P03. *Architectural Problem Analysis*

2014

Sungwon Kang

3. 아키텍처 문제 분석

3.1 아키텍처 문제 분석의 정의

3.2 아키텍처 문제 분석 원리

3.3. 아키텍처 문제 분석 기법

3.4 문제틀 분석방법

3.1 아키텍처 문제 분석(Architectural Problem Analysis)의 개념

- **정의**. 아키텍처 문제의 식별, 정리 및 해결책의 후보를 도출하는 활동.
- 문헌에서 아키텍처적 분석(Architectural Analysis) 라고 부른 활동에 가깝다. 그러나 다음 두 가지 이유로 아키텍처문제분석이라고 부른다.
  - 요구사항 분석과의 용어 혼란 가능성
  - 분석의 결과가 아키텍처 설계 해법에 가져올 수 있는 적극적인 분석(proactive analysis)를 수행해야 한다
- [Larman 02]p32
  - "아키텍처적 분석은 시스템의 비기능적 (예를 들어 품질) 요구사항들을 기능적 요구사항의 문맥 속에서 식별하고 해결한다"라고 정의.
- UP의 Architectural Analysis 정의 [Jacobson 99]p196
  - "아키텍처적 분석의 목적은 분석 패키지, 자명한 분석클래스 및 공통의 특수한 요구사항들을 식별함으로써 분석모델과 아키텍처를 요약하는 것이다."
  - [Larman 02]에 비하여 아키텍처적 분석의 역할이 약함.
- UP와 [Larman 02] 의 방법 모두 아키텍처 설계의 역할이 분석/설계에 보조적

- [Larman 02] p493
  - “아키텍처의 과학은 아키텍처 드라이버에 관한 정보를 수집하고 정리하는 것이고, 아키텍처의 예술은 상충관계, 상호의존성, 우선순위에 따라 이것들을 해결하는 솜씨 있는 선택을 하는 것이다.”

=> 아키텍처 설계는 아키텍처 예술이고, 아키텍처 문제 분석은 이 두 영역의 중간에 있는 활동이다.

3.2 아키텍처 문제 분석원리

1절의 아키텍처 드라이버로 선별된 요구사항 중,

(1) 중요성이 높은 드라이버를 추출한다.

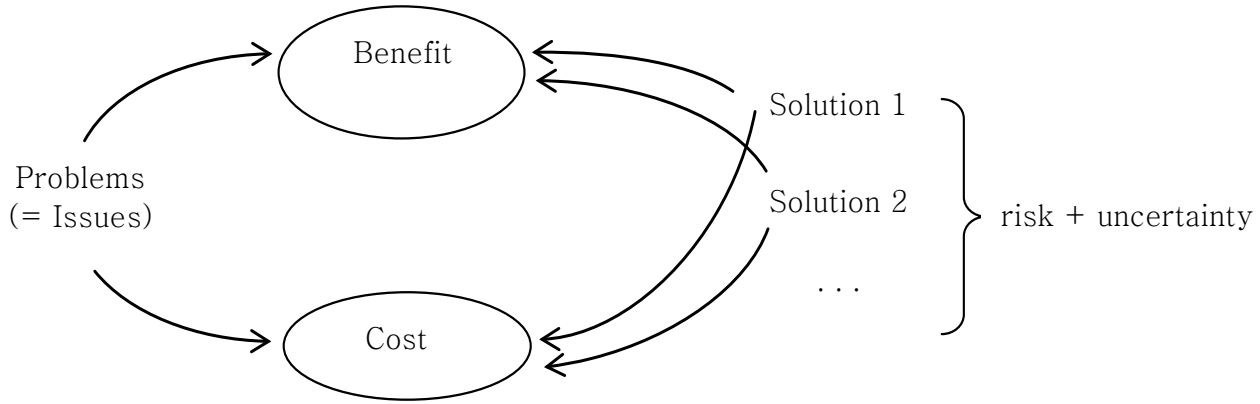
(2) 이들에 대한 가능한 **해법을 도출한다.**

(3) 이들 해법들을 평가하여 그 결과를 정리한다.

초보적 평가방법

문제와 해법의 우선순위를 결정함에 있어서 직관적인 방법에 기반한다.

진보된 평가방법



3.3 아키텍처 문제 분석기법

- **Technical Memo [Cunningham 96]**
- **Issue Cards [Hofmeister 00]**
- **SEI Architectural Decisions Document [Clements 11]p239-241**
- 아키텍처 문제 분석표

Source: [Larman 02]p495

Technical Memo

Issue: Legal—Tax Rule Compliance

Solution Summary: Purchase a tax calculator component.

Factors

- Current tax rules must be applied, by law.

Solution

Purchase a tax calculator with a licensing agreement to receive ongoing tax rule updates. Note that different calculators may be used at different installations.

Motivation

Time-to-market, correctness, low maintenance requirements, and happy developers (see alternatives). These products are costly, which affects our cost-containment and product pricing business goals, but the alternative is considered unacceptable.

Unresolved Issues

What are the leading products and their qualities?

Alternatives Considered

Build one by the NextGen team? It is estimated to take too long, be error prone, and create an ongoing costly and uninteresting (to the company's developers) maintenance responsibility, which affects the goal of "happy developers" (surely, the most important goal of all).

Issue 1

Source: [Hofmeister 00]

Aggressive Schedule

The development schedule is aggressive. Given the estimated effort and available resources, it may not be possible to develop all the software in the required time.

Influencing Factors

O4.1: Time-to-market is short and is not negotiable. A rough estimate of effort required to redesign and reimplement all of the software suggests that it will take longer than two years.

O5.1: Head count cannot be increased substantially.

O1.1: Building is mildly preferred over buying.

O4.2: Delivery of features is negotiable. Low-priority features can be added to later releases.

P7.2: Budget for commercial off-the-shelf (COTS) components is flexible. Both the price and the licensing fees of COTS components must be considered.

