Software Architecture Theory

P05. *Architecture Styles*

2014

Sungwon Kang

5. 아키텍처 스타일

5.1 정의 및 기술방법

5.2 아키텍처 스타일의 종류

5.3 아키텍처 스타일의 적용

Note that “style” precedes “viewpoints.”

“Style” can be thought of as “solution style”.

Architecture Styles

1. Introduction to Architectural Styles
2. Simple Styles
3. Complex Styles

Acknowledgment Much of the following contents is based on [Taylor 09]

Various Styles of Architecture

(Source: http://www.greatbuildings.com/)

Introduction to Architectural Style [Taylor 09]

- For multiple system users in a distributed setting, the following set of design choices ensures effective provisioning of services:
  1. Physically separate the software components used to request services from the components that provide the needed services
  2. Make the service providers unaware of the requester' identity
  3. Insulate the requesters from one another to allow for their independent addition, removal, and modification. Make the requesters dependent only on the service providers.
  4. Allow for multiple service providers to emerge dynamically to off-load the existing providers

What does these decisions define?

=> Not a particular system

=> Not specify the components (or their types), nor their interaction mechanisms

=> The architect should elaborate further to turn them into application-specific decisions.

Definition of Architectural Style [Taylor 09]

- **Definition**: An *architectural style* is a named collection of architectural design decisions that

  (1) are applicable in a given development context,

  (2) constrain architectural design decisions that are specific to a particular system within that context, and

  (3) elicit beneficial qualities in each resulting system

- **정의 (아키텍처 스타일).** 아키텍처 스타일은 컴포넌트와 커넥터들의 어휘를 제공하고 그들의 결합되는 방식에 대한 제약을 정의한다. [Shaw 96]

아키텍처 스타일의 표현

아키텍처 스타일 이름		스타일을 지칭하는 의미 있는 이름
설계요소타입	컴포넌트	컴포넌트 타입 및 실행 방법
	커넥터	커넥터 타입 및 실행 방법
제약사항		이 컴포넌트와 커넥터들이 결합되는 방식에 대한 제약과 대표적인 구조적 형태
적용예제		스타일의 적용 예제들
장단점		스타일의 장점과 단점

- **Reflect less domain knowledge than architectural patterns**
  => Has wider impact

- However, the boundaries between style and pattern are not precise [Taylor 09]

Ksw: They can be clearly distinguished because:

  - Without C&C, architecture will be chaos.
  - Without patterns, we have to work hard to solve problems.
  - Without styles, we lose big shortcuts.

5.2 아키텍처 스타일의 종류

Taxonomy of Styles [Taylor 09]

Simple Styles		Complex Styles
Traditional	**Shared Memory**	- C2
Language-Influenced Styles	- Blackboard	- Distributes Objects
- Main program and subroutines	- Rule-based	
- Object-oriented	**Interpreter**	
Layered	-Interpreter	
- Virtual machines	- Mobile code	
- Client-server	**Implicit Invocation**	
Dataflow Styles	- Publish-subscribe	
- Batch-sequential	- Event-based	
- Pipe-and-Filter	**Peer-to-Peer**	

- “Reflect less domain knowledge than architectural patterns”
  => More broadly applicable “
- ksw: For a specific system, we may use a specific pattern of Pipe-and Filter style.

Green: was classified also as pattern in [Buschmann 96]

