Structured Analysis and Structured Design

- Introduction to SASD
- Structured Analysis
- Structured Design

JUNBEOM YOO
jbyoo@konkuk.ac.kr
http://dslab.konkuk.ac.kr

Ver. 2.0

References

- Modern Structured Analysis, Edward Yourdon, 1989.
- Introduction to System Analysis and Design: a Structured Approach, Penny A. Kendall, 1996.

- Zhou Qun, Kendra Hamilton, and Ibrahim Jadalowen (2002).
  Structured Analysis and Structured Design (SASD) - Class Presentation

Structured Analysis

- Structured analysis is [Kendall 1996]
  - a set of techniques and graphical tools
  - that allow the analysts to develop a new kind of system specification
  - that are easily understandable to the users.
  - Analysts work primarily with their wits, pencil and paper.

- **SASD**
  - **Structured Analysis and Structured Design**

History of SASD

- Developed in the late 1970s by DeMarco, Yourdon and Constantine after the emergence of structured programming.
- IBM incorporated SASD into their development cycle in the late 1970s and early 1980s.
- Yourdon published the book “Modern Structured Analysis” in 1989.
- The availability of CASE tools in 1990s enabled analysts to develop and modify the graphical SASD models.

Philosophy of SASD

- Analysts attempt to divide large, complex problems into smaller, more easily handled ones.
  → **Divide and Conquer**

- **Top-Down approach**

- **Functional view** of the problem

- Analysts use graphics to illustrate their ideas whenever possible.

- Analysts must keep a written record.

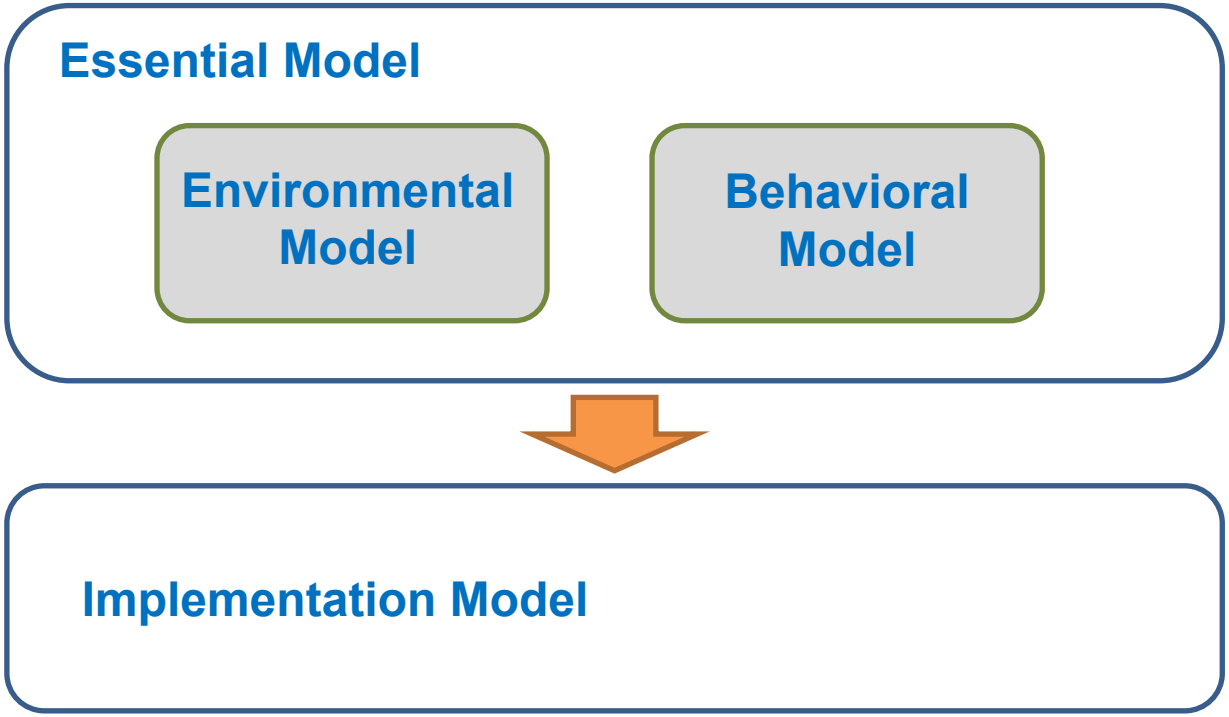
Philosophy of SASD

- "The purpose of SASD is to develop a useful, high quality information system that will meet the needs of the end user."
  [Yourdon 1989]

Goals of SASD

- Improve quality and reduce the risk of system failure.
- Establish concrete requirements specifications and complete requirements documentations.
- Focus on reliability, flexibility and maintainability of system.

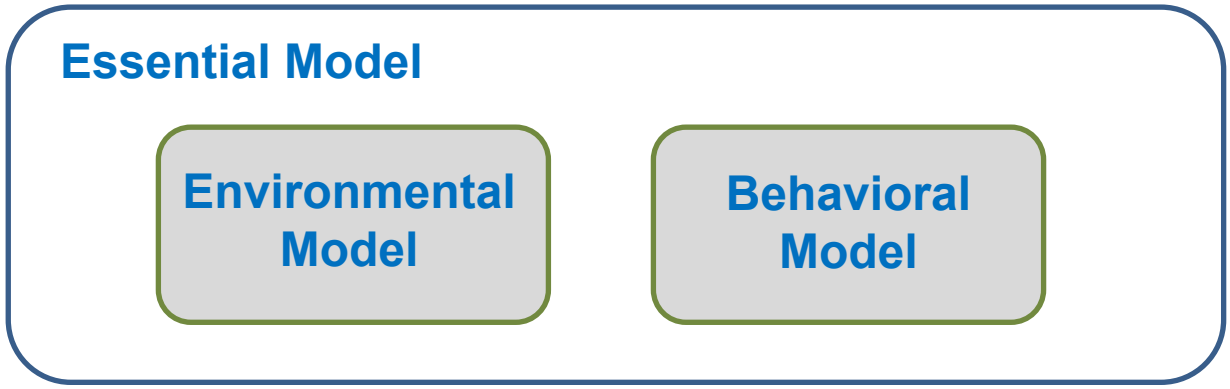
Elements of SASD



Essential Model

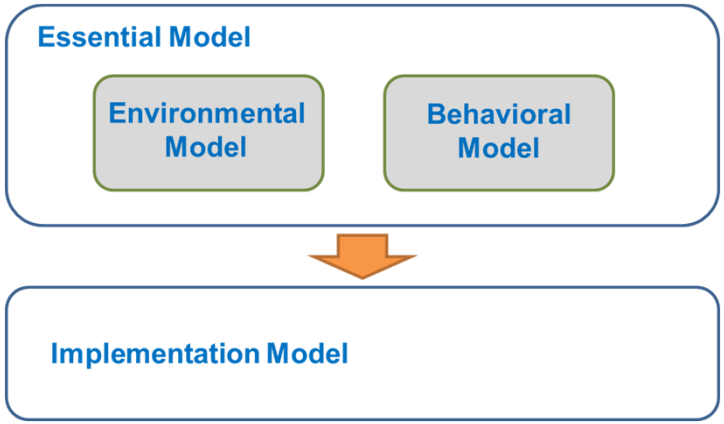
- Model of **what** the system must do
  - Not define how the system will accomplish its purpose.

  - Environmental model
  - Behavioral model



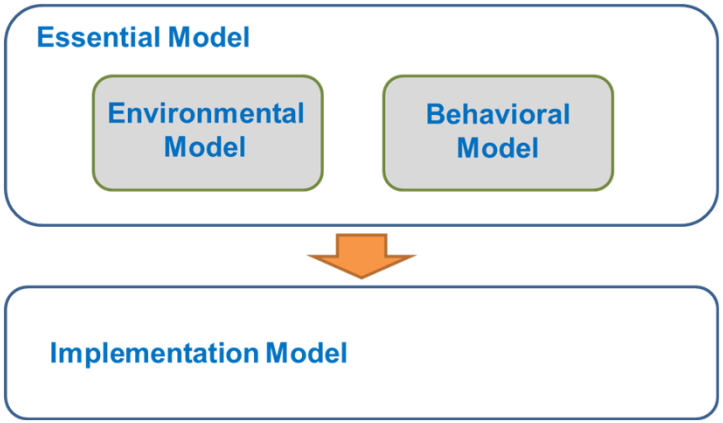
Environmental Model

- Defines the **scope** of the proposed system.
- Defines the **boundary** and **interaction** between the system and the outside world.
- Composed of
  - **Statement of Purpose**
  - **System Context diagram**
  - **Event list**



Behavioral Model

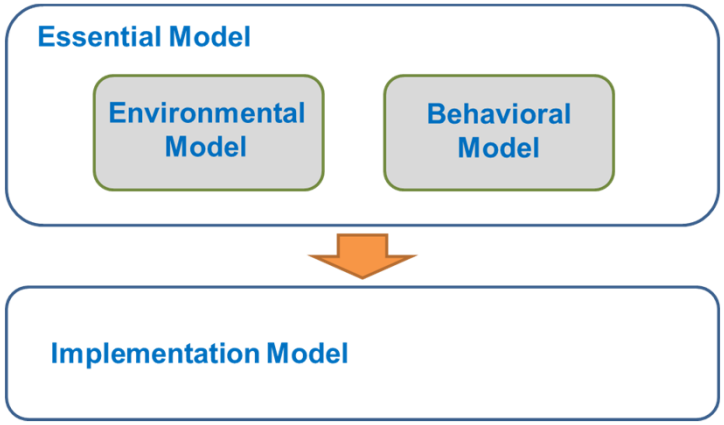
- Model of the **internal behavior** and **data entities** of the system.
- Models **functional requirements**.
- Composed of
  - **Data Dictionary**
  - **Data Flow Diagram (DFD)**
  - **Entity Relationship Diagram (ERD)**
  - **Process Specification**
  - **State Transition Diagram**



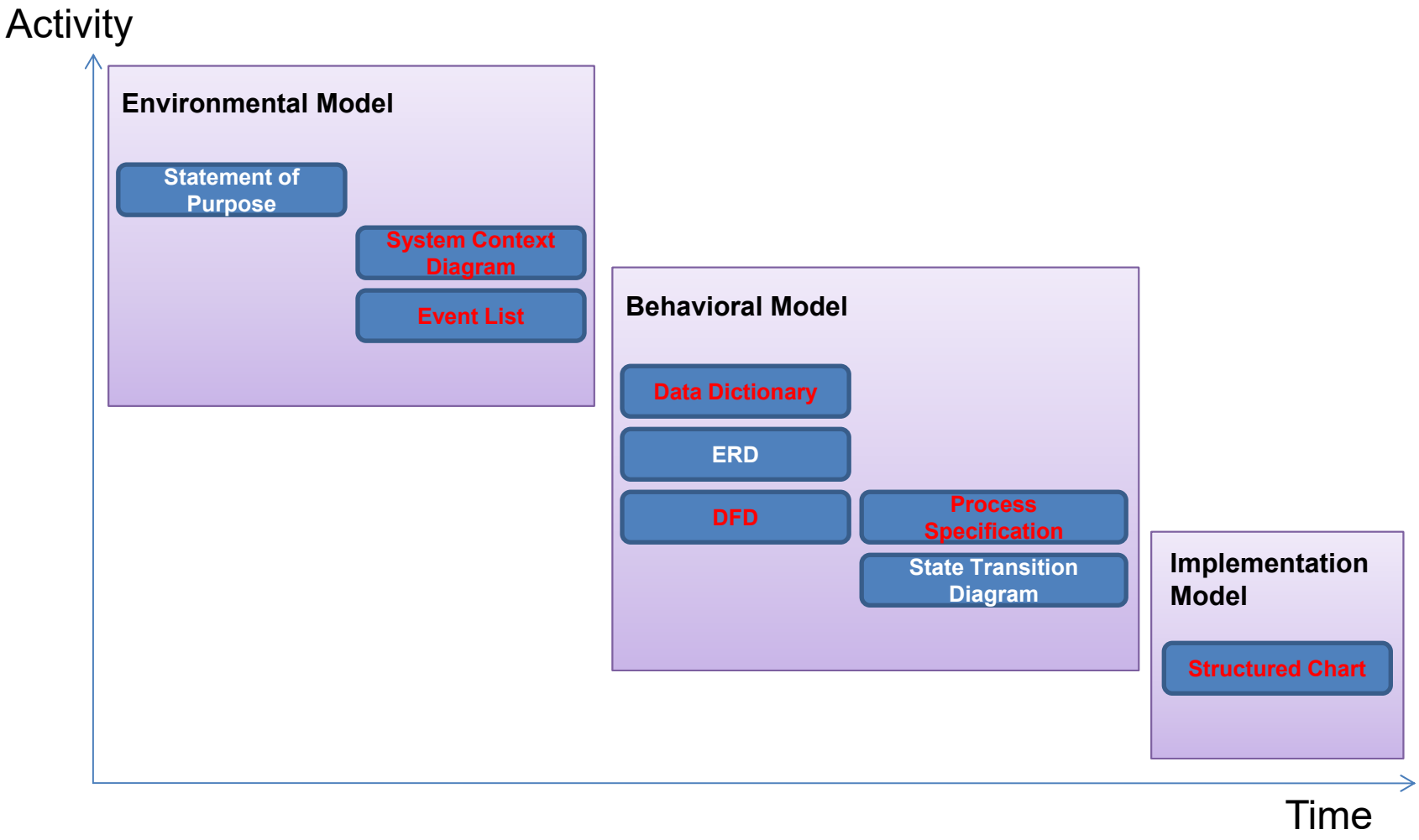
Implementation Model

- Map **functional requirements** to **hardware** and **software**.
  - Minimizes the cost of the development and maintenance
  - Determines which functions should be manual vs. automated.
    - Can be used to discuss the cost-benefits of functionality with user/stakeholders.
  - Defines the Human-Computer interface
  - Defines non-functional requirements

- Composed of
  - **Structure Charts**



SASD Process



STRUCTURED ANALYSIS (SA)

Statement of Purpose

- A clear and concise textual description of the purpose for the system to develop
  - Should be deliberately vague.
  - Intended for top level management, user management and others who are not directly involved in the system.