Solution

Redesigning and reimplementing all of the software will take longer than two years. Three possible strategies are to reuse software, buy COTS, and to release low-priority features at a later stage.

Strategy: *Reuse existing in-house, domain-specific components.* S1A

Several of the in-house domain-specific components are candidates for reuse. However, reuse of some existing components may need substantial redesign and reimplementation. Evaluate each of these components to determine whether it is advantageous to reuse it and whether it will save time and effort.

Strategy: *Buy rather than build.* S1B

Buying COTS software has the potential of saving time and effort. However, the price and licensing fees for some COTS products may be too high. Learning to use new COTS software may increase time and effort. Purchase or license COTS software when it is advantageous and when it will reduce development time substantially.

Strategy: *Make it easy to add or remove features.* S1C

One way to reduce the develop time is to reduce the functionality by delaying delivery of some of the features to a later release. If it is easy to add or to remove functional features without substantial reimplementation, then it is feasible to adjust the functionality to meet the delivery schedule.

SEI Architectural Decisions Document

1. Issue
2. Decision
3. Status
4. Group
5. Assumptions
6. Alternatives
7. Argument
8. Implications
9. Related Decisions
10. Related Requirements
11. Affected Artifacts

☛ Mixes analysis and decision

아키텍처 문제 분석표

표 3-2. 아키텍처 문제 분석표

설계문제	요구사항	설계전략	이유 (이득/비용/위험요인/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.		
. . .	. . .	. . .	. . .	. . .

Note) 기능요구사항은 분석표에 나타나지 않음

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

3.4 문제틀 분석방법

- Michael Jackson teaches us that we should be organized and start early to do architecture design with Problem Frames.

Definition 1. *Problem Frames* are a conceptual language for recognizing familiar, tractable problems in the client's requirements

Definition 2. A *problem frame* is a description of a recognizable class of problems. In a sense, problem frames are *problem patterns*.

Problem Frames

- The idea
  - Many types of software problems are already solved
  - If you stand back from the details of your current problem, you may be able to find a known problem
  - Known problems correspond to particular patterns in the problem frame

- Applying the idea
  - Context: scope the problem setting in terms of domains
  - Problem: state the requirement in the same terms
  - Look for structural similarities to known problems

[Shaw 05]

Context and problem

- First, locate the problem
  - Domains – observable parts of the world
    - Given domains – relevant parts of the world
    - Designed domains – physical representation of information
    - Machine domain – where you will build the solution
- Then, identify the interfaces – shared phenomena
- Then, capture relations in a context diagram
  - One machine, phenomena on the interfaces
- Next, discover what is the problem
  - Requirement is condition on one or more domains that the machine must bring about
  - Add to context diagram to get problem diagram

[Shaw 05]