5.3 아키텍처 스타일의 적용

Style Example 1 - Pipes and Filters Style

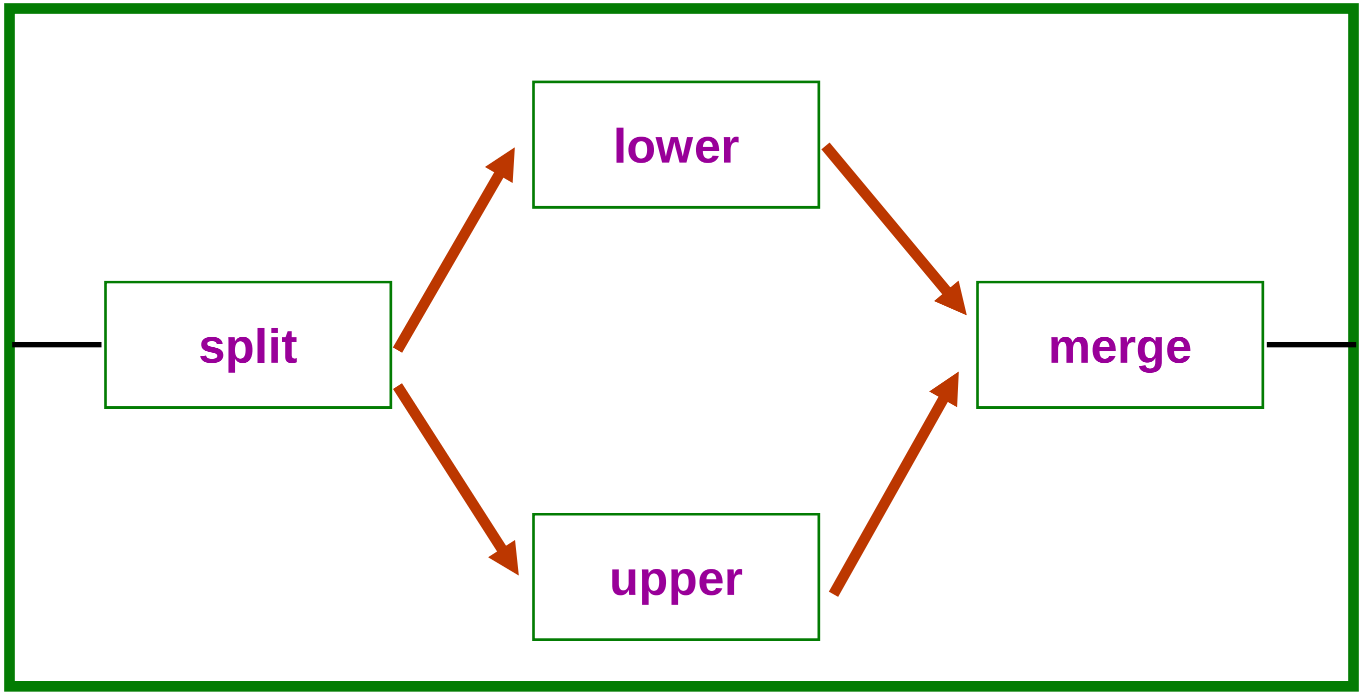
Style Example 2 – [Talyor 09]

Pipes and Filters Style

- Filter
  - – Incrementally transform some amount of the data at inputs to data at outputs
    - Stream-to-stream transformations
  - – Use little local context in processing stream
  - – Preserve no state between instantiations
- Pipe
  - – Move data from a filter output to a filter input
  - – Pipes form data transmission graphs
- Overall Computation
  - – Run pipes and filters (non-deterministically) until no more computations are possible.

