ISSN No. 0976-5697

Volume 3, No. 1, Jan-Feb 2012

International Journal of Advanced Research in Computer Science

REVIEW PAPER

Available Online at www.ijarcs.info

Comparative Performance Evaluation Of Software Architectural Styles With UML

Kamna Gauri*, Dipanwita Thakur
Dept.of Computer Science & Electronics,
AIM & ACT Banasthali University, Tonk (Rajasthan), India
kamna.gk@gmail.com
dipanwita.thakur@gmail.com

Abstract: This paper presents a performance evaluation of different Software Architectural styles. We all have seen many books and articles to consider the feel of different Software Architectural styles of a system. A good architecture should be approachable, simple, clear separation of concerns, resilient, balanced distribution of responsibilities. Here we provide an introduction to the field of Software Architecture, Software Architectural styles. Software Architecture is also described as a strategic design i.e. how a solution is implemented and many application product lines are built around core architecture with variants that satisfy particular customer requirements.

Keywords: architectural style; component; connector; architectural structure framework

I. INTRODUCTION

A Software Architectural style defines a structure or structures of the system in terms of software elements (components, connectors), externally visible properties of these elements and the relationships among them [1]. An awareness of these Architectural styles can simplify many problems of stakeholders in defining the system architecture because these styles play a role of the vehicle for stakeholder communication in the case of large and heterogeneous system. The taxonomy of Architectural Styles and the case studies have incorporated parts of several published papers. To a lesser extent material has been drawn from other articles by the authors.

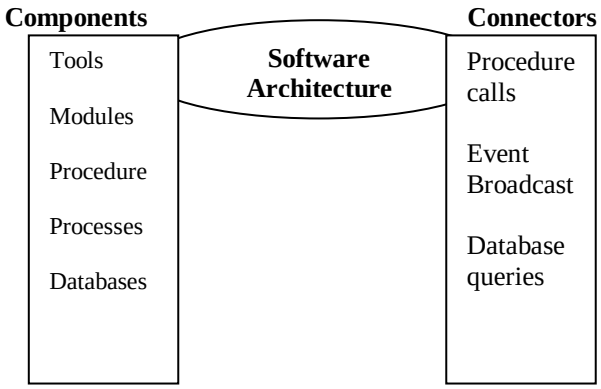
Software Architecture determines how system components are identified and allocated, how the components interact to form a system, the amount and granularity of communication needed for interaction, and the interface protocols used for communication among stakeholders: Customers, managers, designers, programmers, tester and maintainer. Architectural styles define classes of designs along with their associated known properties. Generally these styles manifests the earliest set of design decisions such as constraints on implementation, articulates organizational structure and enables quality attributes [5].

"Abstraction layering and system decomposition provide the appearance of system uniformity to clients, yet allow Helix to accommodate a diversity of autonomous devices. The architecture encourages a *client server model* for the structuring of applications."

UML has established itself as a leading Object oriented analysis and design methodology. It is a language for specifying, constructing, visualizing and documenting artifacts of software intensive systems. [3]

While architectural concepts are often embodied in infrastructure to support specific architectural styles and in the initial conceptualization of a system configuration, the lack of an explicit, independently characterized architecture or architectural style can significantly limit the benefits of software architectural design. [4]

In any style components are connected by connectors as they are the building block of architecture.



Software Architecture = { Elemnets, Form, Rationale }

According to [6] there are three classes of software elements namely processing elements, data elements and connecting elements.

An important aspect of Software Architecture is representation. The representational clarity and power of a Software Architectural description is very significant and determines the completeness and hence success of the succeeding steps in the software development lifecycle which build on this description. Researchers have used text, boxes-and-lines [5], data flow diagrams and other pictorial and textual methods for this purpose.

The software architecture of a system can be therefore represented as a set of views, and a combination of these views comprises of the system as a whole. The major drawback of using the view approach is that the representation becomes view-centric; the side effect is redundancy among different views. These redundancies can

cause inter-view mismatches, such as inconsistencies and incompleteness [7].

Though the different ADLs server their purpose satisfactorily, each of them embodies a particular approach to the specification and evolution of an architecture, with specialized modeling and analysis techniques that address specific system aspects in depth [8].

However, this emphasis on depth over breadth of the model can make it difficult to integrate these models with other development artifacts, because the rigor of formalisms tends to ignore the more day-to-day development concerns [9].

