Applying 4+1 View Architecture with UML 2

White Paper

first FCG
Software Services

Copyright ©2007 FCGSS, all rights reserved.
www.fcgss.com

Introduction

Unified Modeling Language (UML) has been available since 1997, and UML 2 was released in 2004, building on an already successful UML 1.x standard. UML 2 comes with 13 basic diagram types to support Model Driven Architecture (MDA) and Model Driven Development (MDD).

Philippe Kruchten originally presented the 4+1 View Model to describe the architecture of software-intensive systems. This approach uses multiple views to separate stakeholders' concerns. The 4+1 View Approach is widely accepted by the software industry to represent application architecture blueprints. However, the industry has not yet completely embraced UML 2. IT architects are continuing with UML 1.x artifacts to represent architecture and are consequently missing out on the powerful benefits of the improvements made in UML 2.

This article presents the approach for 4+1 View Architecture using UML 2 diagrams. It enhances the original concept of the approach using the current modeling standards and techniques, and aims to encourage Application Architects to adapt relevant UML 2 artifacts. This article discusses each of the views and allocates the UML 2 diagrams to these views. It does not, however, teach semantics and modeling using UML 2 notations. The audience is expected to have a basic understanding of UML and to be able to refer to UML 2 for further details.

UML 2 Diagrams

Let us briefly review the diagrams available in UML 2 Specification.

UML 2 Superstructure Specification divides the 13 basic diagram types into two key categories:

- Part I – Structural Diagrams: These diagrams are used to define static architecture. They comprise static constructs such as classes, objects, and components, and the relationships between these elements. There are six structural diagrams: Package Diagrams, Class Diagrams, Object Diagrams, Composite Structure Diagrams, Component Diagrams and Deployment Diagrams.
- Part II – Behavioral Diagrams: These diagrams are used to represent dynamic architecture. They comprise behavioral constructs such as activities, states, timelines and the messages that run between different objects. These diagrams are used to represent the interactions among various model elements and instantaneous states over a time period. There are seven behavioral diagrams: Use Case Diagrams, Activity Diagrams, State Machine Diagrams, Communication Diagrams, Sequence Diagrams, Timing Diagrams and Interaction Overview Diagrams.

UML 2 has introduced Composite Structure, Object, Timing and Interaction Overview diagrams. The remaining diagrams were borrowed from UML 1.x, although some of them were changed significantly.

4+1 View Architecture

The fundamental organization of a software system can be represented by:

- Structural elements and their interfaces that comprise or form a system
- Behavior represented by collaboration among the structural elements
- Composition of Structural and Behavioral elements into larger subsystems

Such compositions are guided by desired abilities (non-functional requirements) like usability, resilience, performance, re-use, comprehensibility, economic and technology constraints and trade-offs etc. Also, there are cross-cutting concerns (like security and transaction management) that apply across all the functional elements

Architecture also means different things to different stakeholders. For example, a Network Engineer would only be interested in the hardware and network configuration of the system; a Project Manager in the key components to be developed and their timelines; a Developer in classes that make up a component; and a Tester in scenarios. So we need multiple view points for distinct stakeholders' needs, showing what is relevant while masking the details that are irrelevant.

The 4+1 View Approach is an 'architecture style' to organize an application's architecture representations into views to meet individual stakeholder's needs. Figure 1 shows the views in the 4+1 View Architecture.