Terminology

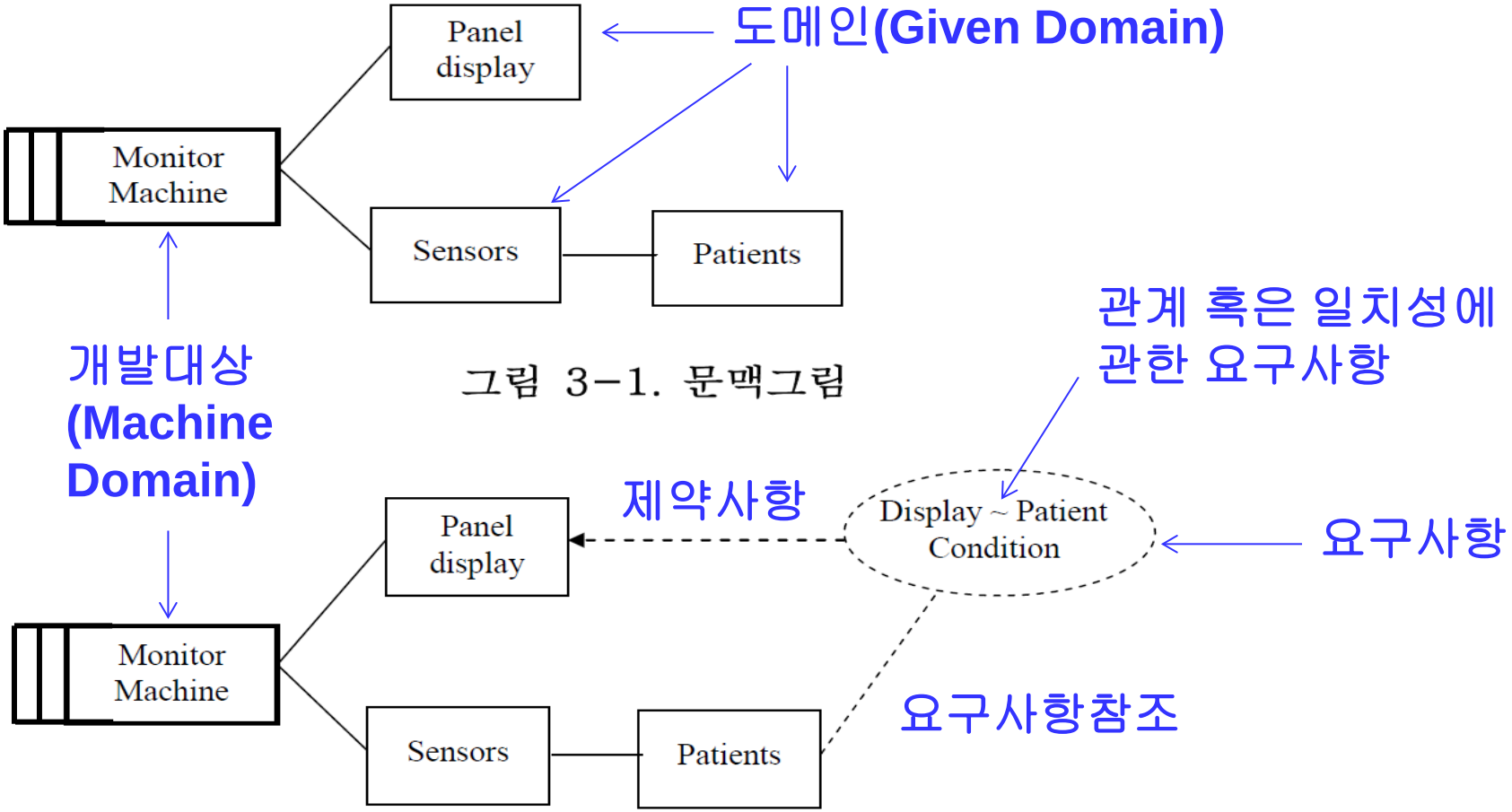


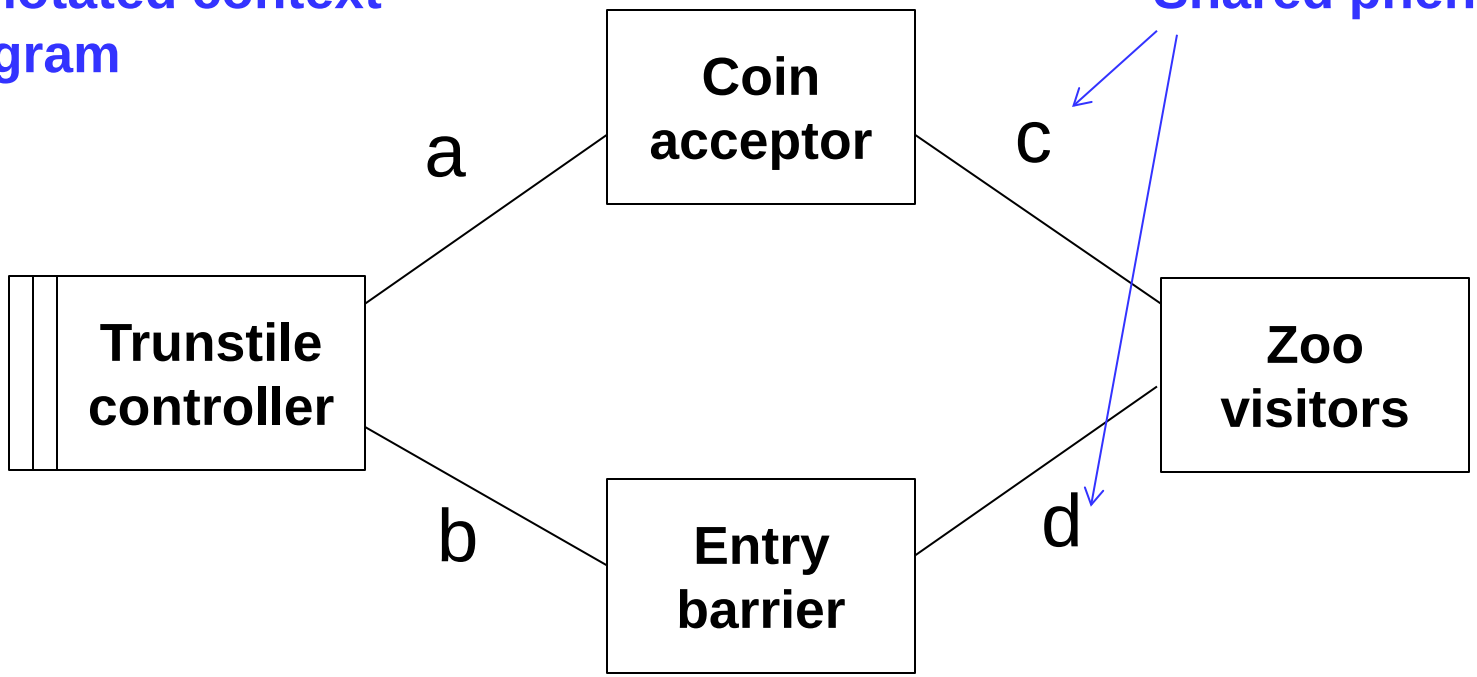
그림 3-1. 문맥그림

그림 3-2. 문제그림

Terminology

Annotated context diagram

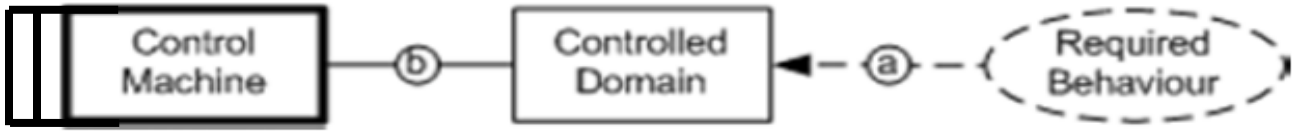
Shared phenomena



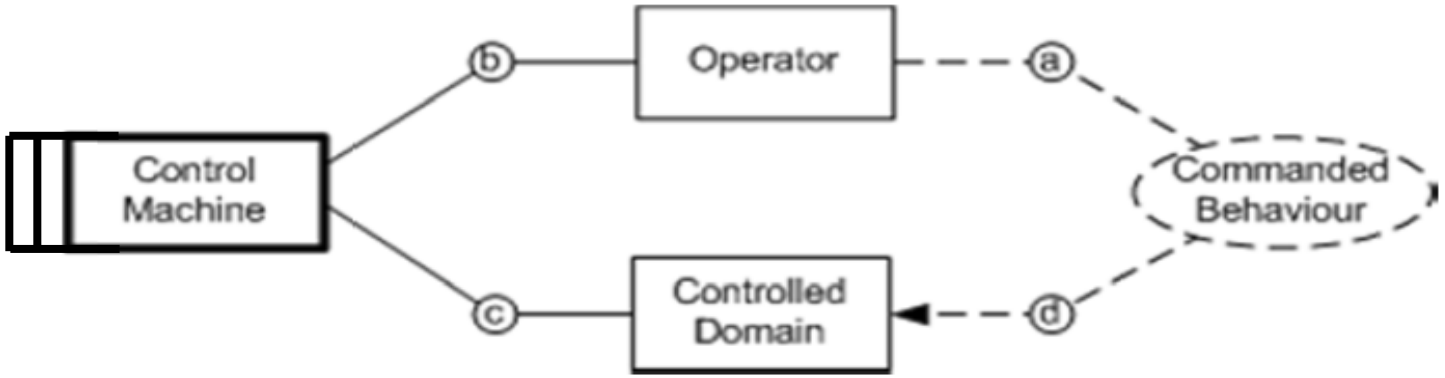
a: Acceptcoin
b: Lock, Unlock, Position

c: InsertCoin
d: Push, Enter

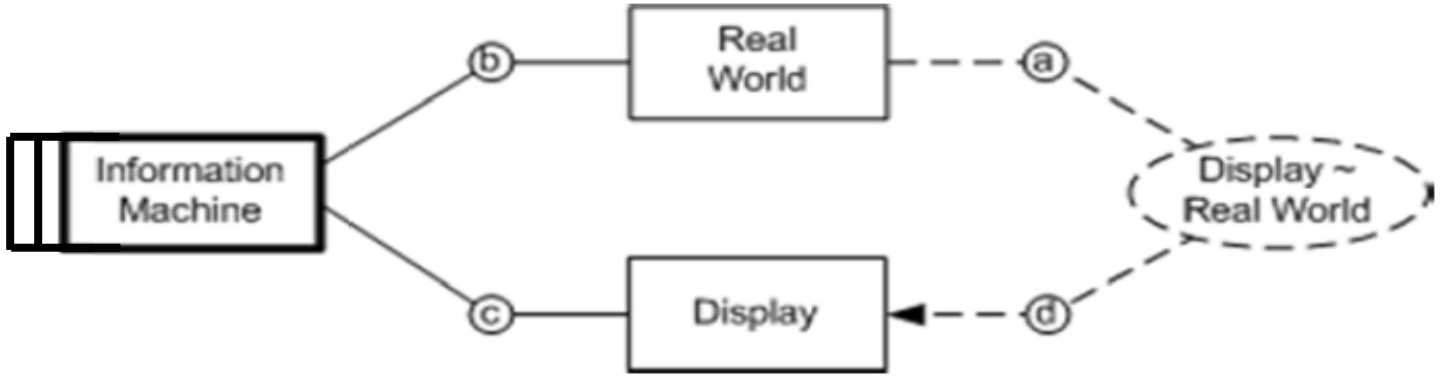