Statement of Purpose – RVC Example

- **PFR (Preliminary Functional Requirements)**

Robot Vacuum Cleaner (RVC) Controller

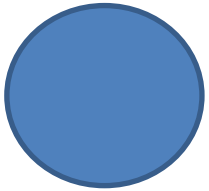
- An RVC automatically cleans and mops household surface.
- It goes straight forward while cleaning.
- If its sensors found an obstacle, it stops cleaning, turns aside left or right, and goes forward with cleaning.
- If it detects dust, power up the cleaning for a while.
- We do not consider the detail design and implementation on HW controls.
- We only focus on the automatic cleaning function.

System Context Diagram

- A special case of DFD (Data Flow Diagram)
  - **DFD Level 0**

  - Highlights the boundary between the system and outside world.
  - Highlights the people, organizations and outside systems that interact with the system under development.

System Context Diagram - Notation

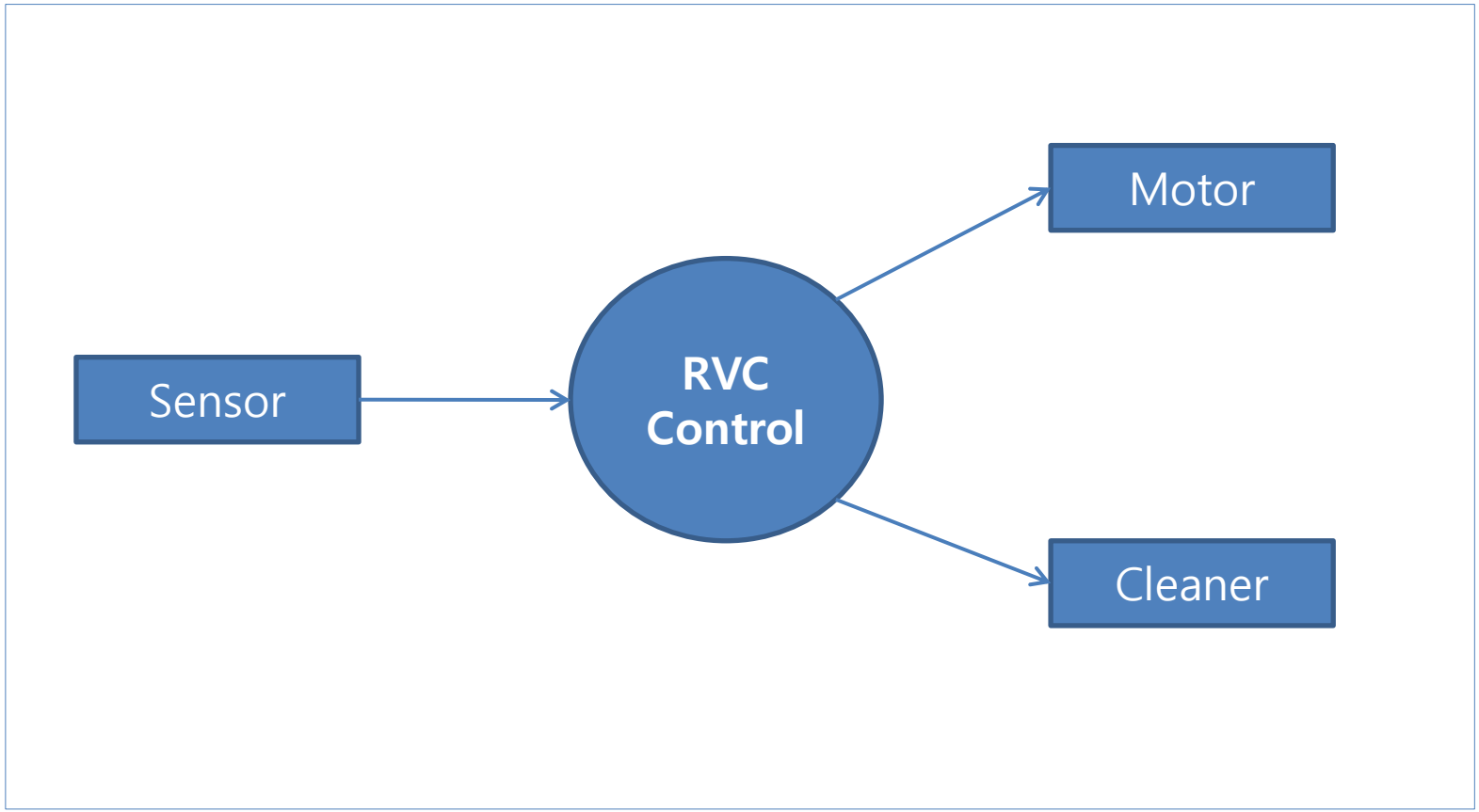


Process : represents the proposed system

Terminator : represents the external entities

Flow : represents the in/out data flows

System Context Diagram – RVC Example



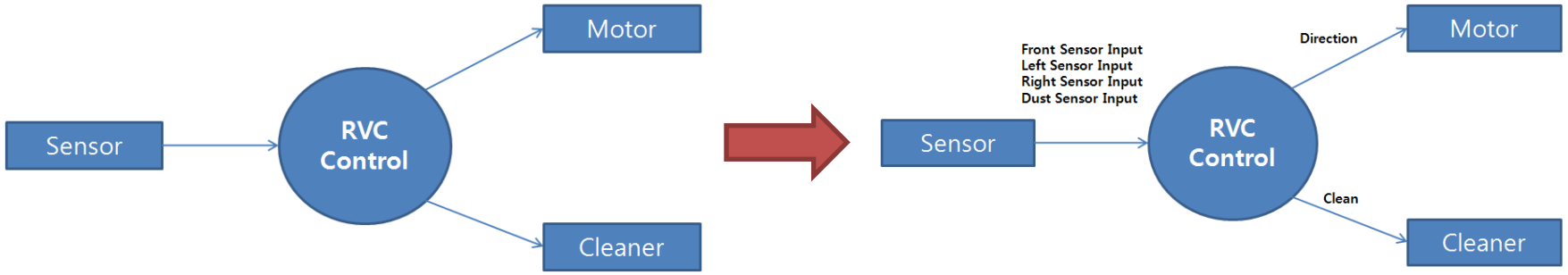
Event List

- A list of the event/stimuli outside of the system to which it must respond.
  - Used to describe the context diagram in detail.

- Types of events
  - **Flow-oriented event** : triggered by incoming data
  - **Temporal event** : triggered by internal clock
  - **Control event** : triggered by an external unpredictable event

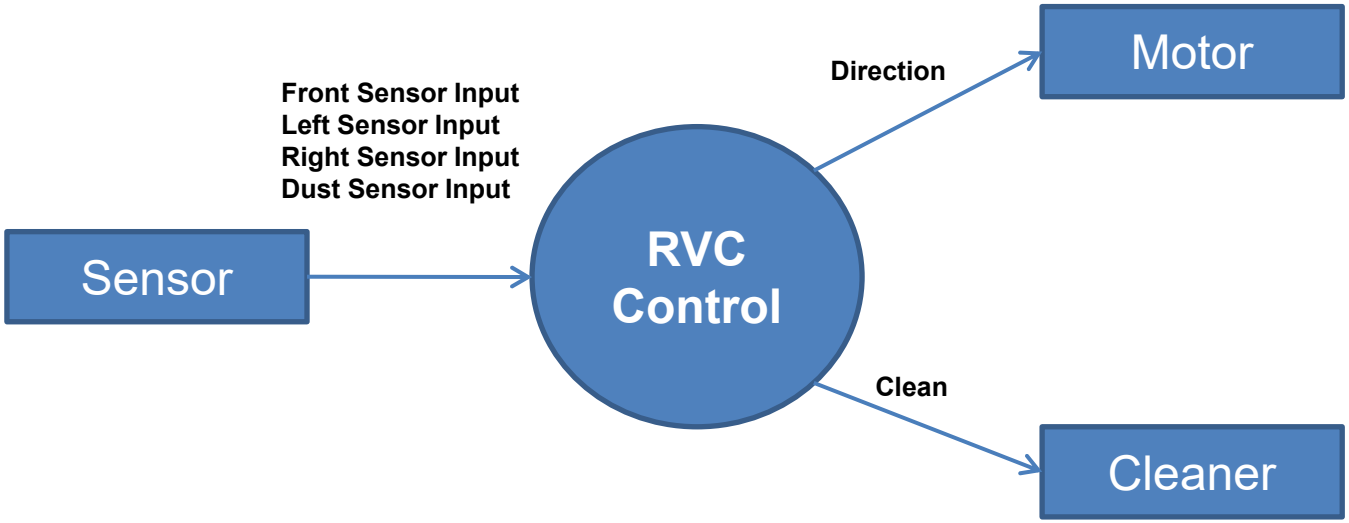
Event List – RVC Example

Input/ Output Event	Description
Front Sensor Input	Detects obstacles in front of the RVC
Left Sensor Input	Detects obstacles in the left side of the RVC periodically
Right Sensor Input	Detects obstacles in the right side of the RVC periodically
Dust Sensor Input	Detects dust on the floor periodically
Direction	Direction commands to the motor (go forward / turn left with an angle / turn right with an angle)
Clean	Turn off / Turn on / Power-Up