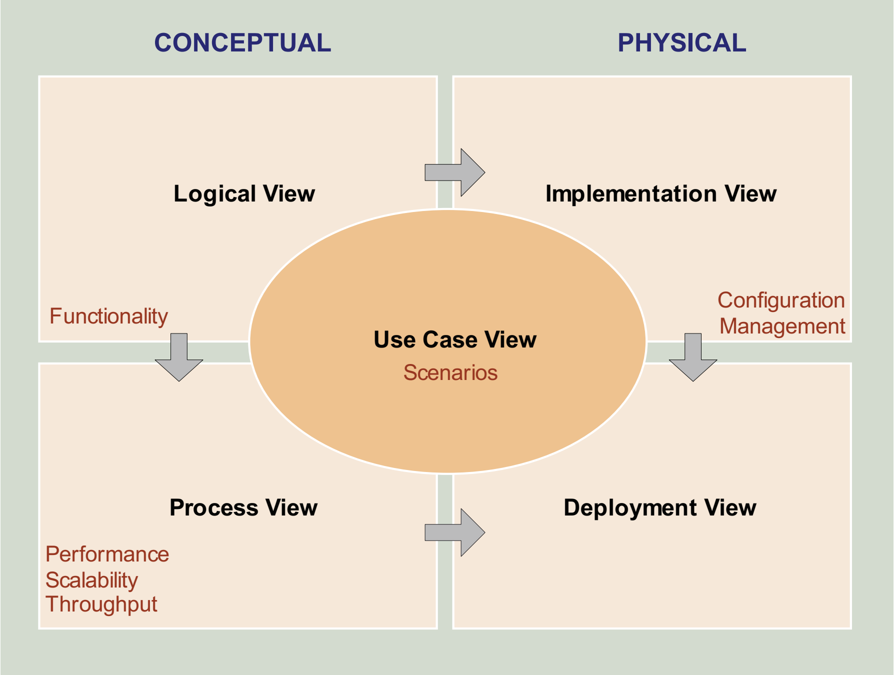


Figure 1: 4+1 View Model

> *Software Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.*
>
> — The definition of Software Architecture as per IEEE Recommended Practice for Architectural Description of *Software-Intensive Systems (IEEE 1471-2000)*

Logical View (Object Oriented Decomposition)

This view focuses on realizing an application's functionality in terms of structural elements, key abstractions and mechanisms, separation of concerns and distribution of responsibilities. Architects use this view for functional analysis.

The logical architecture is represented at different levels of abstraction and progressively evolves in iterations.

1. Vertical and horizontal divisions
   - The application can be vertically divided into significant functional areas (i.e., order capture subsystems, order processing subsystems).
   - Or, it can be horizontally divided into a layered architecture distributing responsibilities among these layers (i.e., presentation layers, services layers, business logic layers, and data access layers).
2. Representation of structural elements as classes or objects and their relationships.

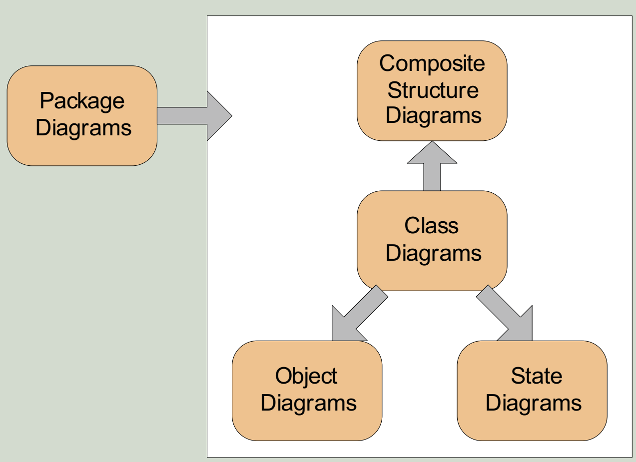
UML 2 provides an elaborate set of diagrams to create a Logical View:

1. Class Diagrams or Structural Diagrams: These diagrams define the basic building blocks of a model. They focus on each individual class, the main operations and relationships to other classes' associations, usage, composition, inheritance etc.
2. Object Diagrams: These diagrams show how instances of structural elements are related. They help understand the class diagrams when the relationships are complex. The object diagrams were informally used in the UML 1.x world; in UML 2 they are formal artifacts.
3. Package Diagrams: These diagrams are used to divide the model into logical containers or 'packages'. They can be used to represent vertical and horizontal divisions as packages.
4. Composite Structure Diagrams: These diagrams help in modeling the parts contained by a class and the relationships between the parts. When parts are related, such diagrams significantly simplify the relationships between the classes. Ports are used to represent how a class hooks into the environment. These diagrams support collaborations that can be used to represent design patterns of cooperating objects. UML 2 has introduced these diagrams as a major improvement over earlier structural constructs.
5. State Machine Diagrams: These diagrams are necessary to understand the instant states of an object defined by a class. These diagrams are optionally used when there is a need to understand the possible states of a class.

The diagram notations are intentionally kept out of the scope of this document. Please refer to Sparx Systems' online tutorial to understand the notations (http://sparxsystems.com/resources/uml2_tutorial/).

While modeling for the Logical View, start with the Class and Package diagrams and expand as necessary. Figure 2 shows the modeling approach for the Logical View. UML also provides profiles for data modeling using Entity Relationship (ER) Diagrams. ER Diagrams can also be considered as another form of Logical View. Some Architects prefer to capture ER Diagrams in a separate view called Data View.