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

Alternating Characters Code

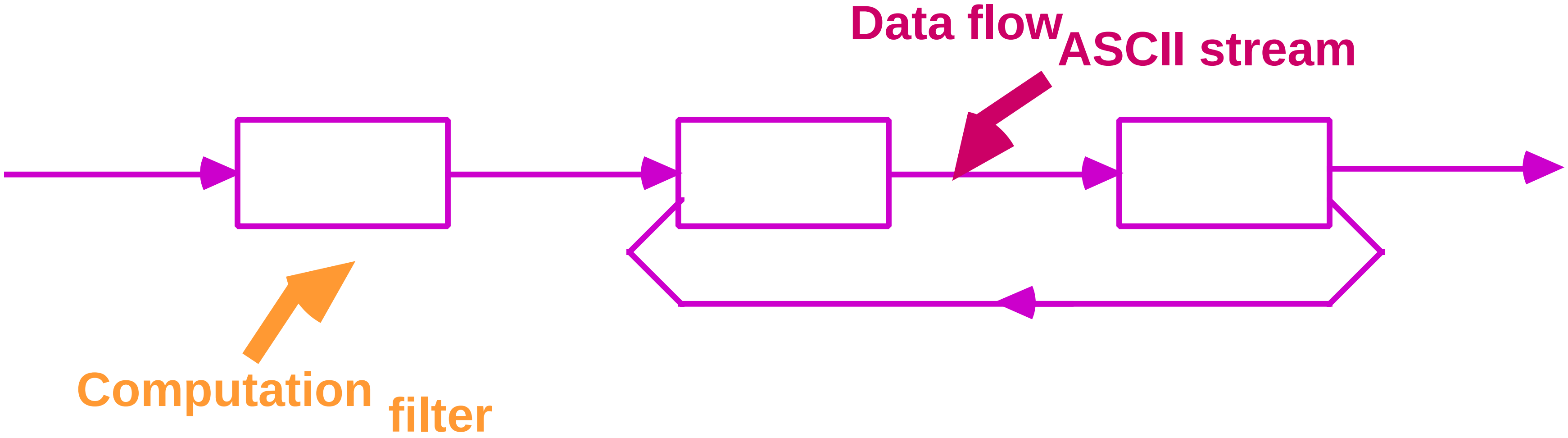


Legend

Filter	🟩
Pipe	→
Binding	—

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

Another Architecture



* Source of the slide:
David Garlan

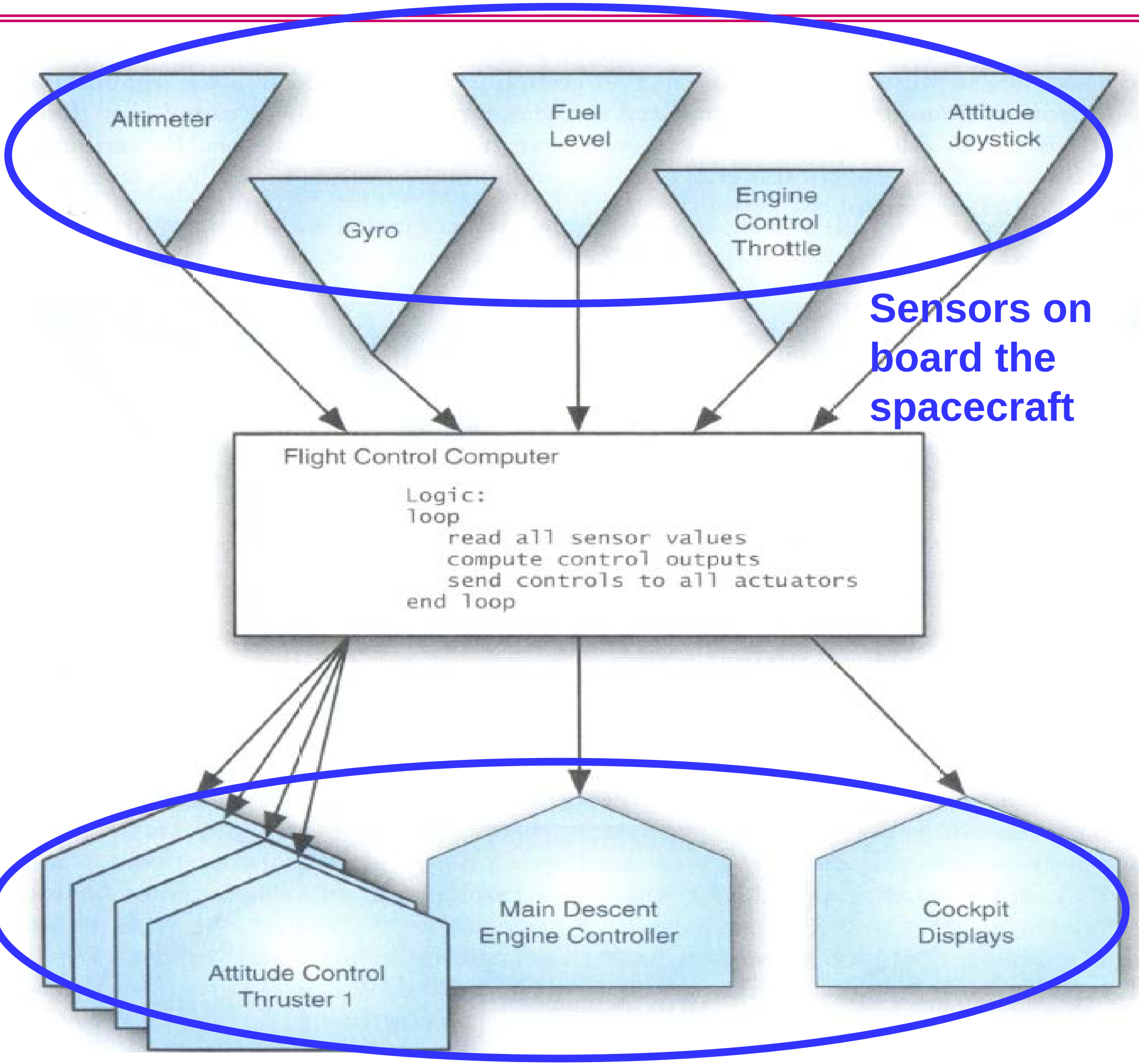
Style Example 2
[Taylor 09]