Context Diagram for RVC

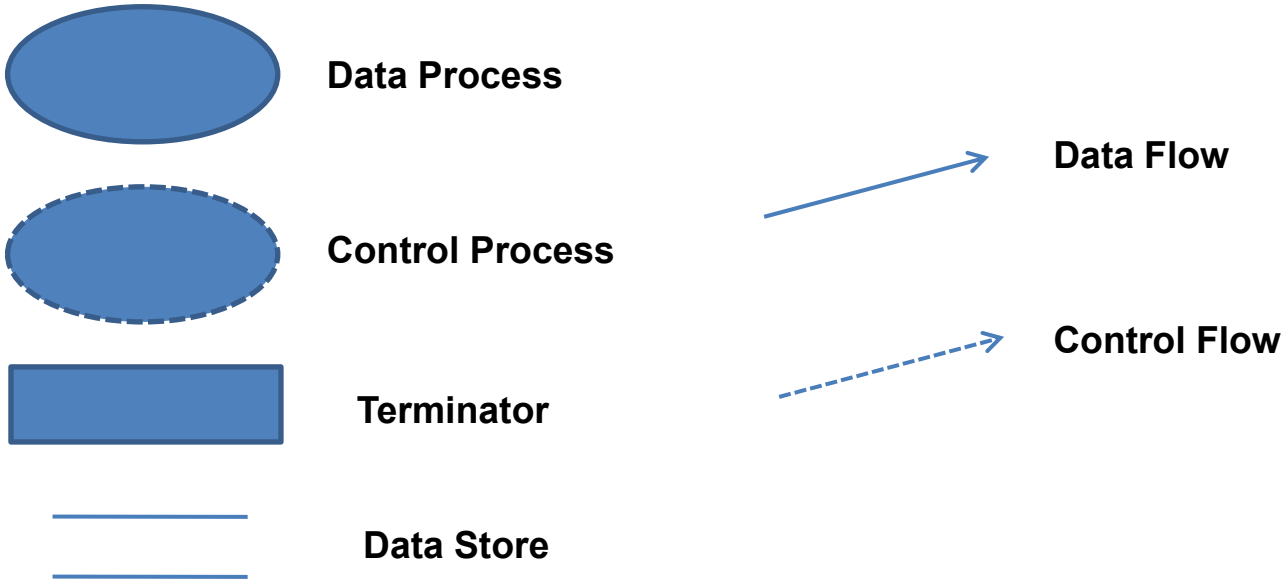
System Context Diagram – RVC Example



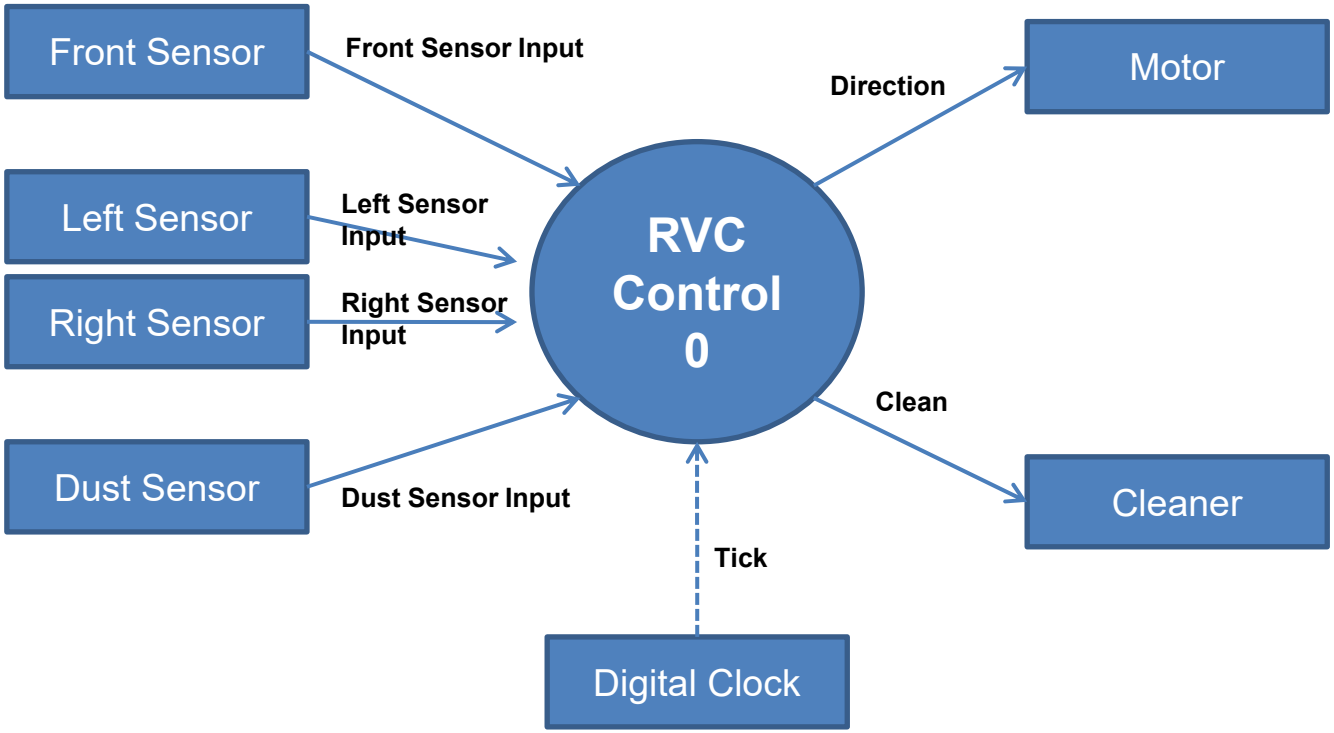
Data Flow Diagram (DFD)

- Provides a means for functional decomposition.
  - Composed of hierarchies(levels) of DFDs.

- Notation (A kind of CDFD)



DFD Level 0 – RVC Example

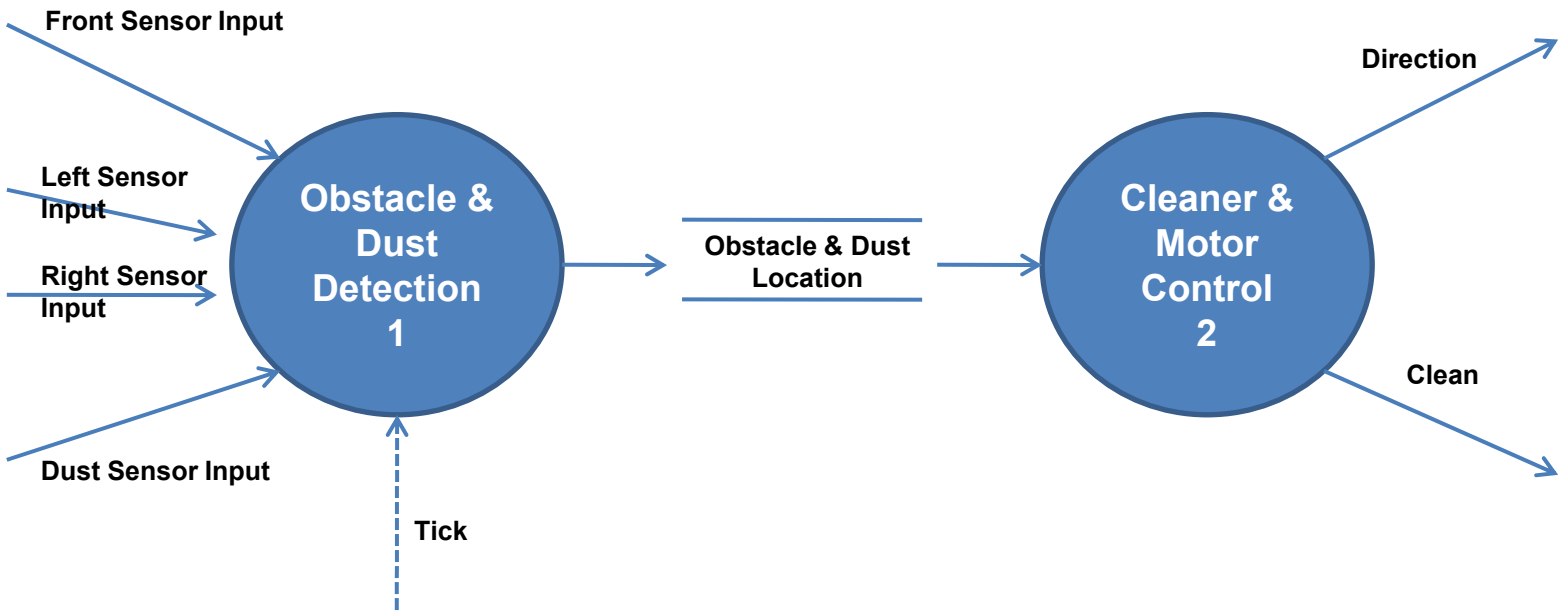


DFD Level 0 – RVC Example

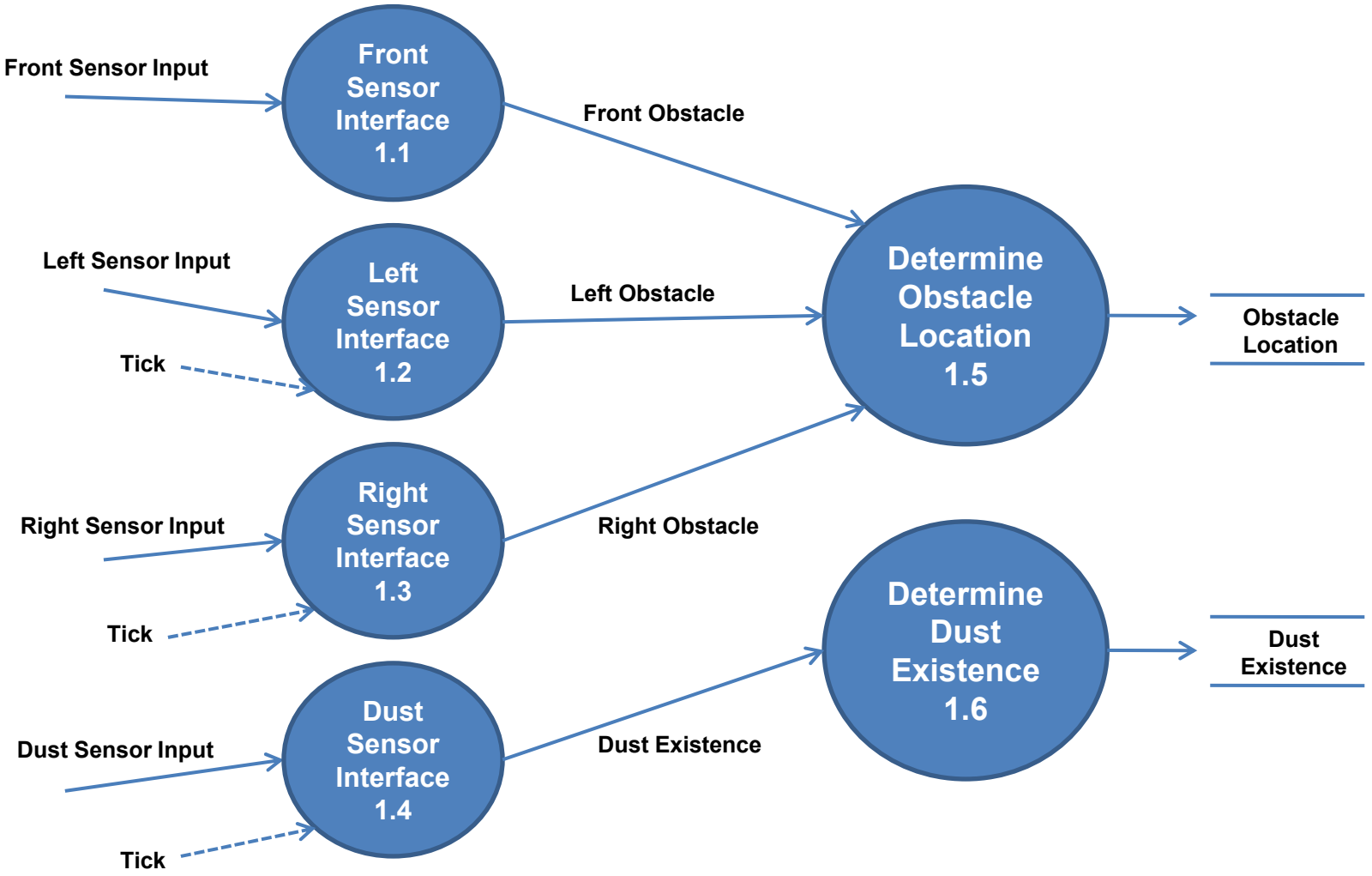
(A kind of) Data Dictionary

Input/ Output Event	Description	Format / Type
Front Sensor Input	Detects obstacles in front of the RVC	True / False ， Interrupt
Left Sensor Input	Detects obstacles in the left side of the RVC periodically	True / False ， Periodic
Right Sensor Input	Detects obstacles in the right side of the RVC periodically	True / False ， Periodic
Dust Sensor Input	Detects dust on the floor periodically	True / False ， Periodic
Direction	Direction commands to the motor (go forward / turn left with an angle / turn right with an angle)	Forward / Left / Right / Stop
Clean	Turn off / Turn on / Power-Up	On / Off / Up