기본적인 문제틀 (Basic Problem Frames) [Jackson 01]



(a) 요구된 행위(Required Behavior) 문제틀



(b)지시된 행위(Commanded Behavior) 문제틀



(c)정보디스플레이(Information Display) 문제틀

Need behavior:

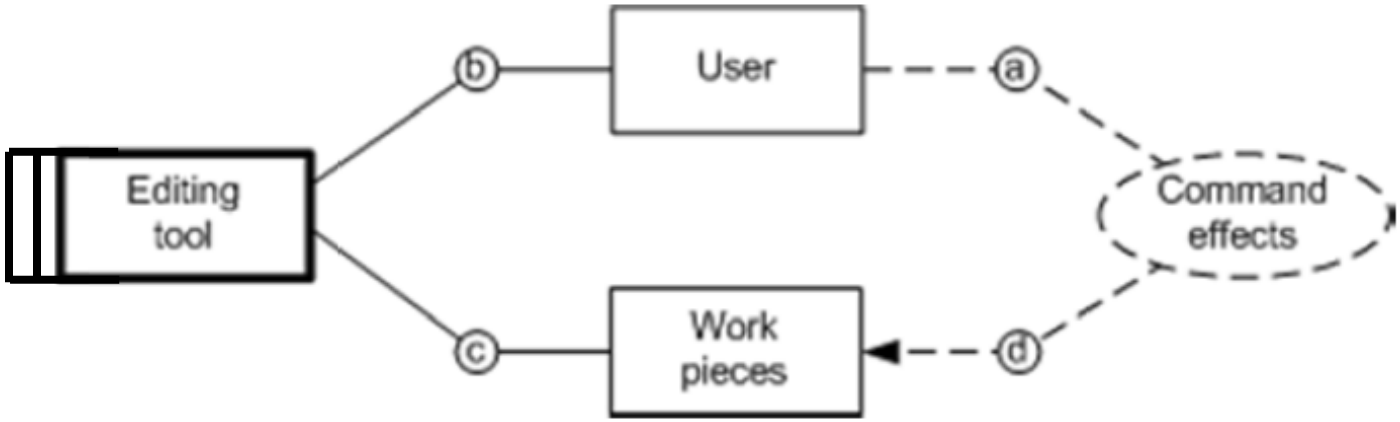
Control part of the physical world to satisfy a condition.

Need enforcement:

Control part of the physical world according to operator instructions.