II. PIPE AND FILTER

The pipe and filter architectural style provides a structure that process a stream of data. This style has only one component type i.e. filter and this filter does some transformation and passes data to other filters through pipes. Recombining filters allows you to build families of related filters. Pipes are the connectors between a data source and the first filter, between filters, and between the last filter and a data sink.

Data source is an entity (e.g., a file or input device) that provides the input data to the system. It may either actively push data down the pipeline or passively supply data when requested, depending upon the situation.

Data sink is an entity that gathers data at the end of a pipeline. It may either actively pull data from the last filter element or it may passively respond when requested by the last filter element.

A. Style Invariants:-

a. Filters are independent (no shared state)
b. Filter has no knowledge of up- or down-stream filters

B. Suitability:-

a. Well suited for those systems that mainly do data transformation

C. Advantages:-

a. Filter is independent i.e. no need to know the id of filters sending/receiving data.
b. A pipe is a two way connector.

D. Disadvantages:-

a. A pipe is a unidirectional channel which moves streams of data from one filter to another.
b. A pipe must connect the output port of one filter to input port of another.

III. IMPLICIT INVOCATION STYLE

Instead of invoking a style procedure directly.....

a. A component can announce (or broadcast) one or more events.
b. Other components in the system can register an interest in an event by associating a procedure with the event.
c. When an event is announced, the broadcasting system (connector) itself invokes all of the procedures that have been registered for the event

Component interfaces are methods and events.

Connectors includes Implicit or Explicit Invocation.

A. Style Invariants:-

a. Announcers of events are unaware of the regarding which component will be affected by those event invocations.
b. Components can not make conjectures about the order of processing.

B. Suitability:-

a. Suitable for applications that involve loosely-coupled collection of components.
b. Particularly useful for those applications which includes
    a) Changing a service provider
    b) Enabling or disabling capabilities

C. Advantages:-

a. Reusability i.e. any component can be introduced into a system simply by registering it for the events of that system.
b. Easy system evolution i.e. Components may be replaced by other components without affecting the interfaces of other components in the system.

D. Disadvantages:-

a. Components relinquish computation control to the system.
b. No knowledge of what components will respond to event.
c. No knowledge of order of responses i.e. when responses are finished.

IV. REPOSITORY STYLE

When large amounts of data are to be shared, the repository style of sharing is most commonly used. Shared data is held in a central database or repository and may be accessed by all sub-systems. In order to exchange the data between sub systems, each sub-system maintains its own database and passes data explicitly to other sub-systems

Component consists of a central data structure representing the current state of the system and a collection of independent components that operate on central data structure.

Connectors consists procedure calls or direct memory access.

A. Suitability:-

a. Suitable for applications in which the central issue is establishing, augmenting, and maintaining a complex central body of information.

B. Advantages:-

a. Competent way to share the large amount of data.
b. Issues like backup, security and concurrency control are managed in a centralized manner.
c. Sharing model is circulated as the repository schema.

C. Disadvantages:-

a. Data evolution is difficult and expensive
b. Difficult to distributed data.
c. Sub systems must agree on a repository data model certainly conciliation.

V. OBJECT ORIENTED STYLE

The data representation is hidden from other objects because objects are responsible for conserving the invariance of the data demonstration.

Components are objects i.e. Data and its associated operations.

Connectors are messages and method invocation

A. Style Invariants :-

a. Internal representation is hidden from other objects.

B. Suitability:-

b. Suitable for those applications where a central issue is identifying and protecting related bodies of information/data.

C. Advantages:-

a. By locating only related methods and features in an object, and using different objects for different sets of features, one can achieve a high level of cohesion.
b. It provides for improved testability through encapsulation.
c. It provides for reusability through polymorphism and abstraction.

D. Disadvantages:-

a. While interaction with other objects an object must know the identity of that another object
b. Objects cause side effect problems i.e. if there are two objects A and B and both use object C Whenever the identity of an object changes it is necessary to modify all other objects that explicitly invoke it.

VI. LAYERED ARCHITECTURAL STYLE

In this style, system is organized hierarchically i.e. like Multi level client – server where each layer acts as both a client and a server and exposes an interface to be used by above layers.

Components are typically collections of procedure

Connectors are typically procedured calls under inhibited visibility.

For e.g. in a client server style Components are clients and servers and connectors are Remote Procedure Call based network interaction protocols.