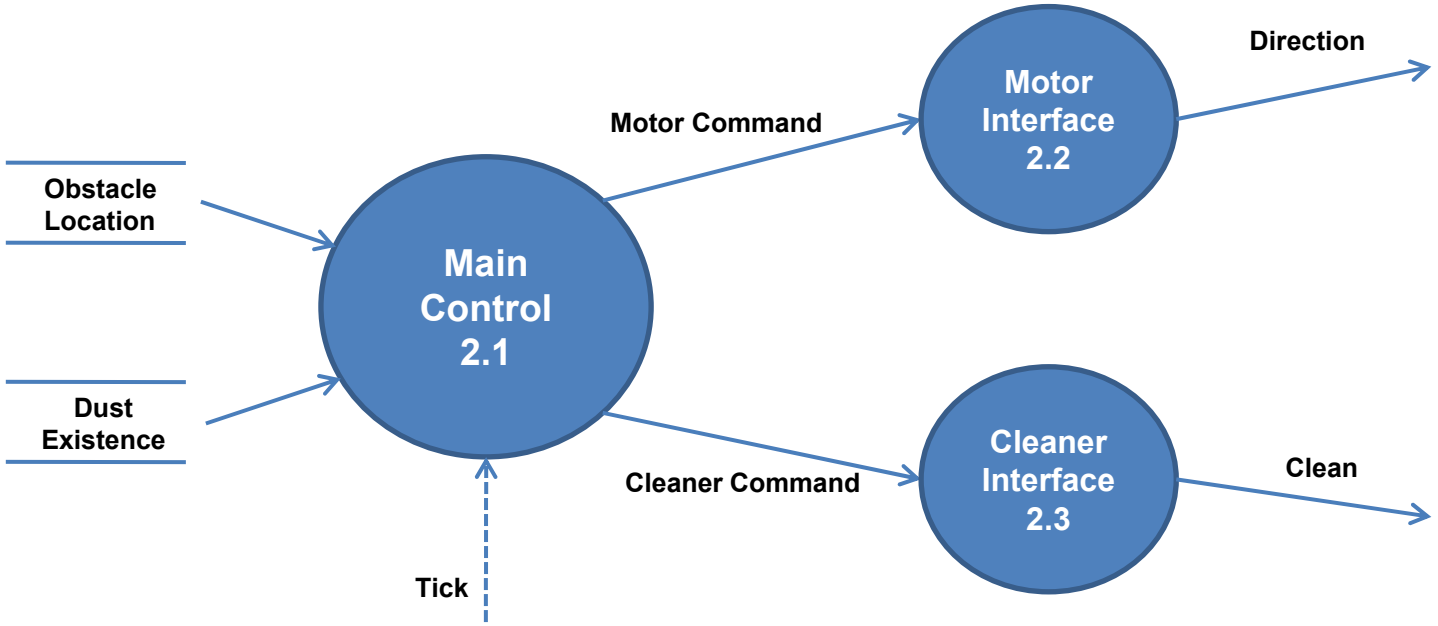
DFD Level 1 – RVC Example



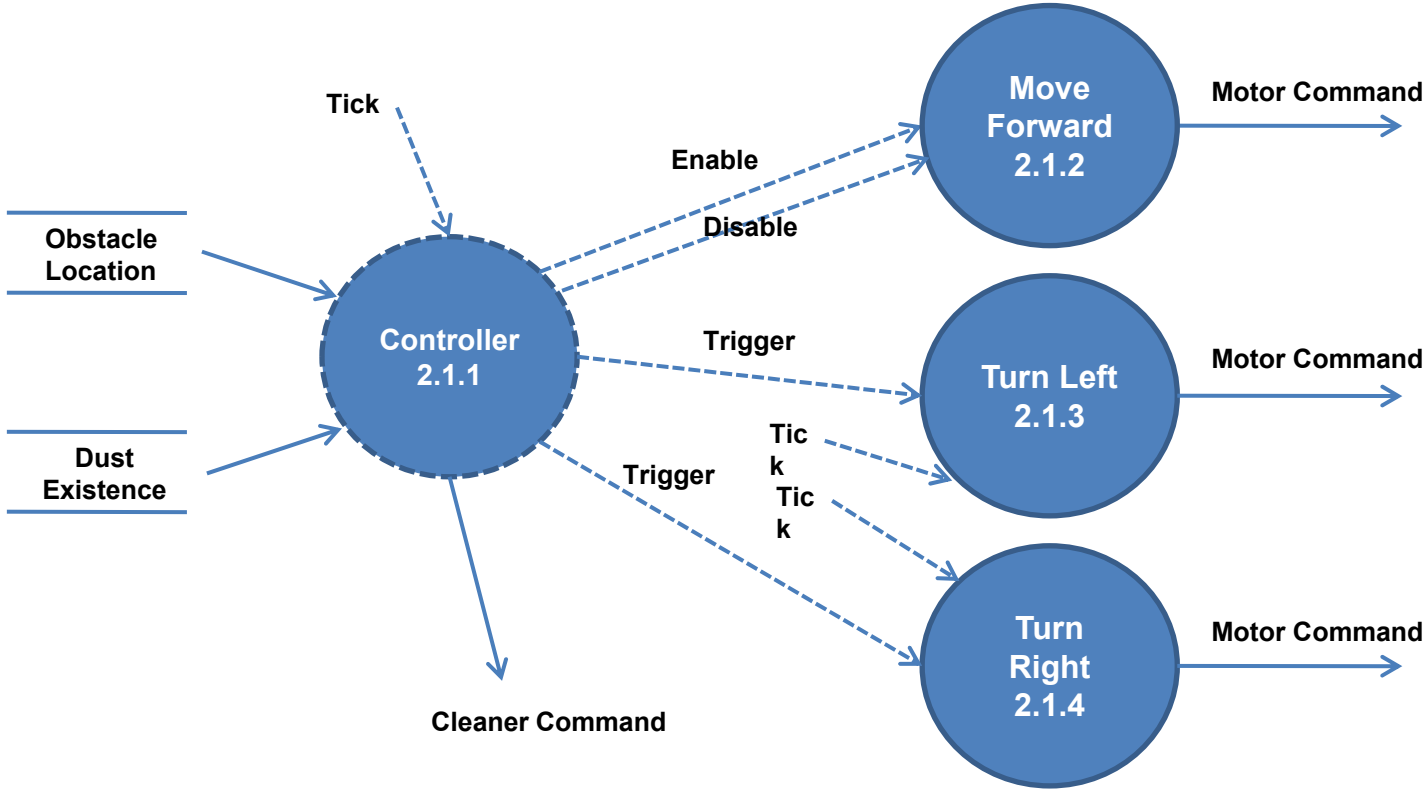
DFD Level 2 – RVC Example



DFD Level 2 – RVC Example

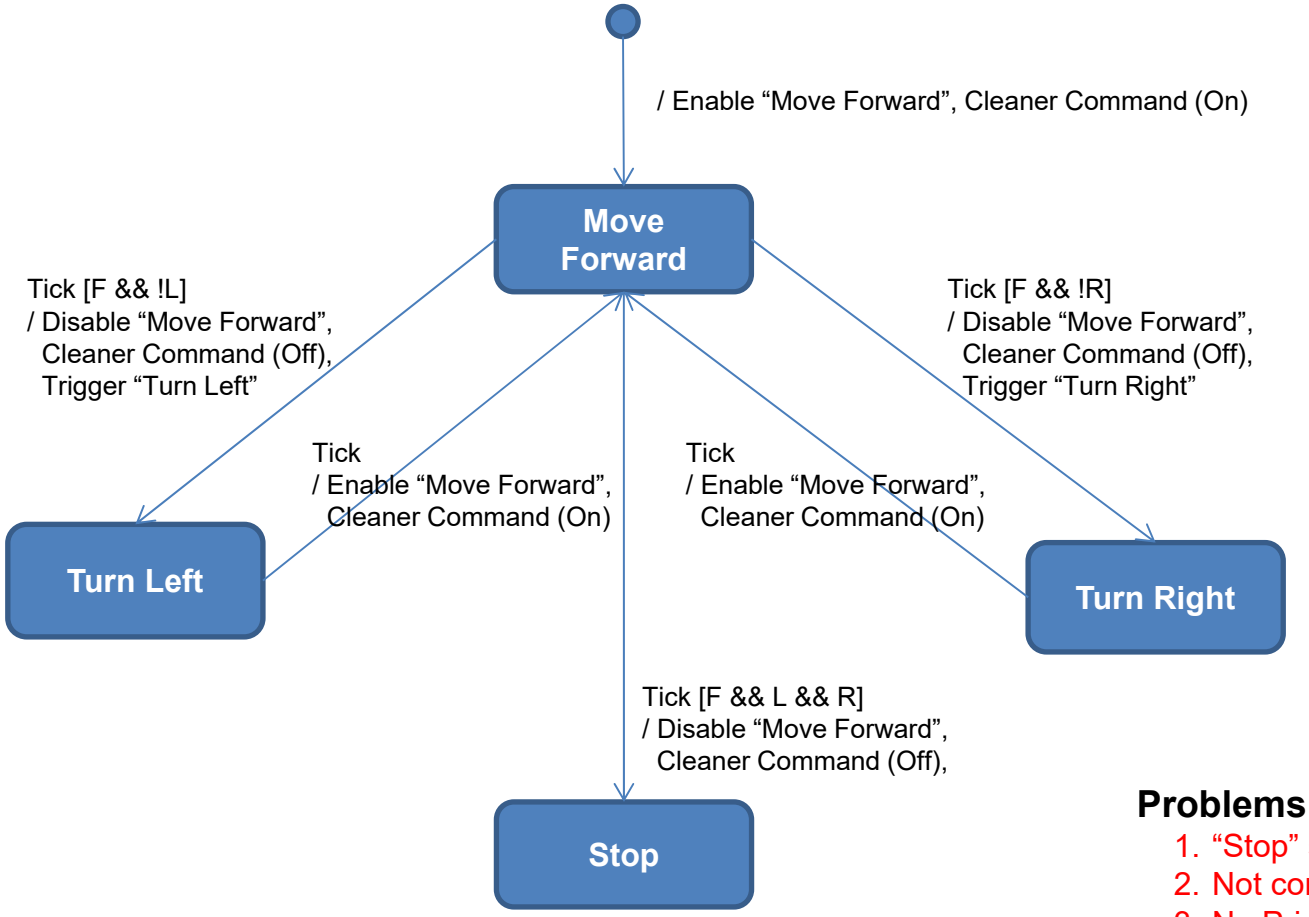


DFD Level 3 – RVC Example



DFD Level 4 – RVC Example

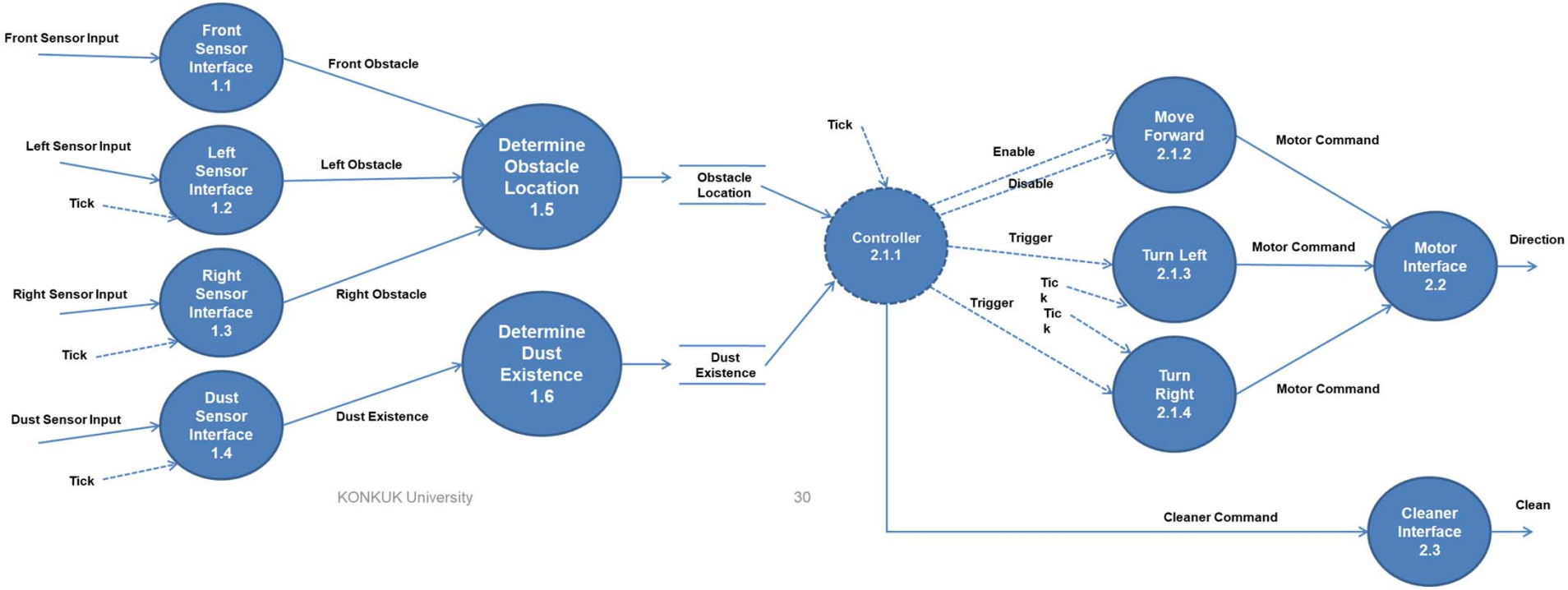
State Transition Diagram for Controller 2.1.1



Problems in this model:

1. “Stop” state (deadlock)
2. Not consider “Dust”
3. No Priority for left/right turn

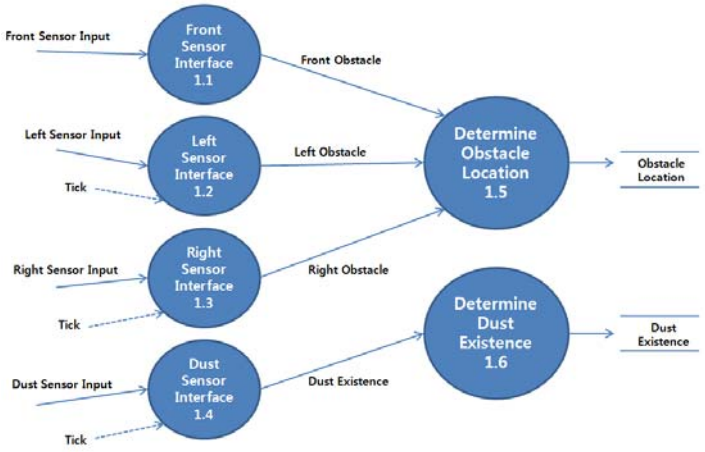
DFD – RVC Example



Process Specification

- Shows process details which are implied but not shown in a DFD.
  - Specifies the input, output, and algorithm of a module in a DFD.
  - Normally written in pseudo-code or table format.