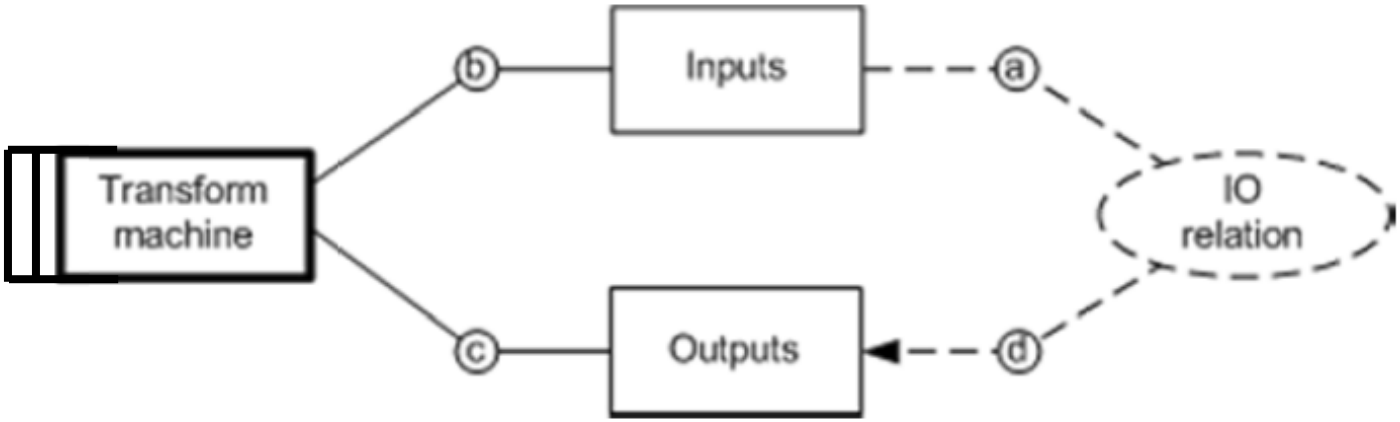
Need to see information:

Obtain state/behavior information from the physical world and present it as required



(d) 단순작업(Simple Workpieces) 문제틀

Need a helping tool:
Build a tool to create & edit persistent information objects



(e) 변형(Transformation) 문제틀

Need processing:
Transform information inputs to required outputs

그림 3 -3. 기본적 문제틀의 그림

One way traffic lights [Jackson 01]

One way traffic lights [Jackson 01]

- Each unit has s Stop light and a Go light.
- The computer controls the lights by emitting RPulses and GPulses to which the units respond by turning the lights on and off.
- The regime for the light repeats a fixed cycle of four phases:

  (1) For 50 seconds, both units show Stop,

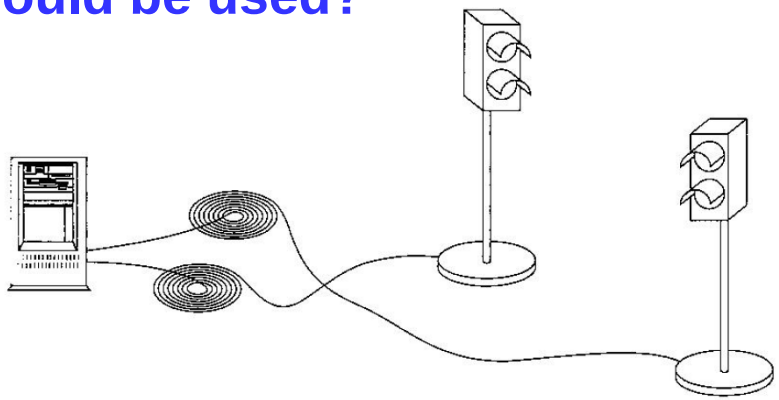
  (2) For 120 seconds one unit shows Stop and the other Go,

  (3) For 40 seconds both show Stop again

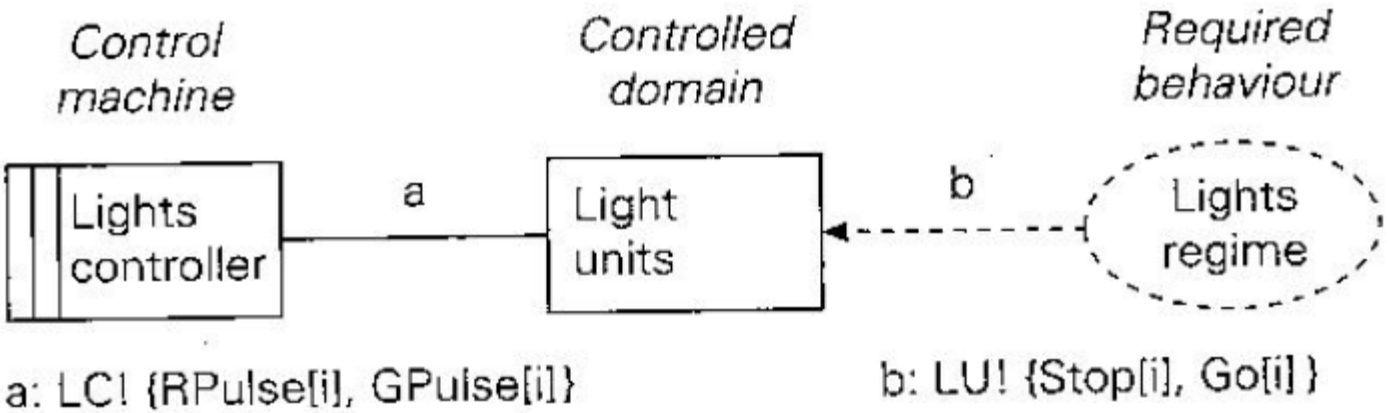
  (4) For 120 seconds the unit that previously showed Go shows Stop and the other show Go.
- Then the cycle is repeated

Which PF should be used?

Stop	Stop	Stop	Go	...
50	120	40	120	
Stop	Go	Stop	Stop	



Required Behavior PF



- LC!{RPulse[i],GPulse[i]} : Light controller controls the RPulse and Gpulse events .
- LU!{Stop[i],Go[i]) : Light units controls Stop and Go signals.

