P06. *Architecture Viewpoint Framework*

2014

Sungwon Kang

6. 소프트웨어 아키텍처를 보는 관점체계

6.1 관점과 뷰

6.2 관점체계

6.3 아키텍처 뷰의 개발 순기에 있어서의 역할

6.4 아키텍처 드라이버와 아키텍처 관점들

6.5 요약

6.1 관점과 뷰

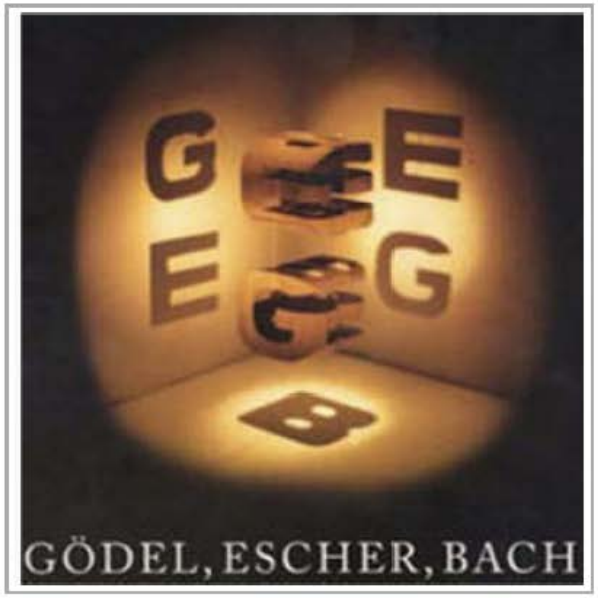
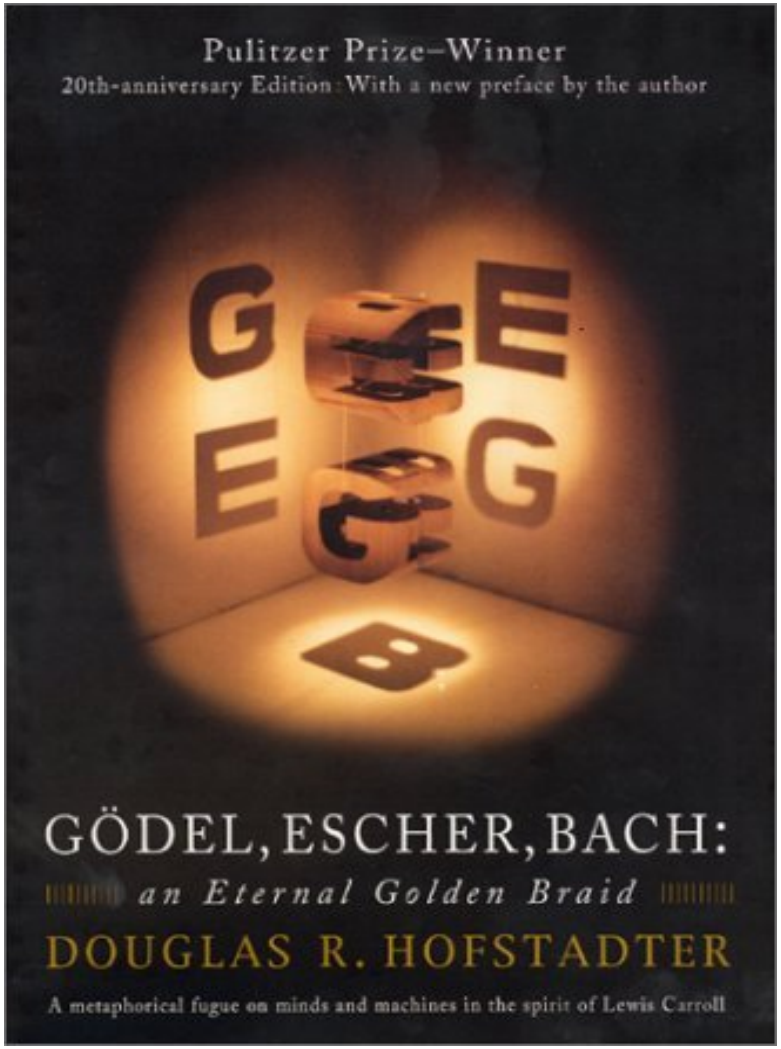


그림 6-1. 세 개의 관점에서 빛을 비추어 본 나무조각[Hoftstader 79]

- Software takes many forms in its lifecycle.
  => Architect needs to design the forms in a consistent and complete manner.
- In some sense, the code and the running software contain all the essentials but “design intentions” are not easily captured, which are important. Why?

Viewpoint and View

- A *viewpoint* is a perspective or an aspect from which the system is viewed at architectural level (or views can be taken)
- A *view* is a representation of a set of system elements and the relations between them



Viewpoints

- "A software architecture is the set of significant decisions about
  the organization of a software system,
  the selection of the structural elements and
  their interfaces by which the system is composed ..." [Booch 99]
- **But what kinds of structure**?
- Many possibilities:
  - *modules*, showing composition/decomposition, mapping to code units
  - *processes*, and how they synchronize
  - *programs*, and how they call or send data to each other
  - how software is deployed on *hardware*
  - how *teams* cooperate to build the system
  - how should pieces of the system work as *components* and *connectors* at run-time