- Example – Left Sensor Interface



Reference No.	1.2
Name	Left Sensor Interface
Input	Left Sensor Input (+Data structure if possible) , Tick
Output	Left Obstacle (+Data structure)
Process Description	“Left Sensor Input” process reads a analog value of the left sensor periodically, converts it into a digital value such as True/False, and assigns it into output variable “Left Obstacle.”

Data Dictionary

- Defines data elements to avoid different interpretations.
  - Not used widely in recent years
  - Often used in a simple form like the example below

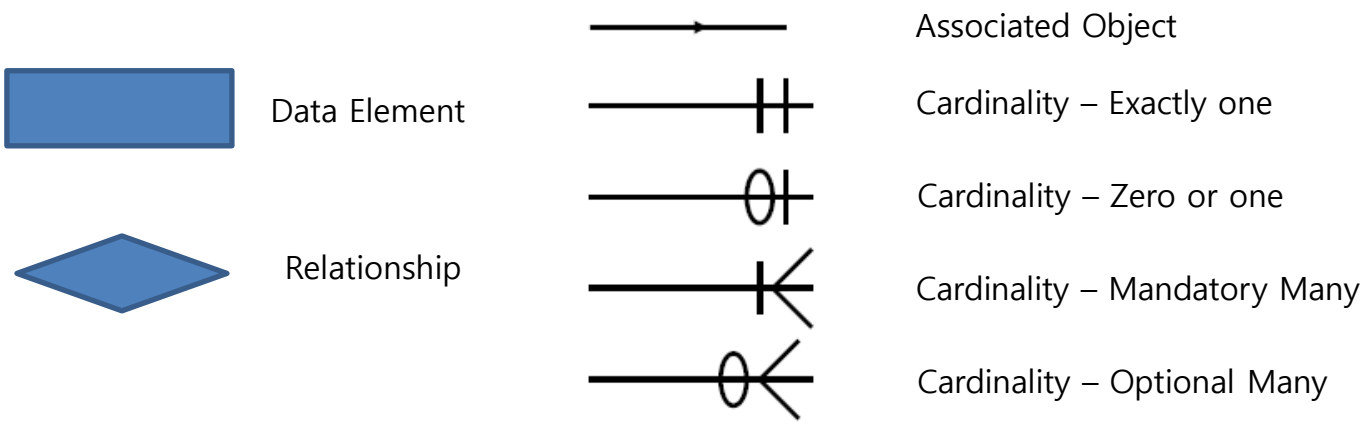
- Example :

Input/ Output Event	Description	Format / Type
Front Sensor Input	Detects obstacles in front of the RVC	True / False , Interrupt
Left Sensor Input	Detects obstacles in the left side of the RVC periodically	True / False , Periodic
Right Sensor Input	Detects obstacles in the right side of the RVC periodically	True / False , Periodic
Dust Sensor Input	Detects dust on the floor periodically	True / False , Periodic
Direction	Direction commands to the motor (go forward / turn left with an angle / turn right with an angle)	Forward / Left / Right / Stop
Clean	Turn off / Turn on / Power-Up	On / Off / Up

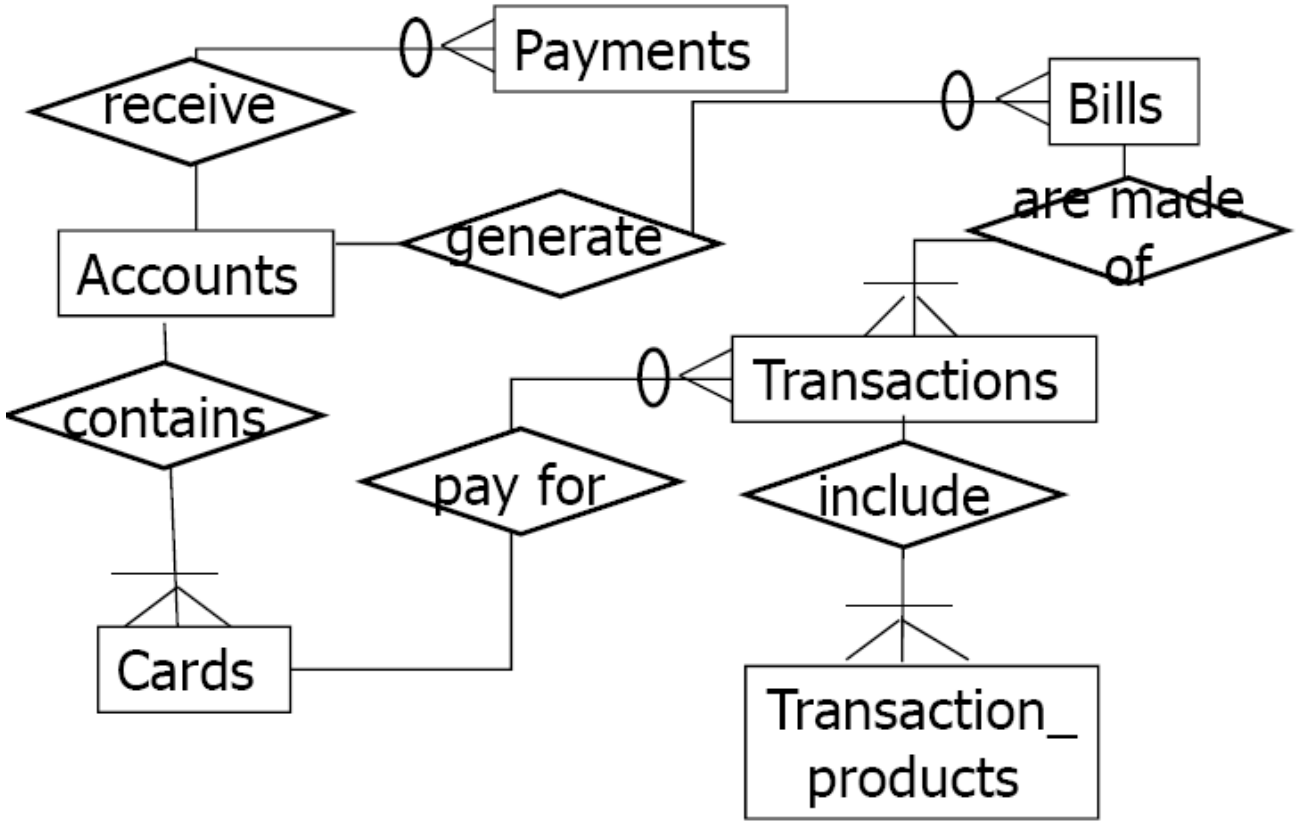
Entity Relationship Diagram (ERD)

- A graphical representation of the data layout of a system at a high level of abstraction
  - Defines data elements and their inter-relationships in the system.
  - Similar with the class diagram in UML.

- Notation (Original)

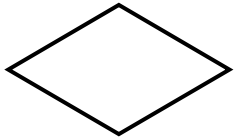


Entity Relationship Diagram – Example



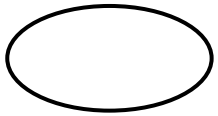
Entity Relationship Diagram – Example

Entity

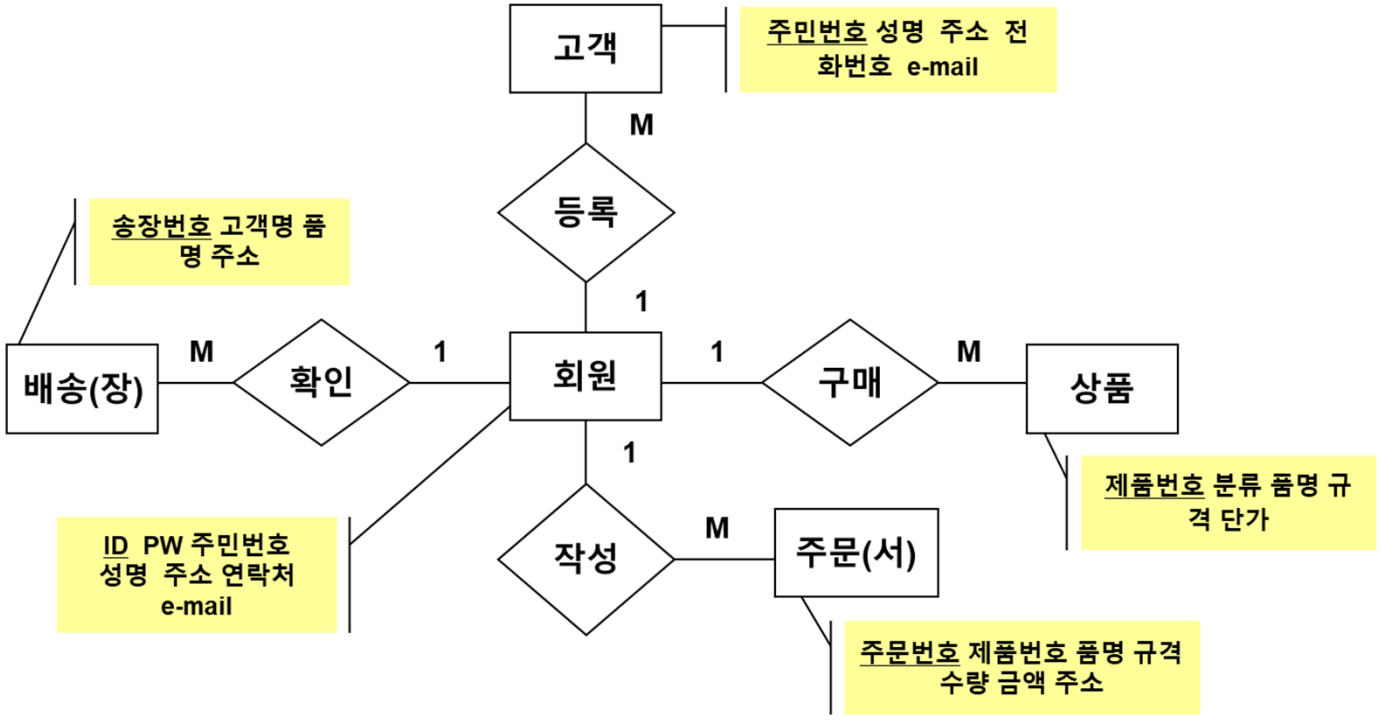


Relationship

Relation Type



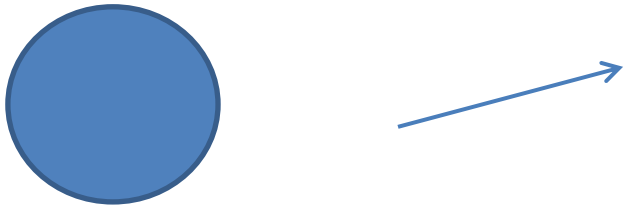
Attribute



State Transition Diagram

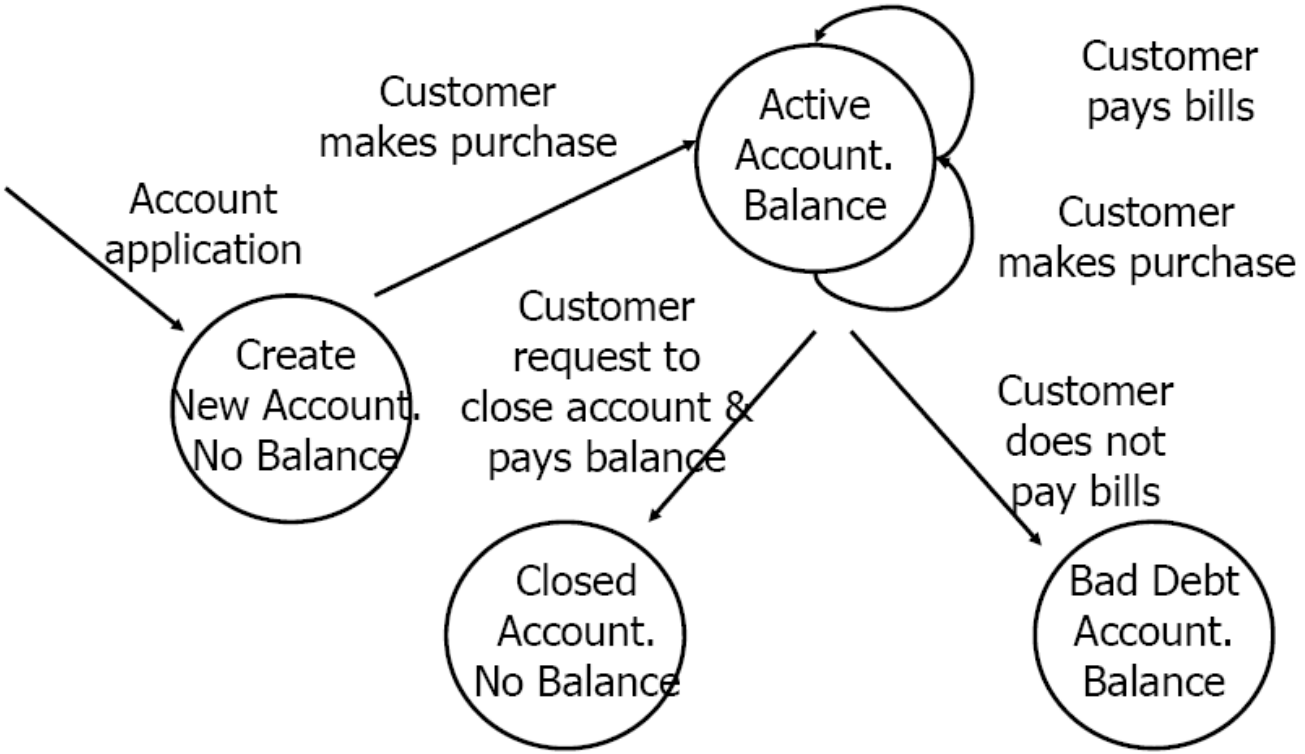
- Shows the time ordering between processes.
  - More primitive than the Statechart diagram in UML.
  - Different from the State transition diagram used in DFD.
  - Not widely used now.