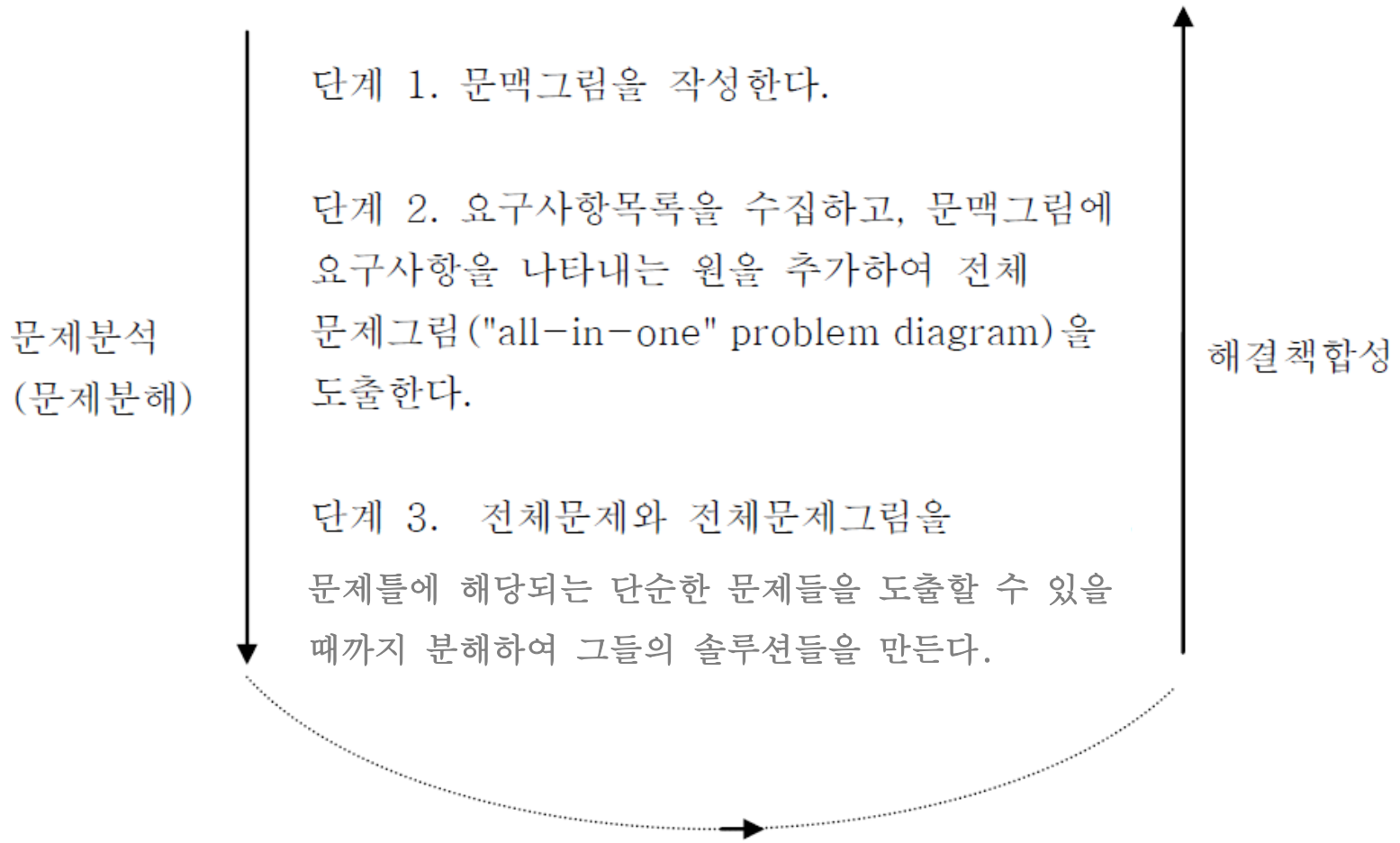


그림 3 -4. 문제분석과 소프트웨어 개발공정([Wikipedia 10, Problem Frames])

문제틀의 적용예제

Source:

[Hall 02] Hall, J.G., Jackson, M., Laney, R.C., Nuseibeh, B., Rapanotti, L., "Relating software requirements and architectures using problem frames", Proceedings of IEEE Joint International Conference on Requirements Engineering, pp. 137-144, 2002.

Warehouse Ordering System

- The warehouse contains initial quantities of a range of products.
- Customers place orders by specifying an order number, a product, and a quantity.
- Orders are processed by passing requests to the warehouse.
- If the warehouse can satisfy an order from stock, it reserves the specified quantity of stock and the allocation is notified. Otherwise no allocation or reservation is made.
- No customer should be left waiting for an order indefinitely, nor should ‘queue jumping’ be allowed.

Subproblems

- **Satisfy the customers' orders if possible**: the machine (i.e. the ordering system) conveys the order to the warehouse and notifies the customer whether the order can be satisfied.

☛ PF => Commanded behavior problem

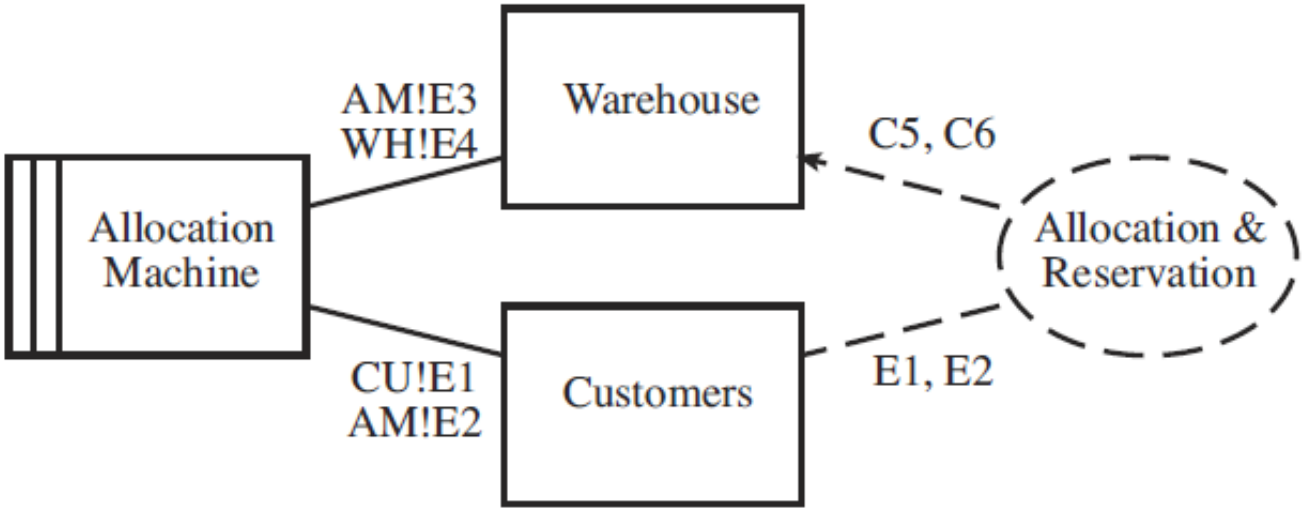
- **Prevent queue jumping**: the machine ensures that orders are processed First Come First Served.