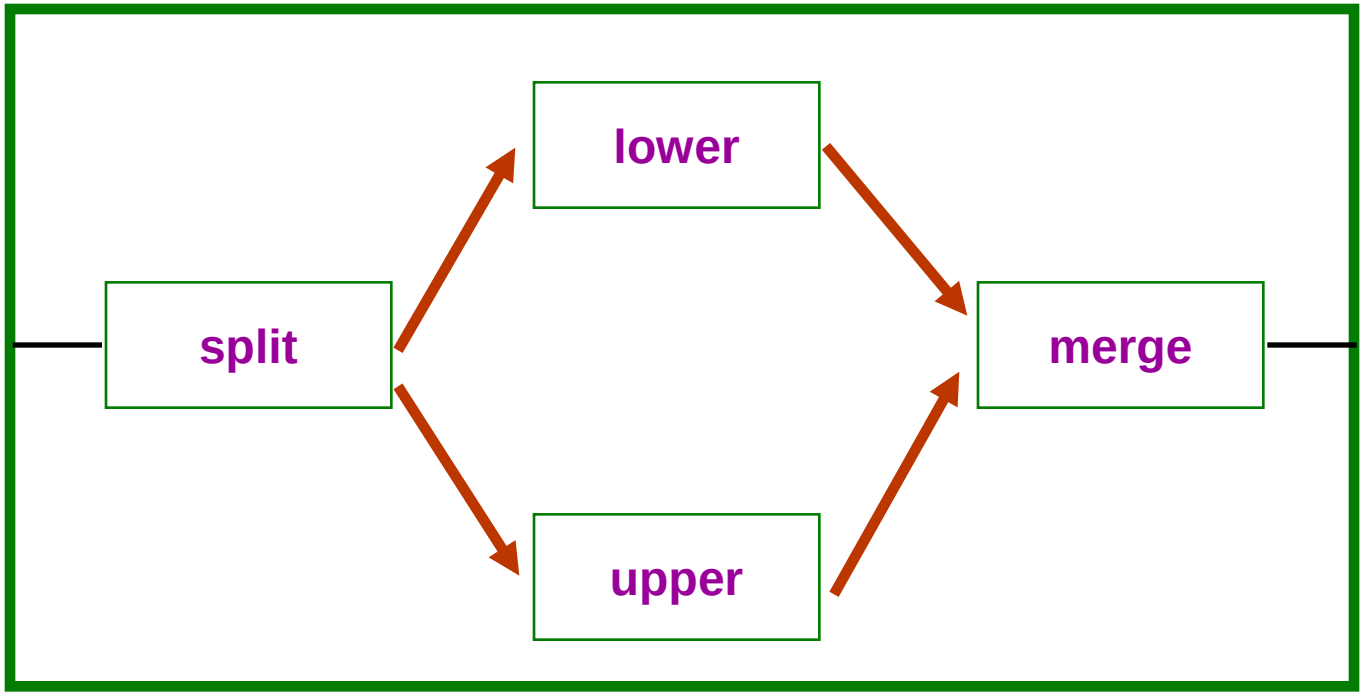
A way of explaining the notion of viewpoint

- Each of these can be the basis of an Architectural Viewpoint

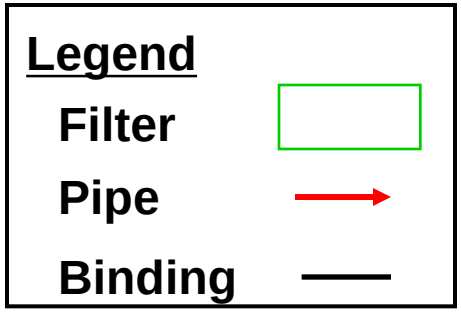
* Source of the slide: David Garlan

1. The Logical Viewpoint

논리뷰 예제: Alternating Characters Code



Originally called "runtime view" by D. Garlan



*** Source of the slide:**
David Garlan

The Logical viewpoint

- Elements (= Components)
  - Computation or functional units

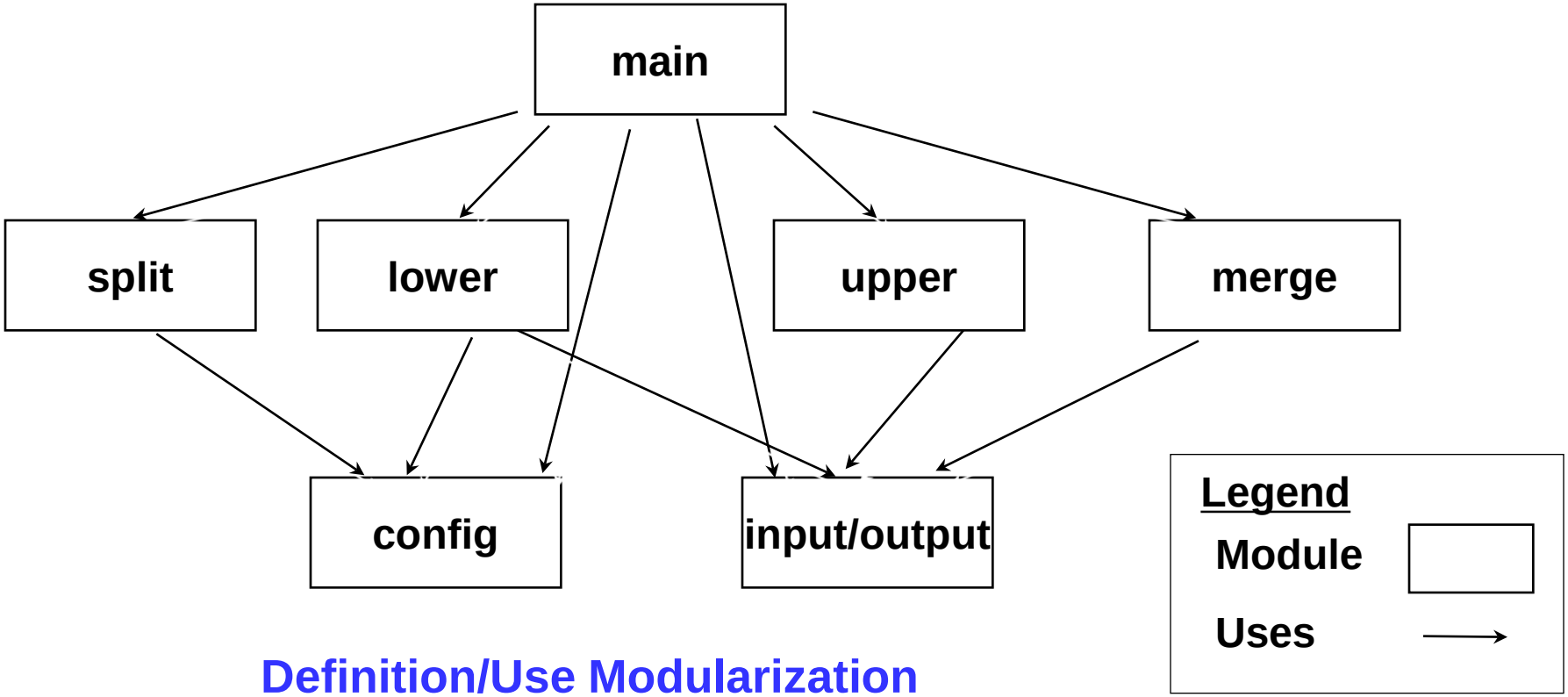
- Relations (= Connectors)
  - Connection and collaboration such as:
    - Data flow (e.g. message passing, conversion)
    - Communication & coordination (interaction)
    - Facilitation (more sophisticated connection)

