of interconnection and communication constraints.
* **Constraints-** Connector constraints ensure adherence to intended interaction protocols, establish intra-connector dependencies, and enforce usage boundaries. An example of a simple and easily enforceable constraint is a restriction on the number of components that interact through the connector. Establishing adherence to more complex connector constraints (e.g., minimal throughput) may require access to information external to the given connector (e.g., a model of the attached components’ dynamic semantics).
* **Evolution -** Analogously to component evolution, the evolution of a connector is defined as the modification of (a subset of) its properties, e.g., interface, semantics, or constraints on the two. Component interactions in architectures are governed by complex and potentially changing and expanding protocols. Furthermore, both individual components and their configurations evolve. ADLs can accommodate this evolution by modifying or refining existing connectors with techniques such as incremental information filtering, sub-typing, and refinement.
* **Non-Functional Properties —** A connector’s non-functional properties are not entirely derivable from the specification of its semantics. They represent additional requirements for correct connector implementation. Modeling non-functional properties of connectors enables simulation of runtime behavior, analysis of connectors, constraint enforcement, and selection of appropriate off-the-shelf connectors (e.g., message buses) and their mappings to processors.