Lunar Lander

- A simple flight control application – software controlling “Lunar Lander”- a notional lunar excursion module to the surface of the moon

- “Sense-Compute- Control (SCC)” pattern used

Style may be more suitable



```
Logic:
loop
    read all sensor values
    compute control outputs
    send controls to all actuators
end loop
```

Figure 4-6. *Sense-compute-control applied to a lunar lander.*

Sensors on board the spacecraft

Actuators on board the spacecraft

Lunar Lander

As the craft approaches the surface . . .

- The **sensors** provide to the computer
  - information on the altitude, latitude, fuel remaining and the pilot's control inputs
- The **flight control computer**
  - processes its control laws and values to be sent to the various actuators
    - to control the descent engine and the attitude control thrusters
    - to update the cockpit displays.
  - determine, for example, from changes in the altitude the descent rate
  - display that to the pilot
- **Pilot** increases or decreases the throttle to slow or increase the descent rate.

  ☛ Strategy applied to Apollo landings

Lunar Lander

- A fully automated landing strategy
  => Software determines the optimal settings

☛ From architectural point of view, these two strategies (i.e. pilot-operated landing vs. fully automated landing) are the same
=> At each cycle of the loop, the software reads the current sensor values, computes the desired settings for the actuators and send those value to the actuators.

Not exactly as they have different sets of machine and domains.

Game version of Lunar Lander

- Different from the SCC version
  - The physics of the universe needs to be simulated by software (rather than by the moon environment)
  - In a real flight control application, the actions of the physical universe affect the space craft

- Game version is used in the subsequent examples.

- Lunar Lander in two styles:
  - Pipe-and-filer
  - C2 style

Lunar Lander in the Pipe-and-Filter Architecture Style

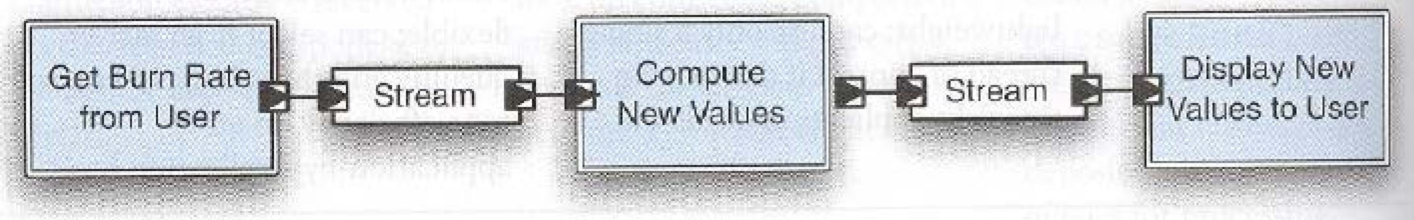


Figure 9-4.
Lunar Lander in the pipe-and-filter architectural style.

What viewtype (= viewpoint) does the above show?