The set of clients is often variable, and the connections are commonly made only as needed. There may be several servers providing the same services, in which case a client may connect to any, and may look up a server in some kind of directory in order to locate an appropriate one; clients may be directed to whichever server is least busy, in order to balance the load and provide fastest service. Style Invariants:-

a. Limit layer (component) interactions to adjacent layers; because in pure layer system Inner layers are hidden from all except the adjacent outer layer for certain function.
b. Virtual machine styles results from fully opaque layers.

A. Suitability:-

a. Suitable for applications that involve distinct classes of services that can be organized hierarchically.

B. Advantages:-

a. In contrast to Object Oriented style, changes in a layer affect at most the adjacent two layers.
b. Increasing abstraction levels because only carefully selected procedures from the inner layers are made available (exported) to their adjacent outer layer.
c. Reuse i.e. different implementations (with alike interfaces) of the same layer can be used interchangeably.

C. Disadvanatges:-

a. Performance requirements may force the coupling of high-level functions to their lower-level implementations.
b. Universally not applicable.
c. Difficult to find the right levels of abstraction.

VII. INTERPRETER STYLE

Interpreter architecture mimics a coded component using a program written in some language. It is a special kind of a layered architecture where a layer is implemented as a true language interpreter. Interpreter style is used in rule based system: Prolog, Coral and scripting languages: Awk, Perl.

Componnets consists one state machine for the execution engine and three memories i.e

a. current state of the execution engine
b. program being interpreted
c. current state of the program being interpreted

Connectors includes procedure calls and direct memory access.

A. Suitabilty:-

Suitable for applications in which the most appropriate language or machine for executing the solution is not directly available.

B. Advantages:-

a. Simulation of non implemented hardware
b. Facilitates portability of application or languages across a variety of platform

C. Disadvantages:-

a. Extra level of indirection slows down execution.
b. Problems with scalability.

VIII. PROCESS – CONTROL STYLE

One sub-system has overall responsibility for control and starts and stops of other sub-systems or processes.

Components are process definition which includes mechanism for manipulating some process variables and control algorithm for deciding how to manipulate process variables.

Connectors are the data flow relations for process variables, set point (desired value for a controlled variable) and sensors (to obtain values of process variables pertinent to control)

A. Style Invariants:-

a. Topologies like open loop & closed loop and Data flow i.e. forward through open loop system, circulating in closed – loop system

B. Suitability:-

a. Suitable for applications whose purpose is to maintain specified properties of the outputs of the process at given reference values.

C. Advantages:-

a. Efficiency i.e. using continuous updates and by following a moving intention.

D. Disadvantages:-

a. Completeness & correctness
b. Control system analysis methods.

IX. EVENT BASED STYLE

In event-based implicit invocation architecture, components register their interest in specific events with an event manager. As part of the registration process, each component makes a callback method on its provided interface available to the event manager as one of its required interfaces. Thereafter, whenever a registered event is detected by the event manager, it calls the callback method of every component registered for it and then gives each one the event.

Components are independent, concurrent event generators and/or consumers

Connectors includes event buses for data sending

A. Style Invariants:-

a. Components communicate with the event buses, not directly to each other

B. Suitability:-

a. Prevalent for large-scale distributed applications is the event-based style.

C. Advantages:-

a. Highly decoupled, Easy to evolve, effective for highly distributed applications.

D. Disadvantages:-

a. However, they are not very flexible. If, for instance, an object of interest is interested in producing an event notification on a number of subjects or channels, it has to explicitly publish the notification on all of them.
b. If we assume that the event service is implemented as a centralized element, it can rapidly become a critical bottleneck as the number of components it has to serve grows.

X. PEER – TO - PEER STYLE

In a peer-to-peer architecture, all components are at the same level and each may require the interface of any or every other component. A peer-to-peer architecture provides little structure for a system. State and behavior of clients or servers are distributed among peers. Network messages take place in the form of data elements.

Components independent components (peers) having their own state and control thread

Connectors Network protocols, generally ritual

A. Style Invariants:-

a. Network (may have redundant connections between peers) can vary arbitrarily and dynamically.

B. Sutability:-

b. Suitable for decentralized computing with flow of control and distributed resources among peers.

C. Advantages:-

a. Highly robust in the face of failure of any given node.
b. Scalable in terms of access to resources and computing power.