☛ PF => Commanded behavior problem

- **Enforce mutual exclusion**: the machine ensures that orders are dealt with by the warehouse One At A Time.

☛ PF => Commanded behavior problem

Satisfying customer orders



E: event
C: cause

(1) CU!E1: Order(O#, Prod, Qty)
(2) AM!E2:Reply(O#, Prod, Success)
(3) AM!E3:Rqst(O#, Prod, Qty)
(4) WH!E4: Alloc(O#, Prod, Qty, Success)
(5) WH!C5: InitQ(Prod, Qty) Current quantity
(6) WH!C6: Resvd(Prod, Qty) Quantity reserved

Figure 3. A commanded behaviour frame for Allocation & Reservation

Formal notation for requirements specification

$a \rightsquigarrow b$ (read as *a* leads to *b*) the occurrence of *a* is sufficient and necessary to cause the occurrence of *b*;

$a \# b$ (read as *a* before *b*): the occurrence of *a* is before the occurrence of *b*; and

$a \hat{\#} b$ (read as *a* immediately before *b*): the occurrence of *a* is before the occurrence of *b*, and no other event instance of the class of *a* or *b* will occur in between.

Satisfying customer orders

- The developers have identified three requirements associated with the Allocation & Reservation requirement.

R1a: $Order(o, p, q) \leadsto Reply(o, p, success)$

Each order will eventually receive a reply where success is either true or false.

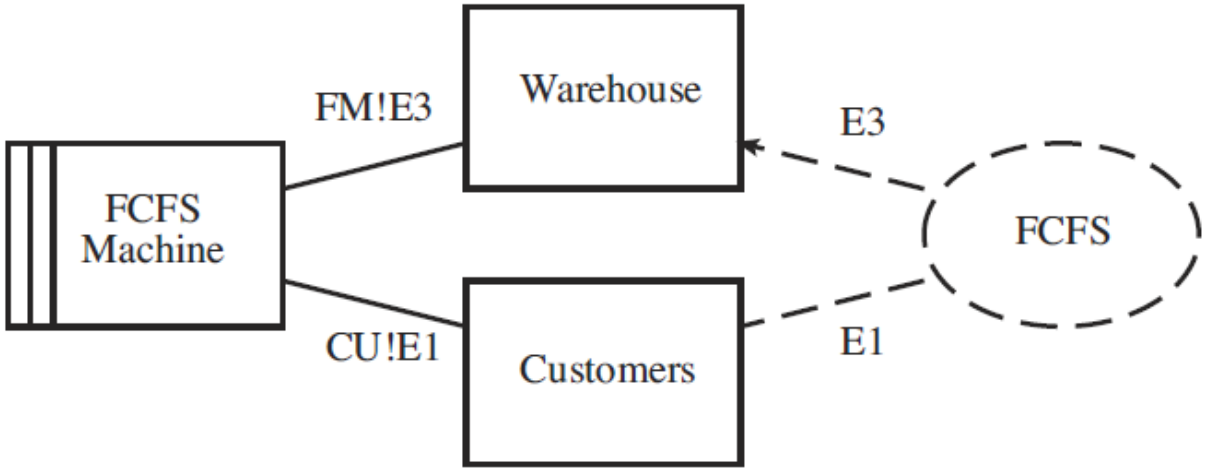
R1b: $Order(o, p, q) \wedge [InitQ(p, q_i) \wedge Resvd(p, q_j) \wedge (q_i - q_j) \geq q] \leadsto Reply(o, p, true) \wedge [Resvd(p, q_j + q)]$

A satisfiable order will result in a reply and a reservation.

R1c: $Order(o, p, q) \wedge [InitQ(p, qi) \wedge Resvd(p, q_j) \wedge (q_i - q_j) < q] \leadsto Reply(o, p, false) \wedge [Resvd(p, q_j)]$

An unsatisfiable order will result in a reply, but no reservation.

Preventing Queue Jumping

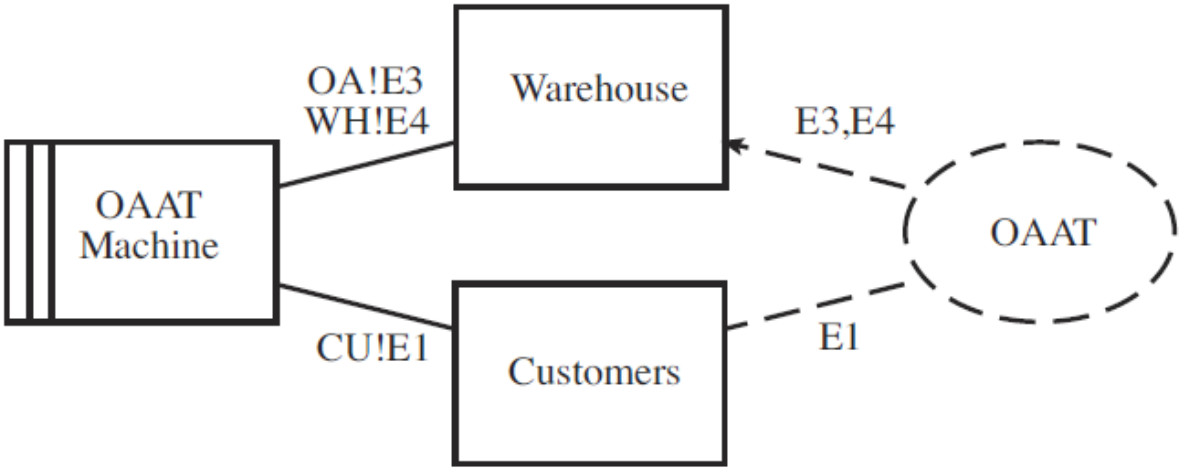


(1) CU!E1: Order(O#, Prod, Qty) (7) FM!E3: Rqst(O#, Prod, Qty)