  ☛ However, at this stage we don't want to be specific about the nature of connection and collaboration

2. The Module viewpoint

모듈뷰 예제: **Alternating Characters Code**

Produces alternating case of characters in a stream



Definition/Use Modularization

* Source of the slide: David Garlan

The Module viewpoint

- Elements: Modules. A module is a code unit that implements a set of responsibilities.

- Relations: Relations among modules include

  (a) A is part of B. This defines a part-whole relation among modules

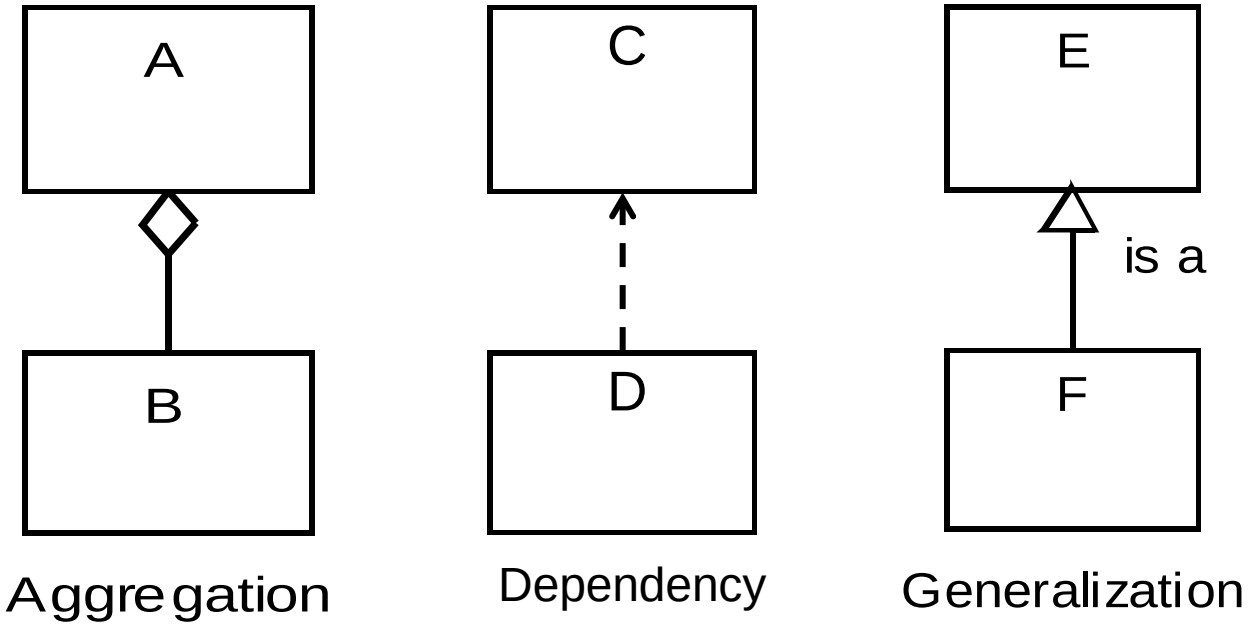
  (b) A depends on B. This defines a dependency relation among modules.

  (c) A is a B. This defines specialization and generalization relations among modules.

* Source of the slide: David Garlan

The Module viewpoint

- Modern Notations for Module Views
  - Informal: box-and-line, with nesting
  - UML: Class diagrams



* Source of the slide: David Garlan

3. The Runtime Viewpoint

Run-time viewpoint

- The architectural definition of a system identifies
  - Components: define the locus of computation
    - Examples: filters, databases, objects, ADTs
  - Connectors: mediate interactions of components
    - Examples: procedure call, pipes, event broadcast
  - Properties: specify information for construction & analysis
    - Examples: signatures, pre/post conditions, RT specs, protocols

* Source of the slide: David Garlan

"To be precise, should read 'runtime component' and 'runtime connector', which are special versions of component and connector in component and connector dichotomy." - KSW

Run-time viewpoint - Issues Addressed

- Gross decomposition of a system into interacting (runtime) components
  - typically hierarchical
  - using rich abstractions for "glue"
  - often using common design idioms/styles
- Emergent system properties
  - performance, throughput, latencies
  - reliability, security, fault tolerance, evolvability
- Rationale
  - relates requirements and implementations
- Envelope of allowed change
  - "load-bearing walls"
  - design idioms and styles

ksw: This is a view specific style rather than architecture style.