- Notation



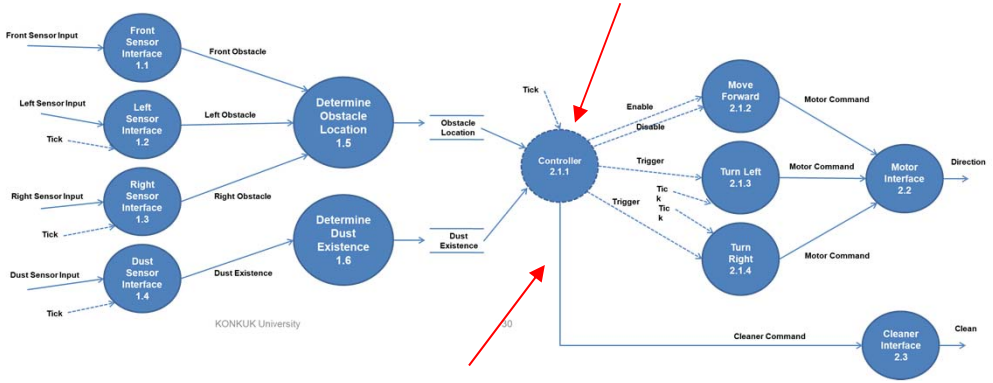
Objects Transitions

State Transition Diagram - Example



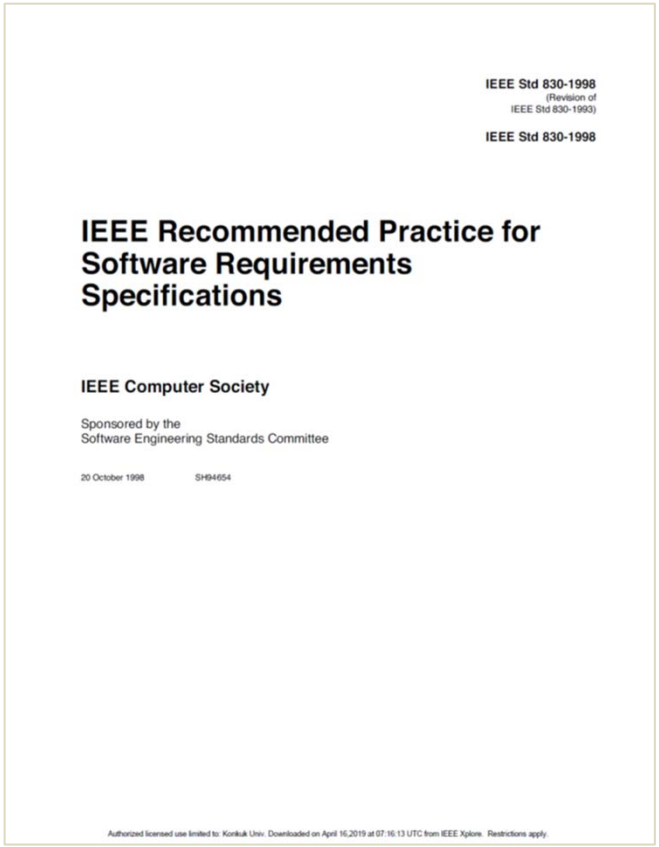
Practice #1 : Structured Analysis

- **Complete the RVC analysis in more details.**
  - Resolve the problems identified
    - "Stop" state (deadlock)
    - Not consider "Dust"
    - No Priority for left/right turn
    - Not design about Cleaner Interface
  - Complete full versions of process specifications and data dictionary
    - System Context Diagram
    - A hierarchy of DFDs
    - Process Specifications
    - Data Dictionary



Practice #2 : Software Requirements Specification (SRS)

- **Write an SRS (Software Requirements Specification) for the RVC Controller.**
  - In accordance with IEEE 830-1998 standards



IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Std 830-1998

IEEE Recommended Practice for Software Requirements Specifications

IEEE Computer Society

Sponsored by the
Software Engineering Standards Committee

20 October 1998 SH94654

Table of Contents

1. Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms, and abbreviations
- 1.4 References
- 1.5 Overview

2. Overall description
- 2.1 Product perspective
- 2.2 Product functions
- 2.3 User characteristics
- 2.4 Constraints
- 2.5 Assumptions and dependencies

3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)

Appendixes

Index

Figure 1—Prototype SRS outline

SRS Templates: IEEE STS 830-1998

A.1 Template of SRS Section 3 organized by mode: Version 1

3. Specific requirements
3.1 External interface requirements
  3.1.1 User interfaces
  3.1.2 Hardware interfaces
  3.1.3 Software interfaces
  3.1.4 Communications interfaces
3.2 Functional requirements
  3.2.1 Mode 1
    3.2.1.1 Functional requirement 1.1
    .
    .
    .
    3.2.1.*n* Functional requirement 1.*n*
  3.2.2 Mode 2
  .
  .
  .
  3.2.*m* Mode *m*
    3.2.*m*.1 Functional requirement *m*.1
    .
    .
    .
    3.2.*m*.*n* Functional requirement *m*.*n*
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

A.2 Template of SRS Section 3 organized by mode: Version 2

3. Specific requirements
3.1. Functional requirements
  3.1.1 Mode 1
    3.1.1.1 External interfaces
      3.1.1.1.1 User interfaces
      3.1.1.1.2 Hardware interfaces
      3.1.1.1.3 Software interfaces
      3.1.1.1.4 Communications interfaces
    3.1.1.2 Functional requirements
      3.1.1.2.1 Functional requirement 1
      .
      .
      .
      3.1.1.2.*n* Functional requirement *n*
    3.1.1.3 Performance
  3.1.2 Mode 2
  .
  .
  .
  3.1.*m* Mode *m*
3.2 Design constraints
3.3 Software system attributes
3.4 Other requirements

SRS Templates: IEEE STS 830-1998

A.5 Template of SRS Section 3 organized by feature

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 System features
  - 3.2.1 System Feature 1
    - 3.2.1.1 Introduction/Purpose of feature
    - 3.2.1.2 Stimulus/Response sequence
    - 3.2.1.3 Associated functional requirements
      - 3.2.1.3.1 Functional requirement 1
      - .
      - .
      - .
      - 3.2.1.3.*n* Functional requirement *n*
  - 3.2.2 System feature 2
  - .
  - .
  - .
  - 3.2.*m* System feature *m*
  - .
  - .
  - .
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

A.6 Template of SRS Section 3 organized by stimulus

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Stimulus 1
    - 3.2.1.1 Functional requirement 1.1
    - .
    - .
    - .
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 Stimulus 2
  - .
  - .
  - .
  - 3.2.*m* Stimulus m
    - 3.2.*m*.1 Functional requirement *m*.1
    - .
    - .
    - .
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

SRS Templates: IEEE STS 830-1998

A.7 Template of SRS Section 3 organized by functional hierarchy

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Information flows
    - 3.2.1.1 Data flow diagram 1
      - 3.2.1.1.1 Data entities
      - 3.2.1.1.2 Pertinent processes
      - 3.2.1.1.3 Topology
    - 3.2.1.2 Data flow diagram 2
      - 3.2.1.2.1 Data entities
      - 3.2.1.2.2 Pertinent processes
      - 3.2.1.2.3 Topology
    - .
    - .
    - .
    - 3.2.1.*n* Data flow diagram *n*
      - 3.2.1.*n*.1 Data entities
      - 3.2.1.*n*.2 Pertinent processes
      - 3.2.1.*n*.3 Topology
  - 3.2.2 Process descriptions
    - 3.2.2.1 Process 1
      - 3.2.2.1.1 Input data entities
      - 3.2.2.1.2 Algorithm or formula of process
      - 3.2.2.1.3 Affected data entities
    - 3.2.2.2 Process 2
      - 3.2.2.2.1 Input data entities
      - 3.2.2.2.2 Algorithm or formula of process
      - 3.2.2.2.3 Affected data entities
    - .
    - .
    - .
    - 3.2.2.*m* Process *m*
      - 3.2.2.*m*.1 Input data entities
      - 3.2.2.*m*.2 Algorithm or formula of process
      - 3.2.2.*m*.3 Affected data entities
  - 3.2.3 Data construct specifications
    - 3.2.3.1 Construct 1
      - 3.2.3.1.1 Record type
      - 3.2.3.1.2 Constituent fields
    - 3.2.3.2 Construct 2
      - 3.2.3.2.1 Record type
      - 3.2.3.2.2 Constituent fields
    - .
    - .
    - .
    - 3.2.3.*p* Construct *p*
      - 3.2.3.*p*.1 Record type
      - 3.2.3.*p*.2 Constituent fields
  - 3.2.4 Data dictionary
    - 3.2.4.1 Data element 1
      - 3.2.4.1.1 Name
      - 3.2.4.1.2 Representation
      - 3.2.4.1.3 Units/Format
      - 3.2.4.1.4 Precision/Accuracy
      - 3.2.4.1.5 Range
    - 3.2.4.2 Data element 2
      - 3.2.4.2.1 Name
      - 3.2.4.2.2 Representation
      - 3.2.4.2.3 Units/Format
      - 3.2.4.2.4 Precision/Accuracy
      - 3.2.4.2.5 Range
    - .
    - .
    - .
    - 3.2.4.*q* Data element *q*
      - 3.2.4.*q*.1 Name
      - 3.2.4.*q*.2 Representation
      - 3.2.4.*q*.3 Units/Format
      - 3.2.4.*q*.4 Precision/Accuracy
      - 3.2.4.*q*.5 Range
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

Practice #3 : Specification Review

- **Review SRSs of other team members.**

- **Compare all three SRSs with respect to followings:**

항목	질문	평가결과
Understandability	문서의 내용이 잘 이해되는가?	등급(A/B/C) + 부족한 내용 + 개선된 내용
Conciseness	명료하게 작성되었나?	
Completeness	누락된 내용은 없는가?	
Consistency	모순되는 내용은 없는가?	
Accuracy	정확하지 않은 애매모호한 내용은 없는가?	
Traceability	요구사항의 Source(Origin)이 명확한가?	
Correctness (Validity)	PFR의 내용에 부합하는가?	

- **Rank the SRSs:**
  - 1st :
  - 2nd :
  - 3rd :

Practice #3 : Specification Review Checklist

Questions		SRS #1	SRS #2	SRS #3	SRS #4
Understandable	문서의 내용이 잘 이해되는가?	A/B/C			
Concise	명료하게 작성되었나?				
Complete	누락된 내용은 없는가?		C 앞에서는 OO 모드가 3개라고 했는데,뒤에는 2개에 대한 설명 밖에 없다.		
Consistent	모순되는 내용은 없는가?				
Accurate	정확하지 않은 애매모호한 내용은 없는가?				
Traceable	요구사항의 Source(Origin)이 명확한가?				
Correct (Valid)	PFR의 내용에 부합하는가?				
Total Score (100점)			95점		
Total Ranking			2등		

판정된 등급(A/B/C)에 대한 근거를 자세하게 설명합니다.

항목별로 적절한 점수, 가중치를 만들어서 총합을 계산합니다.