Figure 4. A commanded behaviour frame for First Come First Served

$$\begin{aligned} \text{R2:}\quad & Order(o_1,p_1,q_1) \rightsquigarrow Rqst(o_1,p_1,q_1) \\ & \wedge\, Order(o_2,p_2,q_2) \rightsquigarrow Rqst(o_2,p_2,q_2) \\ & \wedge\, Order(o_1,p_1,q_1) \,\#\, Order(o_2,p_2,q_2) \\ & \Rightarrow Rqst(o_1,p_1,q_1) \,\#\, Rqst(o_2,p_2,q_2) \end{aligned}$$

Enforcing Mutual Exclusion



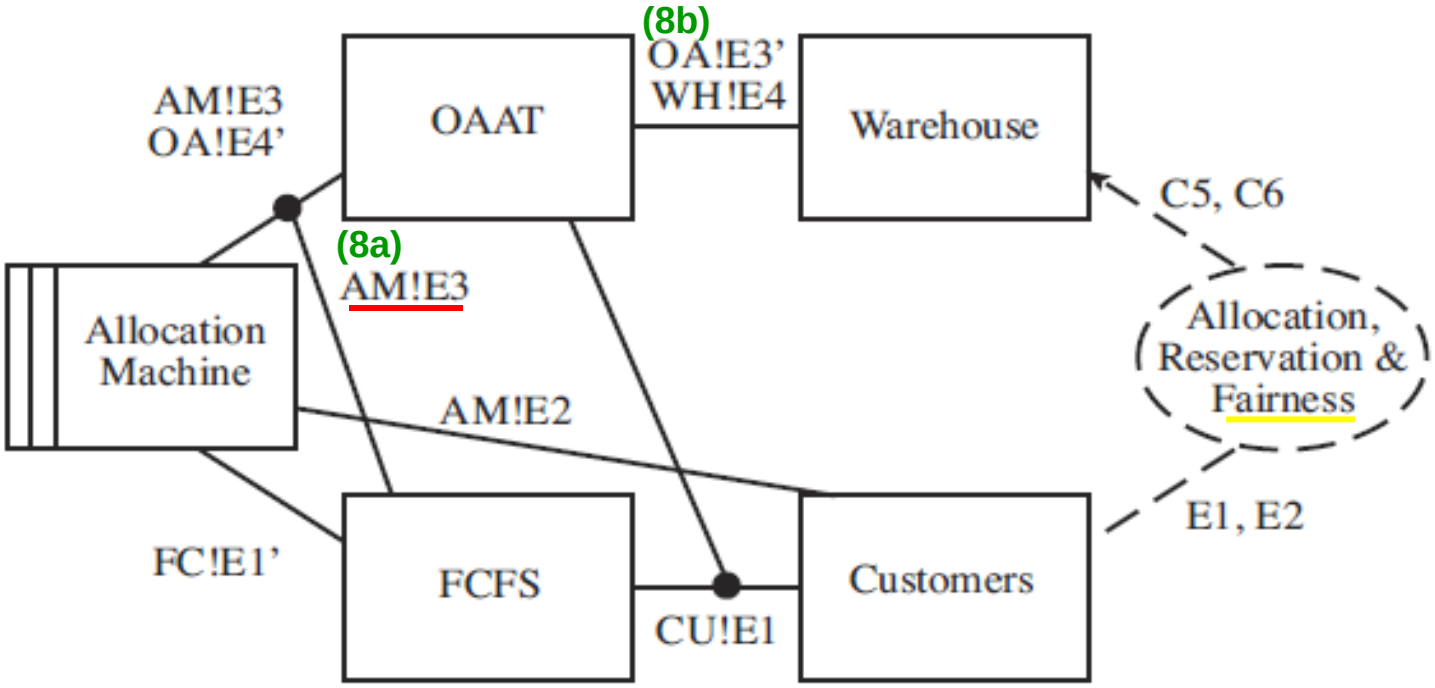
(8) OA!E3: Rqst(O#, Prod, Qty) (4) WH!E4: Alloc(O#, Prod, Qty, Success)
(1) CU!E1: Order(O#,Prod,Qty)

Figure 5. A commanded behaviour frame for One At A Time

$$\text{R3:} \quad Rqst(o,p,q) \rightsquigarrow Alloc(o,p,q,success)$$
$$\Rightarrow Rqst(o,p,q) \hat{\#} Alloc(o,p,q,success)$$

At this point, the developers produce a machine specification for each subframe, build correctness arguments for the satisfaction of each subproblem's frame concern.

Then need to **recompose them** in a way that satisfies the **combined requirements** to produce a correct system.



(1) CU!E1: Order(O#, Prod, Qty)
FC!E1': Order'(O#, Prod, Qty)
(3) AM!E3: Rqst(O#, Prod, Qty)
(8) OA!E3': Rqst'(O#, Prod, Qty)
(5) WH!C5: InitQ(Prod, Qty)

(4) WH!E4: Alloc(O#, Prod, Qty, Success)
OA!E4': Alloc'(O#, Prod, Qty, Success)
(2) AM!E2: Reply(O#, Prod, Success)
(6) WH!C6: Resvd(Prod, Qty)

Figure 6. OAAT and FCFS services as given domains

- The idea of Problem Frames is important.

- However, many problems arise when we want to use it beyond problem analysis:
  - How can the analyzed problems be linked to solutions?
  - How can we utilize the combined problem diagram?

Questions?