* Source of the slide: David Garlan

Elements of a Run-time view Architecture Description

- A Run-time view of a system includes
  - Components: define the locus of computation
    - Examples: filters, databases, objects, ADTs
  - Connectors: define the interactions between components
    - Examples: procedure call, pipes, event announce

- A Run-time architectural style defines a family of architectures constrained by
  - Component/connector vocabulary (types)
  - Topological constraints
  - Semantic constraints

*** Source of the slide: David Garlan**

Examples

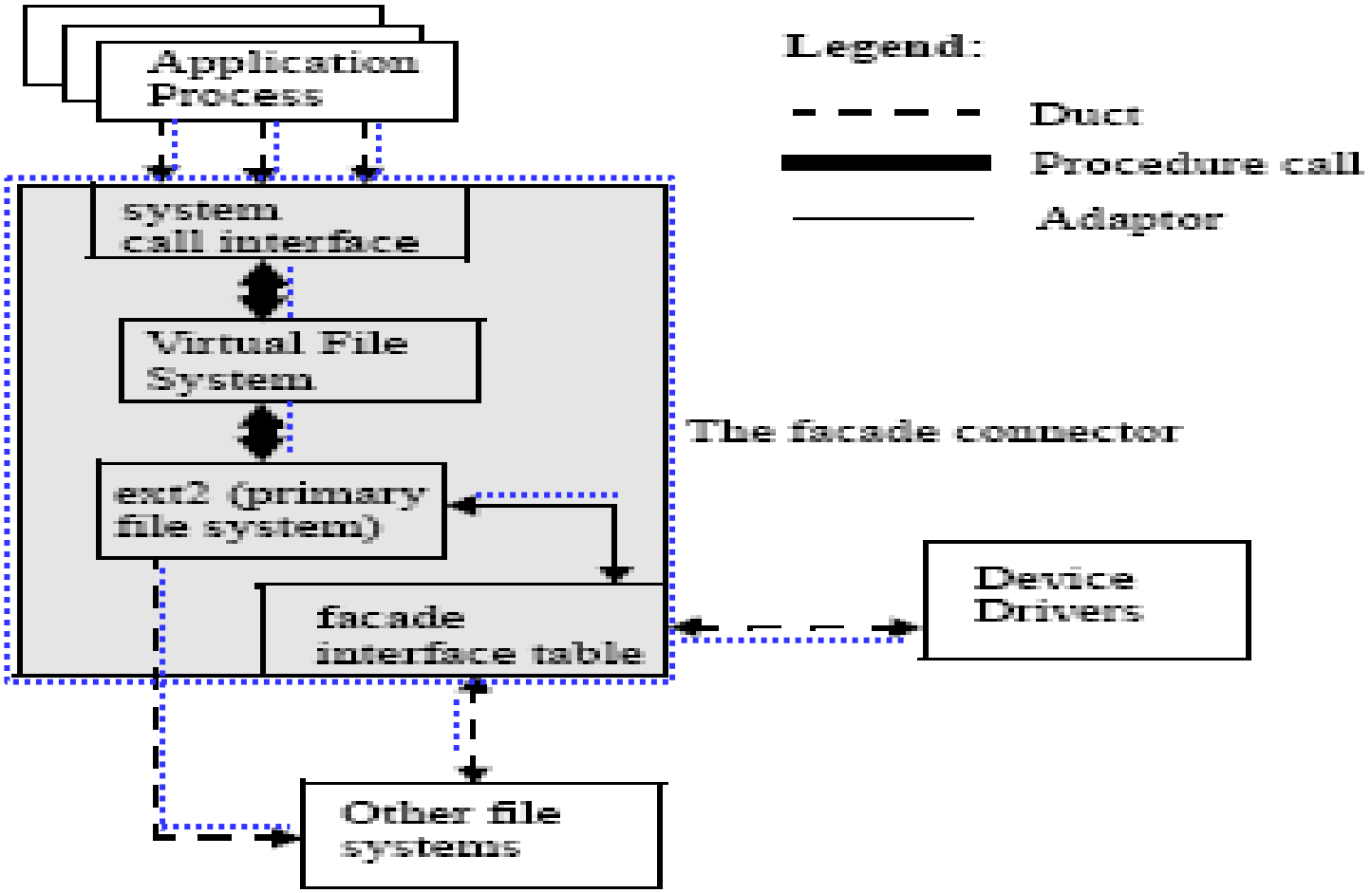


Figure 4. The Linux file facade connector.

Table 1: The file facade connector

Connector Type	Dimension	Value
Arbitrator	Authorization	Access Control Lists
Arbitrator	Isolation	Read/Write
Adaptor	Invocation conversion	Translation