STRUCTURED DESIGN (SD)

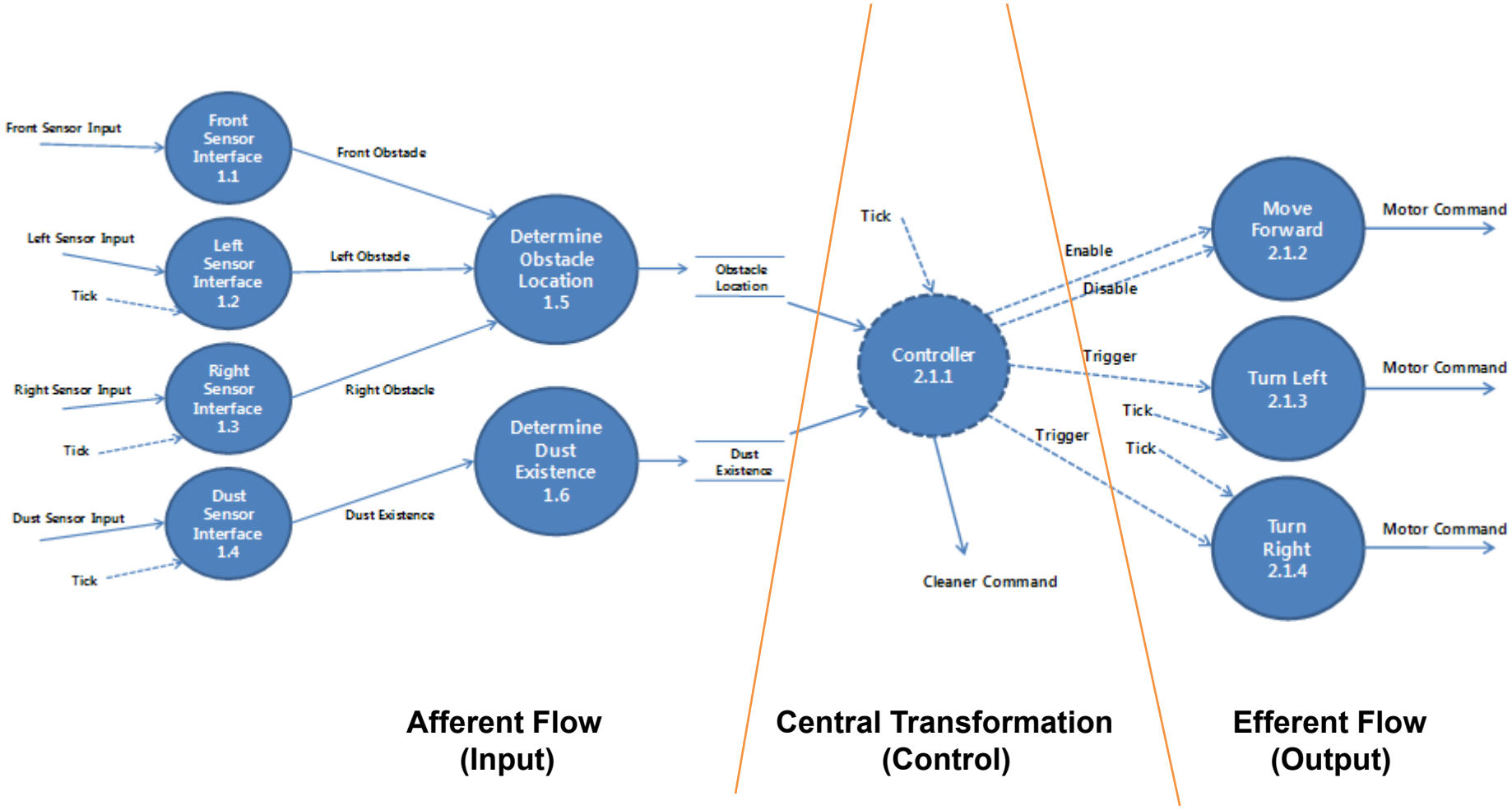
Structure Charts

- **Structured Design (SD)**

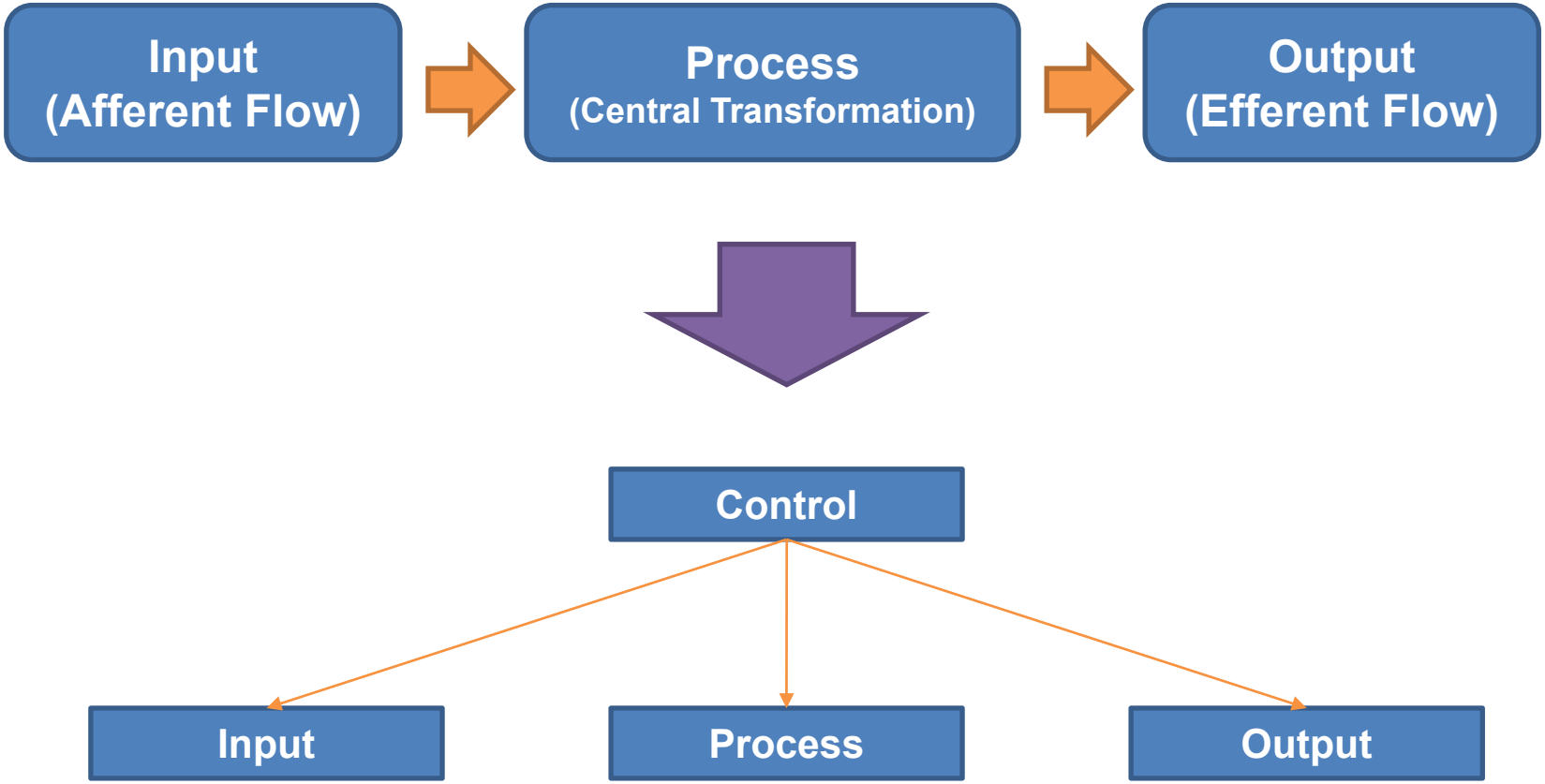
- Functional decomposition (Divide and Conquer)
  - Information hiding
  - Modularity
  - Low coupling
  - High internal cohesion

- Needs a transform analysis.