MODELING LOGICAL VIEW WITH UML2



1. Start with class diagrams to model the system
2. Use package diagrams to logically group diagrams

Optional use
3. Object diagrams when relationships between classes need to be explained through instances
4. State Charts when internal states of a specific class are to be explained
5. Composite Structures when parts of a class and relationships between parts are to be modeled

Figure 2: Modeling Logical View

Process View (Process Decomposition)

This view considers non-functional aspects such as performance, scalability and throughput. It addresses the issues of concurrency, distribution and fault tolerance. It shows the main abstractions from the Logical View executing over a thread as an operation. A process is a group of tasks that form an executable unit; a software system is partitioned into sets of tasks. Each task is a thread of control that executes with collaboration among different structural elements (from the Logical View). Process View also encompasses re-usable interaction patterns to solve recurring problems and to meet non-functional service levels.

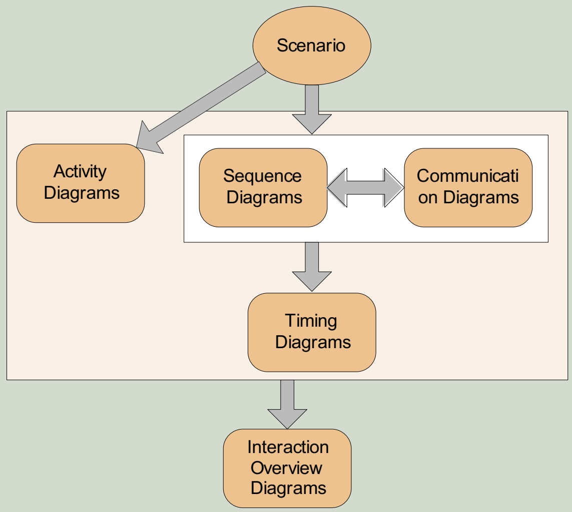
The process architecture can be represented at various levels of abstraction such as interactions between systems, subsystems and objects etc. based on the need.

The Process view can be represented by the following UML 2 diagrams:

1. Sequence Diagrams: These diagrams show the sequence of messages passed between the objects on a vertical timeline. UML 2 has made significant improvements on Sequence diagram notations to aid Model Driven Development. The fragment types such as loop, assert, break and alt help in diagramming to the level of detail that keeps code and models in sync – not just in terms of structure, but also in behavior. Today's modeling tools are yet to catch up on utilizing the power of UML 2 Sequence diagrams.
2. Communication Diagrams: These diagrams show communications between objects at runtime during a collaboration instance. These diagrams are closely related to Sequence diagrams. While Sequence diagrams are focused on the flow of messages throughout an interaction over a timeline, the Communication diagrams focus on the links between the participants; they were previously called Collaboration diagrams. In addition to changing the name, UML 2 has made improvements relative to Sequence diagrams.
3. Activity Diagrams: These diagrams are similar to flowcharts and have a wide range of uses in different view points. In the process view, they can be used to depict the program flows and complex business logic with actions, decision points, branching, merging and parallel processing. UML 2 has made several improvements and standardized the Activity Diagram constructs. Now you can also represent the time events, external sending and receiving signals in the Activity Diagrams.
4. Timing Diagrams: Timing Diagrams specifically address modeling for performance. They depict the amount of time allowed for a participant to receive events and switch between the states, and how long a participant can stay in a specific state. These diagrams were introduced in UML 2 and can be used for performance design.
5. Interaction Overview Diagrams: These diagrams provide an overview of how several interactions work together to implement a system concern. They are a fusion of Activity, Sequence and Timing Diagrams. Each part of an interaction could be represented by a distinct diagram type. UML 2 introduces these diagrams and they can be used to aid high-level overviews and understand the overall system behavior.

While modeling for Process View, you can start with either Sequence or Communication diagrams. Both these diagrams can be created from each other, so it is merely a personal preference on which one to use. As the scenarios get complex, you can use other diagrams. Figure 3 shows the order of modeling for Process View with UML 2 diagrams.

MODELING PROCESS VIEW WITH UML2



1. **Use either Sequence or Communication Diagrams for modeling simple interactions in use case realizations**

Optional use

2. **Add Activity diagrams to realize scenarios where business logic is a sequence of actions and involves branching and parallel processing**
3. **Add timing diagrams when modeling for performance**
4. **For complex scenarios, that can be composed of other scenarios, use Interaction overview diagrams**

Figure 3: Modeling Process View

Implementation or Development View (Subsystem Decomposition)

This is a view of a system's architecture that encompasses the components used to assemble and release a physical system. This view focuses on configuration management and actual software module organization in the development environment. The software is actually packaged into components that can be developed and tested by the development team. While the Logical View is at the conceptual level, the diagrams in this view represent the physical-level artifacts that are built by the team.