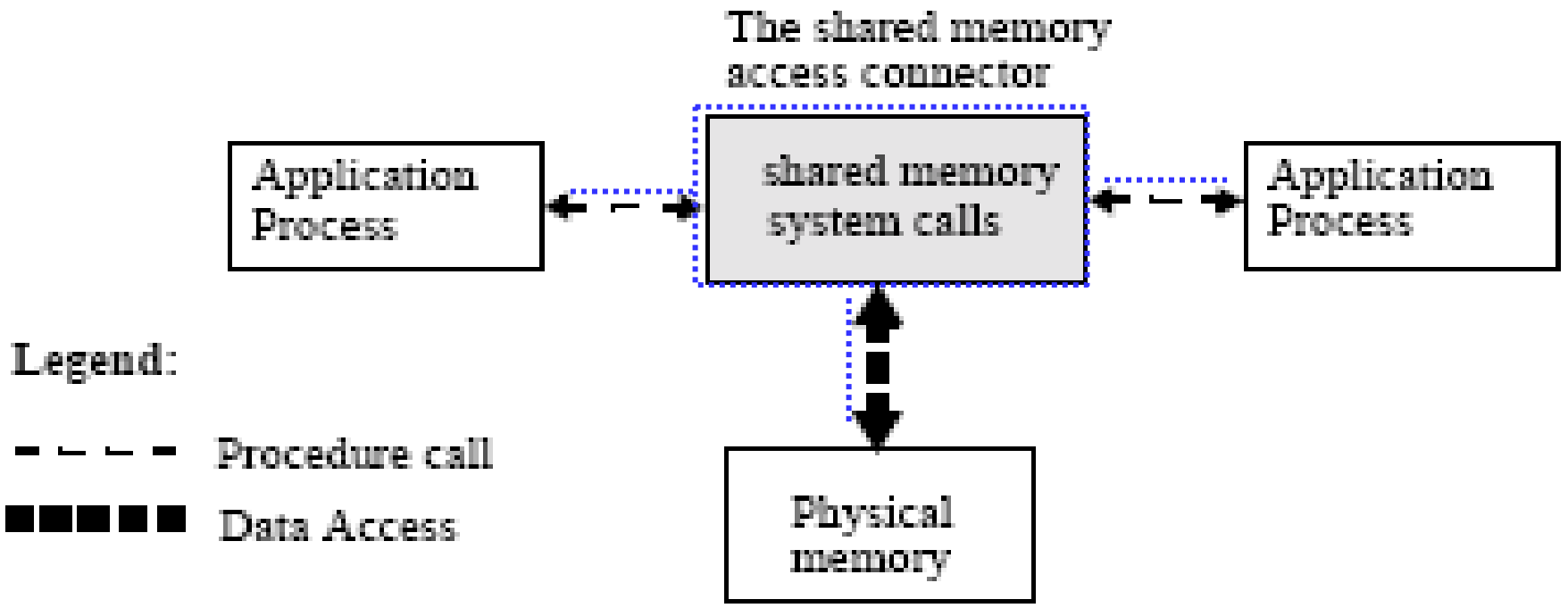


Figure 5. The shared memory access connector.

Table 2: Shared memory access connector

Connector Type	Dimension	Value
Data access	Locality	Global
Data access	Access	Accessor and Mutator
Data access	Transient availability	Heap
Data access	Accessibility	Protected
Data access	Cardinality	N defines; N uses

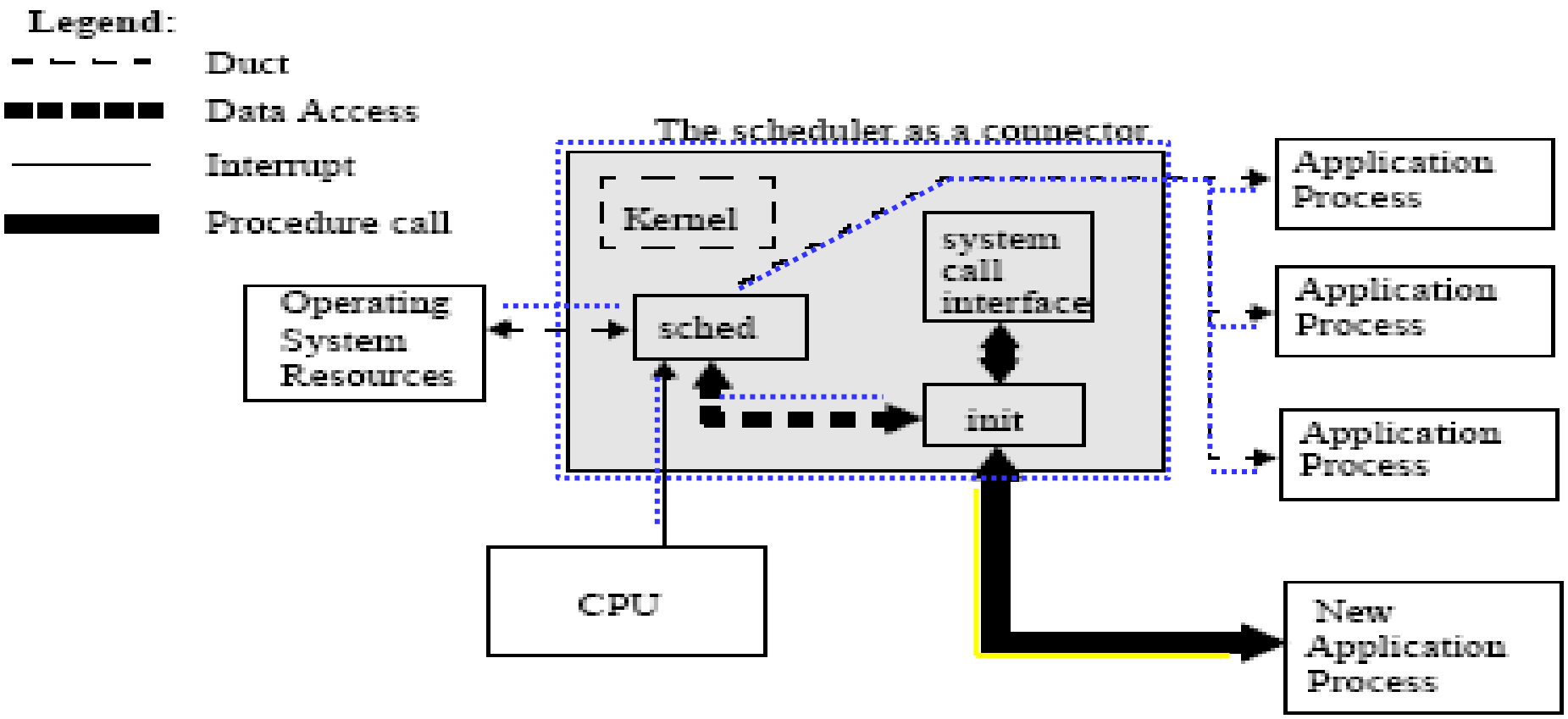


Figure 6. The process scheduler connector.