Implementing Lunar Lander in Pipe-and-Filter Architecture Style

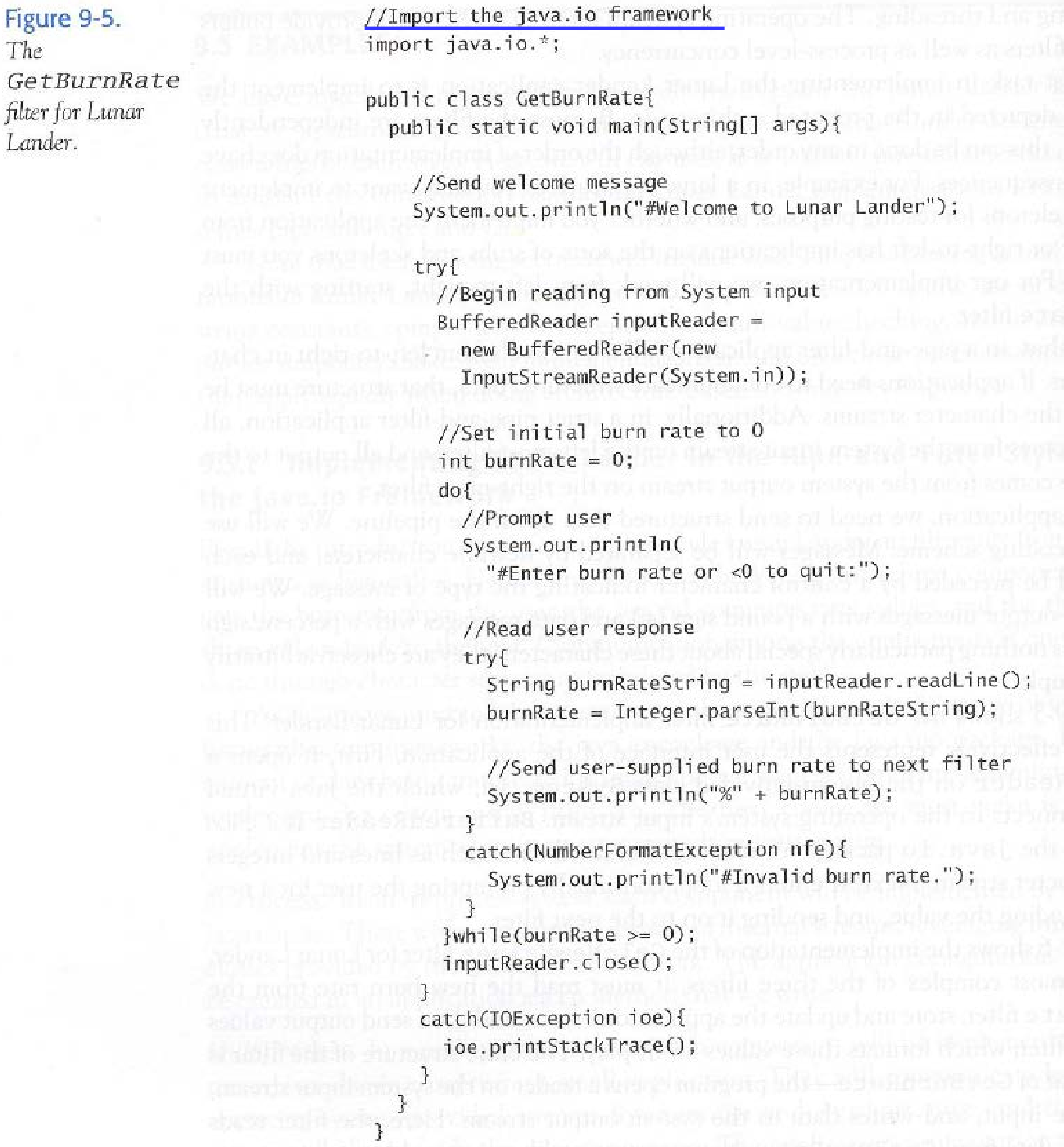


Figure 9-5. The `GetBurnRate` filter for Lunar Lander.

```
//Import the java.io framework
import java.io.*;

public class GetBurnRate{
  public static void main(String[] args){

    //Send welcome message
    System.out.println("#Welcome to Lunar Lander");

    try{
      //Begin reading from System input
      BufferedReader inputReader =
        new BufferedReader(new
        InputStreamReader(System.in));

      //Set initial burn rate to 0
      int burnRate = 0;
      do{
        //Prompt user
        System.out.println(
          "#Enter burn rate or <0 to quit:");

        //Read user response
        try{
          String burnRateString = inputReader.readLine();
          burnRate = Integer.parseInt(burnRateString);

          //Send user-supplied burn rate to next filter
          System.out.println("%" + burnRate);
        }
        catch(NumberFormatException nfe){
          System.out.println("#Invalid burn rate.");
        }
      }while(burnRate >= 0);
      inputReader.close();
    }
    catch(IOException ioe){
      ioe.printStackTrace();
    }
  }
}
```