Component Diagrams are used to represent the Implementation View. These diagrams show different components, the ports available and the dependencies on the environment in terms of provided and required interfaces. UML 2 has improved the Component Diagrams specifically with the interfaces and ports. The components can be tied to the classes and composite structures that realize these components. These diagrams can now be used to precisely represent the software components built in a system and their dependencies both in black-box and white-box views.

Deployment or Physical View (Mapping Software to Hardware)

This view encompasses the nodes that form the system's hardware topology on which the system executes; it focuses on distribution, communication and provisioning.

The software executes on a network of computers, or processing nodes. The various elements such as processes, tasks and objects need to be mapped to the nodes on which they execute. These physical configurations can differ between production, development and testing environments. The software should be built to be flexible to scale across these hardware changes. Hence, this view accommodates the non-functional requirements such as availability, reliability, performance, throughput and scalability.

This view provides all possible hardware configurations, and maps the components from the Implementation View to these configurations.

Deployment Diagrams show the physical disposition of the artifacts in the real-world setting. UML provides constructs to represent Nodes such as devices, execution environment and middleware; artifacts such as jar files and connections; and dependencies between these devices. The nodes can be embedded, for example, representing the application server running within the physical device. UML uses very simple notation for nodes and artifacts. However, the current modeling tools allow you to import external images to depict these nodes.

Use Case View or Scenarios (putting all together)

In addition to the four views discussed above, this is the central view for capturing scenarios. The Use Case View encompasses the use cases that describe the behavior of the system as seen by its end users and other stakeholders. Although traditionally discussed as the last view, this is the first view created in the system development lifecycle.

This view represents the scenarios that tie the four views together, and forms the reason why all the other views exist. With all other views in place, this view seems redundant (hence +1). However it represents the architecturally significant requirements in the form of scenarios. It also aids to verify that all the required scenarios are met.

UML 2 provides the Use Case Diagrams to represent this view. These diagrams are comprised of use cases and actors. They are closely tied to detailed scenario descriptions in text. As an architecture view, we are only interested in significant use cases that need to be modeled.

Activity Diagrams can also be used to represent scenarios and business processes. The current UML tools are well geared to generate the business process automation code from activity diagrams. So the activity diagrams can be used to both represent the requirements in the Use Case View and executable processes in the Process View.

Figure 4 shows the UML diagrams allocated to the views on the 4+1 View Model.