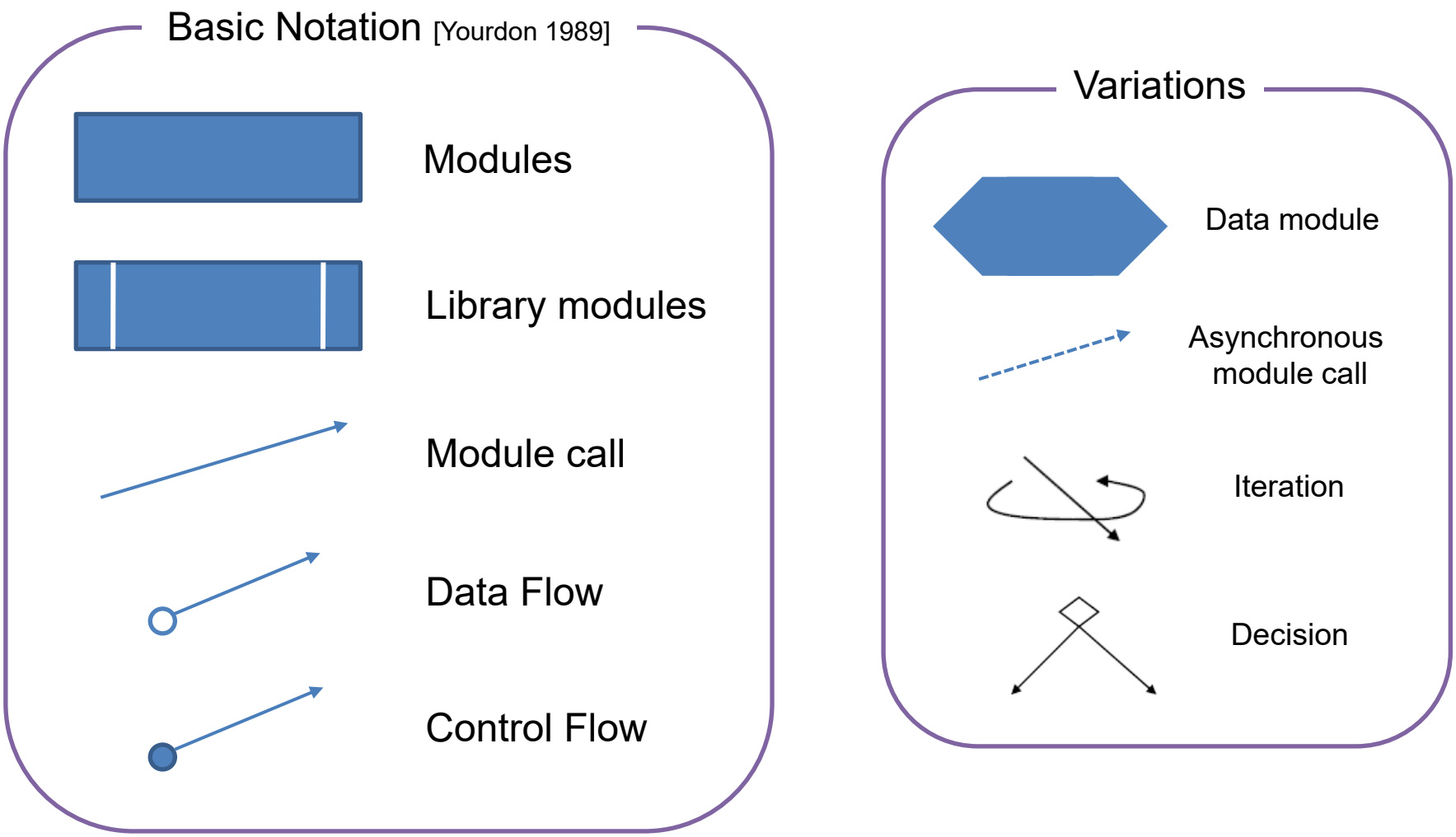
Structured Charts – Transform Analysis



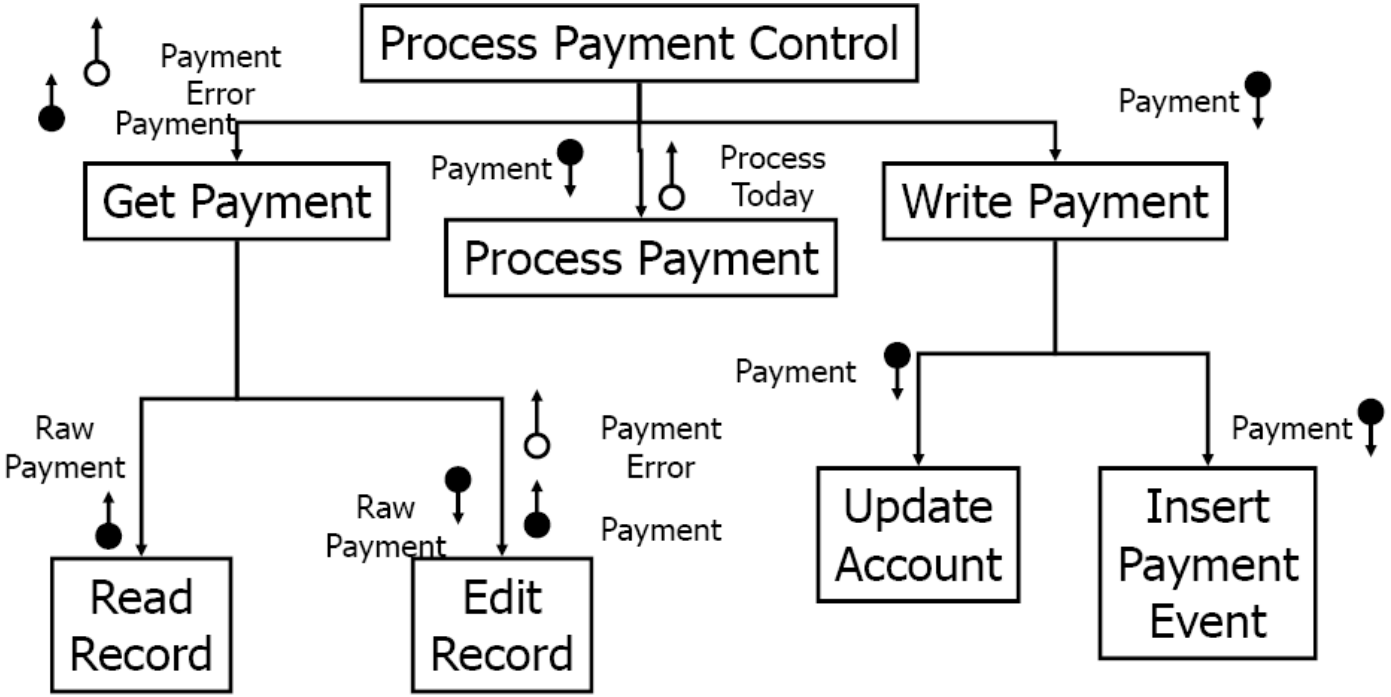
Structured Charts – Transform Analysis



Structured Charts – Notation

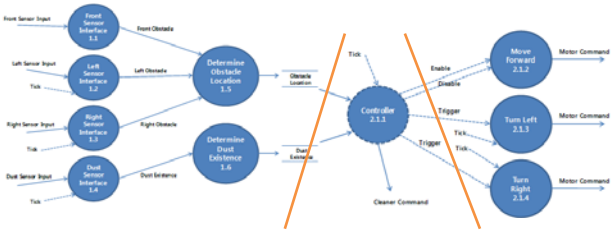


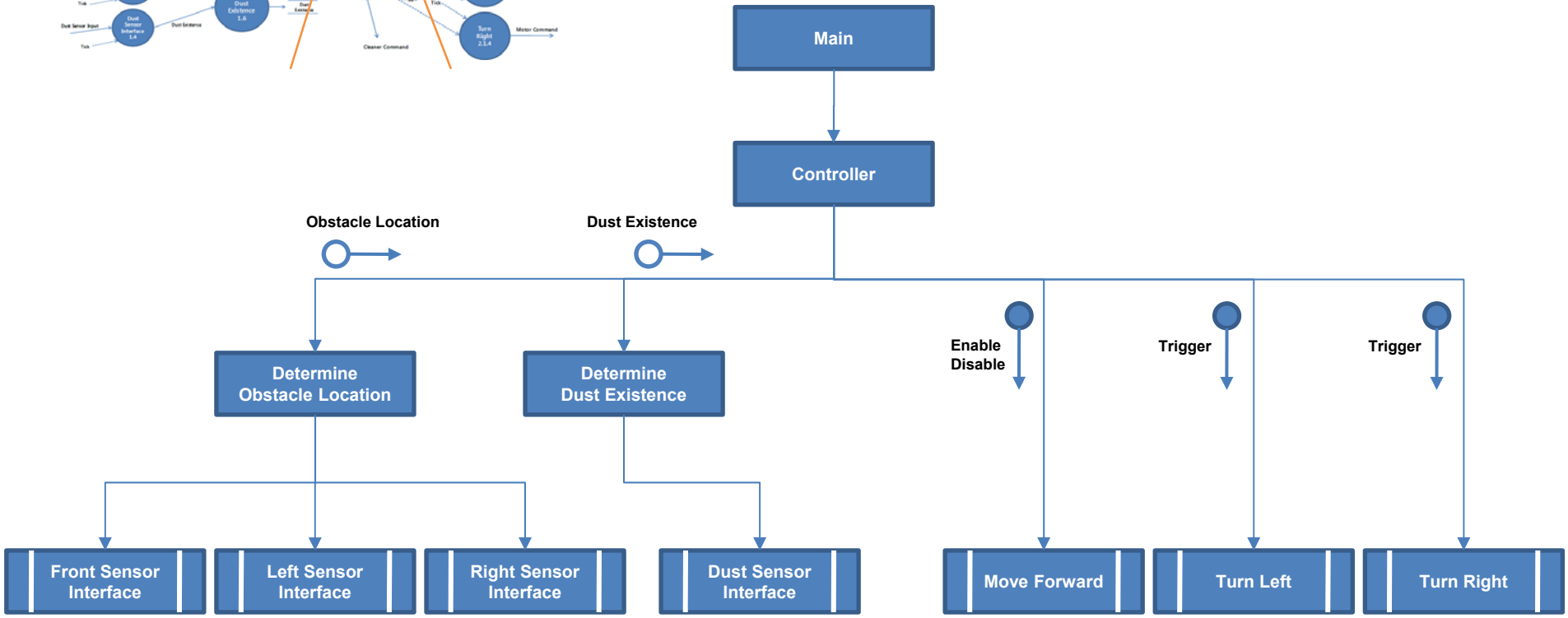
Structured Charts - Example



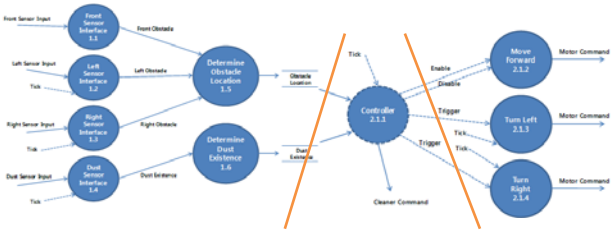
Zhou Qun, Kendra Hamilton, and Ibrahim Jadalowen (2002)

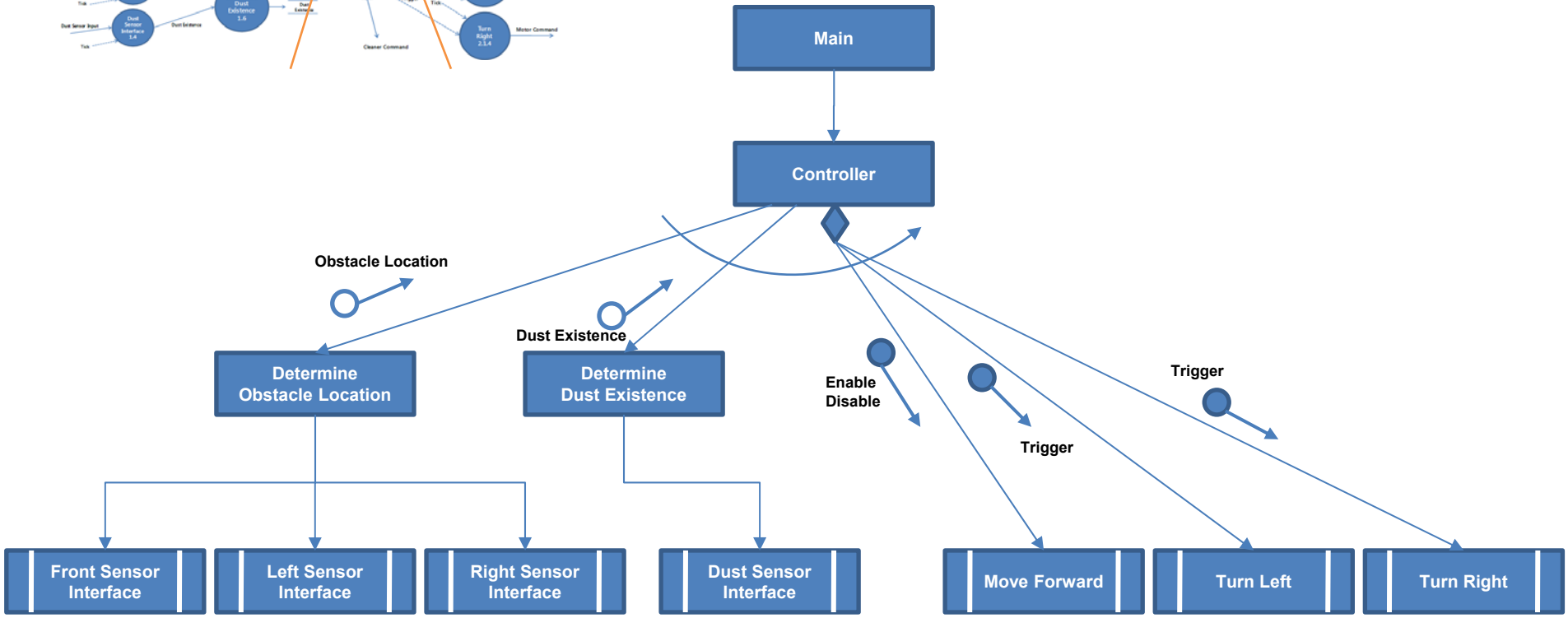
Structured Charts – RVC Example (Basic)





Structured Charts – RVC Example (Advanced)

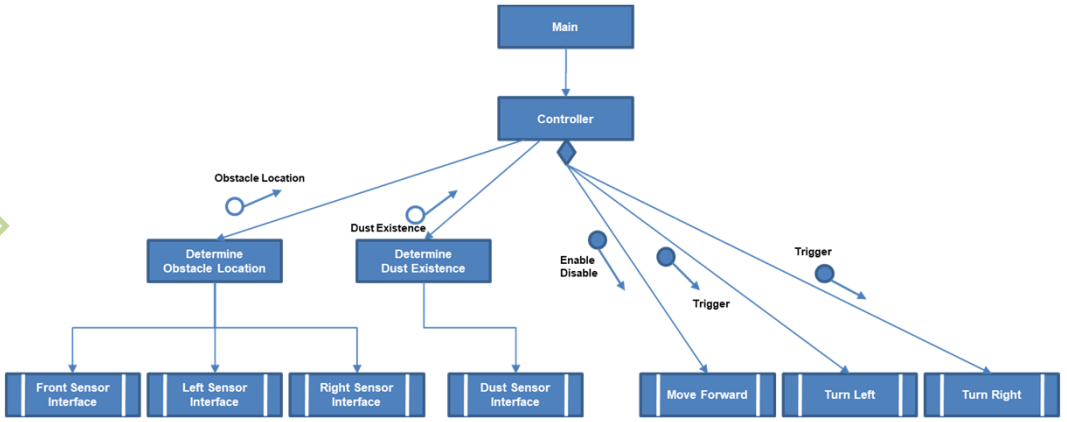




Practice #4 : Structured Design

- **Complete the RVC Design in more details.**
  - Complete a full version of Structure Charts on your own.
    - Use the 1st ranked SRS in Practice #3

The 1st ranked SRS for RVC Control SW



Structured Chart

Pros of SASD

- Has distinct milestones, allowing easier project management tracking.
- Very visual – easier for users/programmers to understand
- Makes good use of graphical tools
- Well known in industry
- A mature technique
- Process-oriented way is a natural way of thinking
- Flexible
- Provides a means of requirements validation
- Relatively simple and easy to read

Cons of SASD

- Ignores non-functional requirements.
- Minimal management involvement
- Non-iterative – waterfall approach
- Not enough use-analysts interaction
- Does not provide a communication process with users.
- Hard to decide when to stop decomposing.
- Does not address stakeholders' needs.
- Does not work well with Object-Oriented programming languages.

When to use SASD?

- Well-known problem domains
- Contract projects where SRS should be specified in detail
- Real-time systems
- Transaction processing systems
- Not appropriate when time to market is short.

- In recent years,
  SASD is widely used in developing real-time embedded systems.

SASD vs. OOAD

- Similarities
  - The both have started off from programming techniques.
  - The both use graphical design and tools to analyze and model requirements.
  - The both provide a systematic step-by-step process for developers.
  - The both focus on the documentation of requirements.

- Differences
  - SASD is process-oriented.
  - OOAD is data(object)-oriented.
  - OOAD encapsulates as much of the system's data and processes into objects,
  - While SASD separates them as possible as it can.

Summary

- SASD is a process-driven software analysis technique.
- SASD has a long history in the industry and it is very mature.
  - Provides a good documentation for requirements.
- In recent years, it is widely used for developing real-time embedded system's software with C programming language.