D. Disadvantages:-

a. Vigilance on protocols

XI. BLACK BOARD STYLE

Blackboard architecture has a blackboard component, acting as a central data repository, and a number of components that act independently on the common data structure stored in the blackboard, responding to changes in it and in response making further changes. The components interact only through the blackboard.

Components are of two types i.e. Central data structure and Components which are operatable on the Central data structure.

Connectors are the blackboard state as system is entirely controlled by these states.

A. Suitability:-

a. Suitable for those problems for which no deterministic solution strategy is known.
b. Suitable for artificial intelligence systems such as vision, speech and pattern recognition.

B. Advantages:-

a. Ensures data integrity and Reliable, secure, testability guaranteed

C. Disadvantages:-

a. Problems with scalability

XII. RULE BASED STYLE

Inference engine parses user input and determines whether it is a fact/rule or a query. If it is a fact/rule, it adds this entry to the knowledge base. Otherwise, it queries the knowledge base for valid rules and attempts to resolve the query. Facts and queries takes place as data elements. Rule-based systems provide a means of codifying the problem-solving knowhow of human experts. These experts tend to capture problem-solving techniques as sets of situation-action rules whose execution or activation is sequenced in response to the conditions of the computation rather than by a predetermined scheme. [2]

Components are user interface, inference engine, knowledge base

Connectors are tightly interconnected, with direct procedure calls and/or shared memory.

A. Advamtages:-

a. Behavior of application can be very easily modified through addition or deletion of rules from the knowledge base

B. Disadvantages:-

a. When a large number of rules are involved then understanding the interactions between multiple rules affected the same facts can become very difficult.

Table I. Architectural Structural Framework [10]

Architectural structures	Components	Relations	Use
Module structure	work assignments	is-a-sub module-of	Allocating a project's labor and other resources during development and maintenance.
Conceptual or Logical structure	Abstractions of the system's functional requirements.	shares-data-with	Understanding the interactions between units in the problem space.
Process or Coordination Structure	Processes or threads	– Synchronizes-with – Can't-run-without – Can't-run-with – Preempts,	Modeling dynamic aspects of a running system.
Physical structure	Hardware (computers, networks, etc.)	communicates-with	Create models to reason about performance, availability, security, etc
Uses structure	procedures or modules	assumes-the-correct-presence-of	To model system extendibility and incremental system building (e.g., Make file dependencies).
Calls structure	Procedures	calls	To model trace of execution in a program.
Data Flow structure	Programs or modules	transmits-data-to	To model data transmission, this can aid requirements traceability.
Class structure	classes and interfaces	inherits-from, implements	To model collections of similar behavior and parameterizes differences.

XIII. REFERENCES

[1] Bass, L., Clements, P., Kazman, R.: .Software Architecture in Practice,. Addison-Wesley, 1998, ISBN 0-201-19930-0.

[2] F. Hayes-Roth, 'Rule-based systems," Communications of the ACM, vol. 28, pp. 921-932, September 1985.

[3] Abdurazik, A.: "Suitability of the UML as an Architecture Description Language with Applications to Testing," February 2000, ISE-TR-00-01

[4] Allen, R., Garlan, D.: "A formal basis for Architectural Connection," ACM Transactions on Software Engineering and Methodology (TOSEM) Volume 6 , Issue 3, Pages: 213 - 249, 1997

[5] Evaluating Software Architectures: Methods & Case Studies, Addison-Wesley, ISBN 020170482X.

[6] Perry, D., Wolf, A.: "Foundations for the study of software architecture," ACM SIGSOFT Software Engineering Notes, Volume 17 , Issue 4 (October 1992), Pages: 40 - 52, 1992.

[7] Egyed, A. and Medvidovic, N. "Extending Architectural Representation in UML with View Integration," Proceedings of the 2nd International Conference on the Unified Modeling Language (UML), Fort Collins, CO, October 1999, pp. 2-16 (44 papers out of 166).

[8] Egyed, A. and Medvidovic, N. "Consistent Architectural Refinement and Evolution using the Unified Modeling Language," Proceedings of the 1st Workshop on Describing Software Architecture with UML, co-located with ICSE 2001, Toronto, Canada, May 2001, pp. 83-87.

[9] Medvidovic, N., Rosenblum, D.: "Assessing the Suitability of a Standard Design Method for Modeling Software Architectures." In Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), pages 161-182, San Antonio, TX, February 22-24, 1999.

[10] Design & Use of Software Architectures: Adopting and Evolving a Product Line Approach: ISBN: 0201674947.