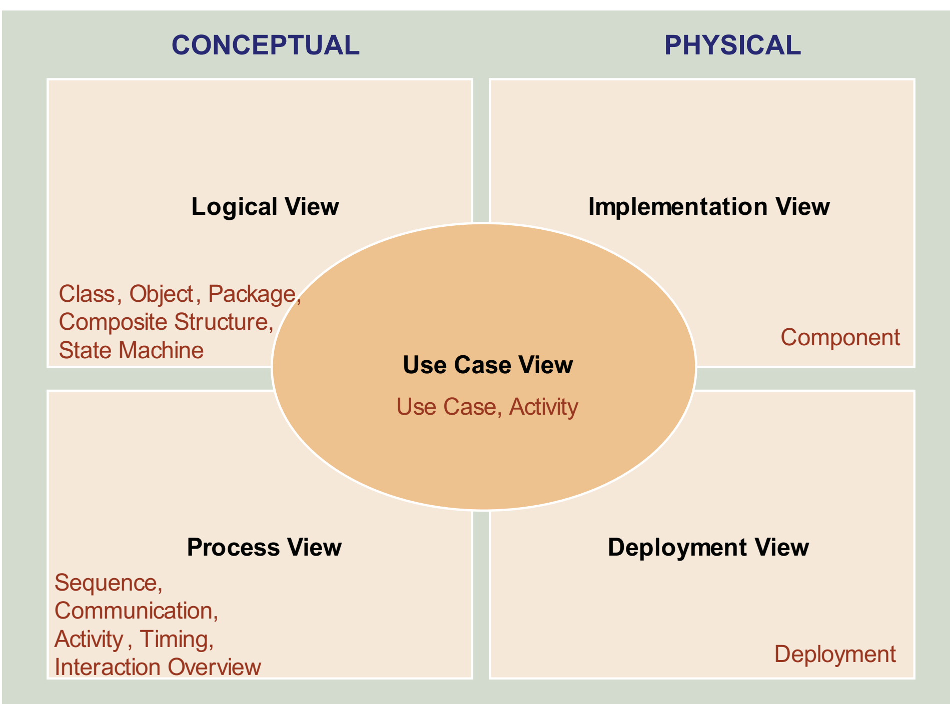


Figure 4: 4+1 View Model with UML2

Relationships between Views

The Logical View and the Process View are at a conceptual level and are used from analysis to design. The Implementation View and the Deployment View are at the physical level and represent the actual application components built and deployed.

The Logical View and the Implementation View are tied closer to functionality. They depict how functionality is modeled and implemented. The Process View and Deployment View realizes the non-functional aspects using behavioral and physical modeling.

Use Case View leads to structural elements being analyzed in the Logical View and implemented in the Development View. The scenarios in the Use Case View are realized in the Process View and deployed in the Physical View.

Conclusion

UML 2 has made significant improvements in its constructs that affect the representations in Logical and Process Views. Architects today have additional diagrams to model the functional and non-functional aspects and communicate them to the teams.

Typically, the views in the 4+1 View approach are sufficient to model the Application Architecture. Additional views such as Security View and Data View could be added based on the specific application's requirements. It should also be noted that the 4+1 View Approach is best suited to represent the Application Architecture. More complex view points are necessary if you need to depict the Enterprise Architecture for an entire organization.

References and Further Reading

1. Philippe Kruchten's original publication on 4+1 View Approach, www.win.tue.nl/~mchaudro/sa2004/Kruchten4+1.pdf
2. UML 2 Superstructure Specification from OMG, http://www.omg.org/technology/documents/modeling_spec_catalog.htm
3. Grady Booch's presentation on Software Architecture, www.booch.com/architecture/blog/artifacts/Software%20Architecture.ppt
4. Learning UML 2.0 (O'Reilly), Russ Miles & Kim Hamilton
5. UML 2.0 in a Nutshell (O'Reilly), Dan Pilone and Neil Pitman
6. UML 2 Tutorials from Sparx Systems, (http://sparxsystems.com/resources/uml2_tutorial/)

Acknowledgements

My acknowledgements to Martin Israelsen, Director of Technology and Architecture, FCG Software Services, for his technical review and invaluable inputs on this article. Special thanks to Suzanne Cain, Marketing Specialist, FCG Software Services, for editing this document and Papia Roy, First Consulting Group, for designing and finalizing it.

About the Author

Veer Muchandi is a Senior Technical Architect with FCG Software Services. He has been providing IT Solutions for over 14 years. His areas of expertise include requirements, business modeling and solution architecture. He is a Sun Certified Enterprise Architect and OMG Certified UML Professional. Veer lives in Atlanta, Georgia.

About FCG Software Services

FCG Software Services (FCGSS), a division of First Consulting Group, is a U.S. software development company that has provided on-site/offshore software development and other IT-enabled services for over ten years. Assessed at CMMI Level 5, FCGSS has offshore development centers located in Bangalore, India, and Ho Chi Minh City, Vietnam. FCGSS' experienced personnel work on-site with organizations to define requirements, design software solutions and coordinate the activities of the offshore development teams. Through a combination of qualified resources, a sound development methodology and a wide domain of technical expertise, FCGSS delivers high quality products at significantly reduced rates.

About the Publisher

Sparx Systems is a software development company specializing in high performance, and highly productive and scalable visual tools for the planning, design and construction of software-intensive systems. With customers in industries ranging from aerospace and automotive engineering to finance, defense, government, entertainment and telecommunications, Sparx Systems is a leading vendor of innovative solutions based on the Unified Modeling Language (UML) and its related specifications. A Contributing Member of the Object Management Group (OMG), Sparx Systems is committed to realizing the potential of model-driven development based on open standards. The company's flagship product, Enterprise Architect, has received numerous accolades since its commercial release in August, 2000. Now at version 6.5, Enterprise Architect is the design tool of choice for over 100,000 registered users in more than 60 countries worldwide.

first FCG™
Software Services

11675 Great Oaks Way, Suite 144
Alpharetta, GA 30022
www.fcgss.com
678-229-1100

©2007 FCGSS_US_WP_Applying 4+1 w_UML2