Table 3: The scheduler as a connector

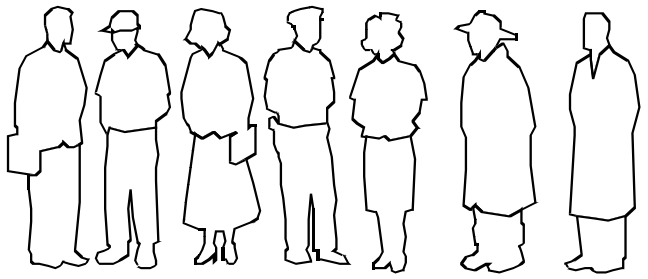
Connector Type	Dimension	Value
Arbitrator	Fault handling	Authoritative
Arbitrator	Concurrency weight	Heavy
Arbitrator	Authorization	Access control lists
Arbitrator	Scheduling	Time

6.2 관점체계

- Which viewpoints are relevant? It depends on:
  - who the stakeholders are
  - how they will use the documentation.
  - what kinds of systems we are building
    => What kind of **architecture style** has been selected (!!)

- Question: Who are the stakeholders of architecture documentation, and what information do they need?

"No less important is what development role the viewpoints plays" - ksw



* Source of the slide: David Garlan

Viewpoints Framework

- What viewpoints are there?
  - ⇒ Many!
  - ⇒ An architect needs to choose the useful ones.

- Viewpoints Framework: There are many proposals for collections of viewpoints for architectural design
  - Kruchten′s 4+1 Viewpoints: basis of RUP
    - Logical, Process, Development, Physical, Use cases
  - Siemens Four viewpoints
    - Conceptual(=Logical), Module, Code, Execution
  - ANSI/IEEE 1471-2000
    - More of a generic viewpoint framework

Some viewpoints – SEI and Garlan

1. How is it structured as a set of code units? Module views (module *viewpoint*)
   - Views in the *module viewpoint* show elements that are units of implementation.

2. How is it structured as a set of elements that have run-time behavior and interactions? Run-time views (*C&C viewpoint*)
   - Views in the *C&C view* show elements that have run-time behavior and interaction.
   - ksw: Our preferred term is "the runtime viewpoint". There is often, but not always, a correspondence between the two.

3. How does it relate to non-software structures in its environment? Allocation views (allocation *viewpoint*)
   - Views in the *allocation viewpoint* show how software structures are allocated to non-software structures.

The SEI Approach – Not Recommended

Viewpoint	Styles [Clements 11]
Module Viewpoint	Decomposition, Uses, Generalization Layered, Aspects, Data model
C&C Viewpoint	**Dataflow** Pipe-and-Filer **Call-Return** Client Server, Peer-to-peer, SOA **Event-based** Publish-subscribe **Repository** Shared-data
Allocation Viewpoint	Deployment, Install, Work Assignment Implementation, Data stores

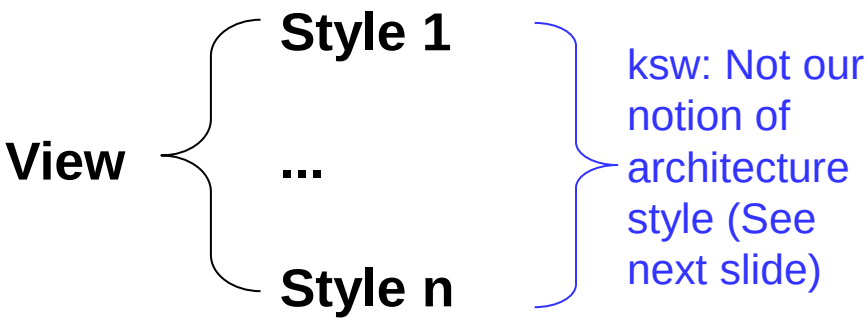
When people talk about "SOA", do they presuppose C&C view?
No! It is rather about the architecture itself.

All of these look like viewpoints.

- Relationship with viewpoints explicit.
- Some are styles but some are not
  - Some Module viewpoint styles and Allocation viewpoint styles look more like viewpoints.

Architecture Viewpoints and Views (SEI)

- A view is a representation of a set of system elements and the relations associated with them.
  - Not *all* system elements
    -- *some* of them.



- **A view selects element *types* and relation *types* of interest, and shows those.**

[Source: Garlan Lecture Notes]

ksw: "A view selects … *while selecting elements and relations of interest*"

ksw: "I would say that (architecture) style determines the viewpoints that are suitable for it."

참조 관점체계

	개념적 수준	논리적 수준	물리적 수준	배치수준
실행측면	사용시나리오관점 (Scenario Viewpoint) ->시나리오뷰	논리적 관점 (Logical viewpoint) ->논리뷰 혹은 논리적 아키텍처	실행관점 (Execution Viewpoint) ->실행뷰 혹은 실행아키텍처	물리적 실행요소의 배치뷰 (Deployment of Runtime Elements Viewpoint) ->실행요소 배치뷰
코드측면	기능관점 (Functional Viewpoint) 혹은 개념적 관점 (Conceptual viewpoint) ->개념뷰 혹은 개념적 아키텍처	모듈관점 (Module Viewpoint) ->모듈뷰 혹은 모듈아키텍처	코드관점 (Code Viewpoint) ->코드뷰	코드요소의 배치뷰 (Deployment of Code Elements Viewpoint) ->코드배치뷰

More abstract ← → **More concrete**

대상시스템에 따라 적절한 관점체계가 달라진다. 그러나 위와 같은 측면과 수준에 상응하는 관점이 필수적이다. 왜냐하면 개념적 수준은 요구사항과 연결을 주고 물리적 수준은 구현과 연결을 주기 때문이다.