Lunar Lander in the C2 Architecture Style

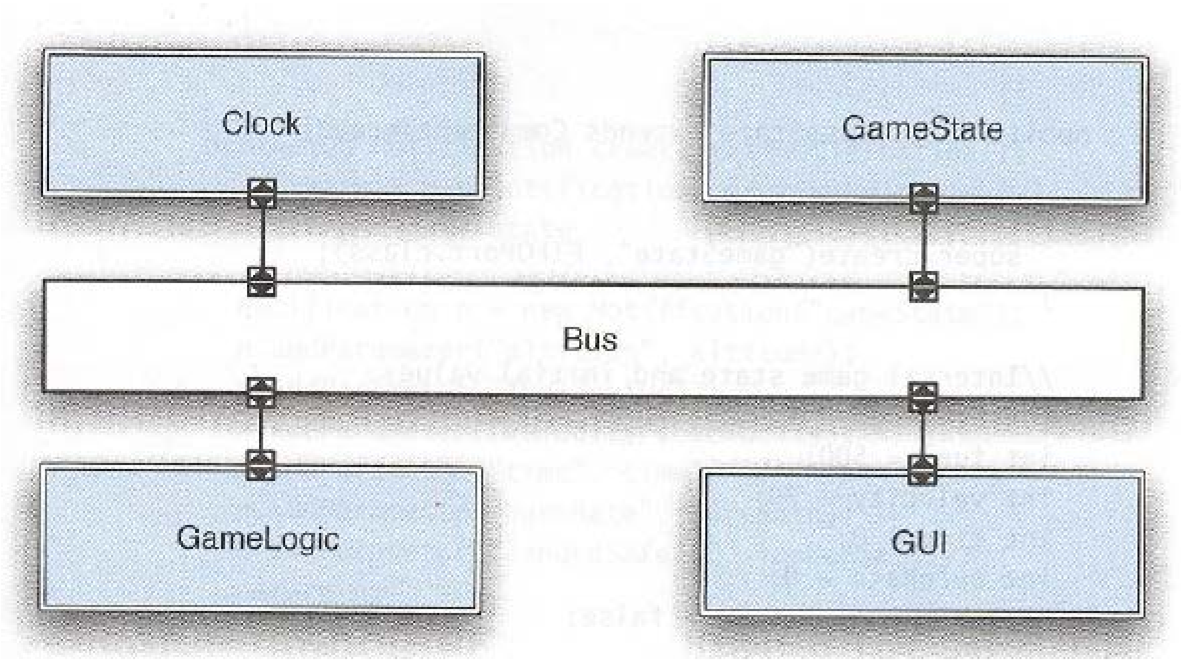


Figure 9-8. *Lunar Lander in the C2 architectural style.*

```
import c2.framework.*;
public class GameState extends ComponentThread
{    public GameState()
     { ...
      }
     ...
```

Domain to Architecture Styles Mapping 의 활용

Application Domain	Subdomain	Architecture Styles
Embedded Software		SCC
Distributed Application		CS, Peer-to-Peer
GUI		C2, Event-based
X	X1	SX11, SX12, …
	X2	SX21, SX22, …
	…	…
…		…

Classification in [Taylor 09] (Cf. "5. Architecture Styles") is a grouping of architecture styles.

Lab 2. 아키텍처 문제 분석

- **아키텍처 문서의 C를 작성**

Step 1. 아키텍처 드라이버 선정

Step 2. 이슈도출

- 여러 개의 아키텍처 드라이버가 하나의 이슈 제기 가능
- 이슈화되지 않은 선정된 아키텍처 드라이버가 없어야 함

Step 3. 이슈별 아키텍처 전략들 도출

아키텍처 문제 분석표

설계문제	요구사항	설계전략	이유 (이득/비용/위험/불확실성 포함)	관련사항 (관련전략/파급효과/완화전략)
Issue 6. 피처의 용이한 추가와 제거.	QR02. 출시시간이 짧다. C03. 피처개발이 협상가능하다. C05. 새로운 종류의 피처가 매 3년마다 추가될 수 있다. C09. 사용자 상호작용 모델이 새로운 패러다임과 표준에 맞게 수정되어야 한다.	AS10. 컴포넌트와 모듈을 관심(concern)에 따라 분리시킨다.	DR20. 기존의 프레임워크를 사용하여 빠른 시간에 새로운 솔루션을 얻기 용이. - . . .	범용컴퓨터와 도메인특정 하드웨어 이슈 참조.
		AS11. 피처를 분리된 컴포넌트들로 캡슐화시킨다.		
		AS12. 사용자 상호작용 모델을 분리시킨다.		
. . .	. . .	. . .	. . .	. . .

QR: Quality Requirement, C: Constraint, AS: Architecture Strategy, DR: Design Rational

Questions?