표 6-2. 아키텍처 관점의 정의

아키텍처 관점		관심사항
실행 측면	시나리오관점[55]	아키텍처 드라이버인 시나리오로 구성되며, 시스템의 아키텍처가 구체화되면서 시스템과 액터들의 상호작용으로부터 내부 컴포넌트들의 상호작용 시나리오로 구체화되어 다른 뷰들을 형성하고 연결(align)하는 역할을 수행한다.
	논리적 관점	시나리오 관점과 개념적 관점이 사용자가 이해할 수 있는 비기술적(non-technical) 아키텍처 뷰라면, 논리적 관점은 개발을 위하여 필요한 기술적(technical) 아키텍처 관점으로 기술적 아키텍처 관점 가운데 첫 번째로 만들어지는 관점이다. 논리적 관점의 초점은 높은 추상 수준에서 아키텍처 설계 문제를 일차적으로 해결하는 것이다. 이후 논리적 아키텍처는 이어지는 관점들에 의하여 수정될 수 있다.
	실행관점	시스템 실행시의 개체와 그들의 관계 및 데이터흐름을 결정한다.[56] 또한 실행시 개체들이 어떻게 상호 통신(communication) 및 조절(coordination)을 하는지를 결정한다.
	실행요소배치관점	실행요소들을 노드들과 소프트웨어 플랫폼으로 배치한다.
코드 측면	개념적 관점	시스템을 구현 가능하고 재사용성이 높은 서브시스템들로 분할하고 기능적 요구사항을 컴포넌트로 분배한다.
	모듈관점	시스템을 모듈로 분할하고 계층구조로 만든다. 모듈뷰에 나타나는 모듈들 간의 관계는, 합성(decomposition), 의존(dependency), 그리고 일반화(generalization) 관계이다. 개발조직을 위한 개발책임할당의 기반이 된다.[57]
	코드관점	객체코드, 라이브러리(libraries), 바이너리(binaries) 등의 소스코드를 버전, 파일, 디렉터리에 따라 배치 컴포넌트로 정렬(organize)한다.
	코드배치관점	코드 관점에서 도출된 배치컴포넌트들의 파일을 물리적인 시스템 노드들과 소프트웨어 플랫폼으로 배치한다.

6.3 아키텍처 뷰의 개발순기에 있어서의 역할

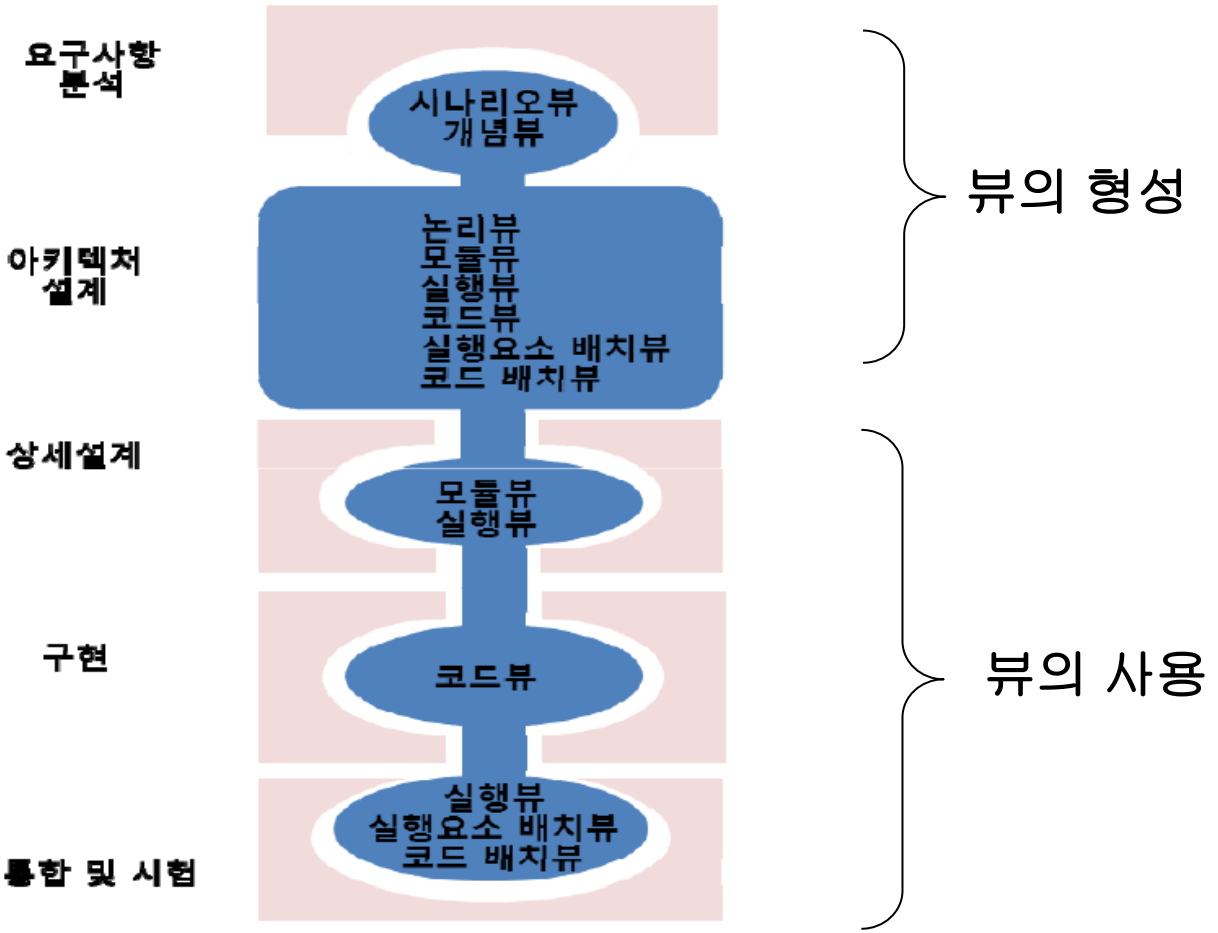


그림 6-2. 개발순기에서의 아키텍처 뷰의 역할

6.4 아키텍처 드라이버와 아키텍처 관점들

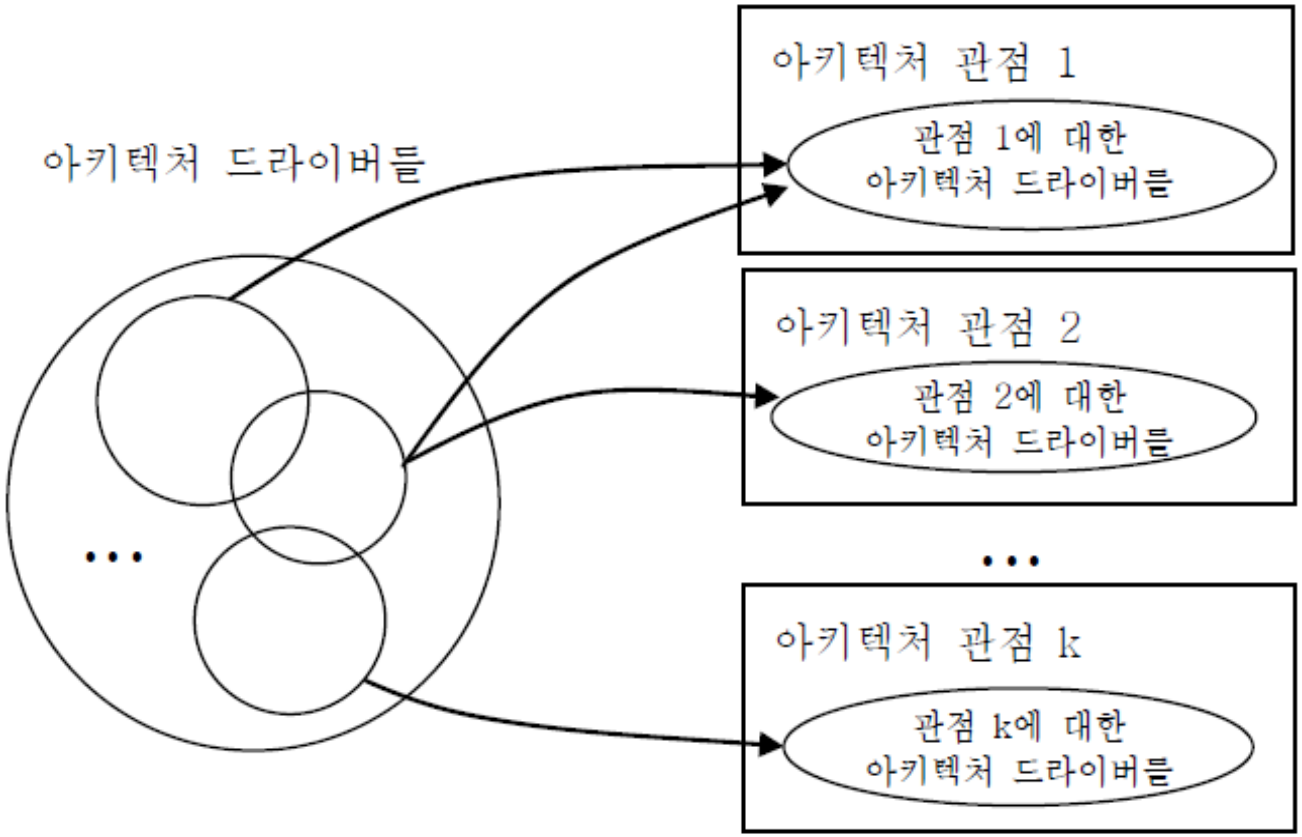


그림 6-3. 아키텍처 관점에 따른 아키텍처 드라이버들의 분리

표 6-3. 관점별 관심 품질속성

아키텍처 관점		관련 품질속성		
		비공통	공통	
실행측면	시나리오 관점		성능 (Performance) 신뢰성 (Reliability) 가용성 (Availability) 사용-용이성 (Usability) 확장성 (Scalability)	이해성 (Understandability)
	논리적 관점	안전성 (safety) 보안성 (Security) 시험가능성 (Testability)		
	실행 관점			
	실행요소 배치관점			
코드측면	개념관점		구현성 (Buildability) 재사용성 (Reusability) 통합가능성 (Integratability) 시험가능성 (Testability)	
	모듈관점	확장성 (Scalability) 변경용이성 (Modifiability)		
	코드 관점	이식성 (Portability)		
	코드 배치관점			

아키텍처의 **core**가 모듈관점과 논리관점임을 보여줌

6.5 요약

- Styles의 경우와 마찬가지로 Viewpoint도, solution을 만들어가는 접근방법이 아니라 solution의 표현 방법이다.
- 따라서 Styles이나 Viewpoints의 결정은 넓은 의미의 설계절차에 포함되고, 좁은 의미의 설계절차에는 포함되지 않는다.
- (큰 시스템의 경우, 설계가 recursive 하게 진행될 경우, 새로운 recursion level의 시작점에서 그 subsystems에 맞는 Styles과 Viewpoints의 결정이 새로이 이루어질 수도 있다.)

Questions?