SASD of CFG Generator

Original : Team 6	Amender :Team 5
200811425 김평석	200711453 류진렬
200811435 신성호	200711454 윤병현
200811451 이형열	200711459 이남섭
200811454 전인서	200711463 이준하

Contents

- **Structured Analysis**
  - Statement of Purpose
  - System Context Diagram
  - Event List
  - Data Flow Diagram
  - Final State Machine
  - Data Dictionary
  - Process Specification

- **Structured Design**
  - Structured Charts

- **Appendix**
  - Middle Language
  - Parse Code
  - Construct CFG

Structured Analysis

Statement of Purpose

- Entire Statement of Purpose
- Specific Statement of Purpose

Entire Statement of Purpose[1/1]

Entire Statement of Purpose
CFG Generator is operated in CUI-based Cygwin.
CFG Generator make State lists and Edge lists from C language-program input.
After confirming the validity of C language-program input, CFG Generator makes Middle-Languages.
CFG Generator derives the CFG information from the Middle-Languages.
As the selected option, Derived CFG information is outputted on file or console.
As the system condition, proper messages are outputted on console.

Specific Statement of Purpose[1/3]

Input Statement of Purpose
Line command is the input type for executing this program. It contains Source File Name, Ouput File Name and Ouput Option.
C Program File that will be inputted in CFG Generator is single file that has *.c filename extension. It has no more 200 letters.
The C Program File code has Main function.
The C Program File code must not have any Pointer.
The C Program File code must not have any header and library defined by user.
The C code input file is must be a perfect file that has no syntax error.

Specific Statement of Purpose[2/3]

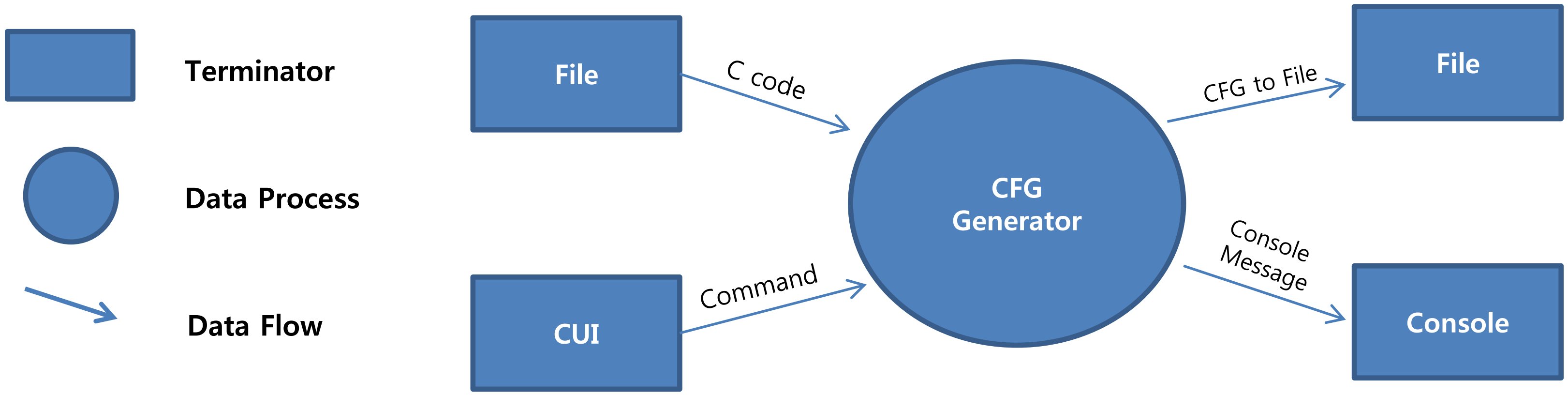
Parsing Statement of Purpose
Classify the C Code file using the defined token.
Convert the classified C Code to Middle Language.
CFG file doesn’t have branch at the macro.
CFG file dosen’t have branch at the trinomial operator(x ? y : z)
CFG file dosen’t have branch at user-defined-function.

Specific Statement of Purpose[3/3]

Constructing CFG Statement of Purpose
Generate the block lists of the CFG from Middle Language.
Using the block Lists, generate the edge Lists.

Output Statement of Purpose
When Command Error or File Error is occurred, Error Message and Help Message will be outputted on console.
As the output option inputted by user, CFG Generator outputs the CFG on file or on console using CFG component List.

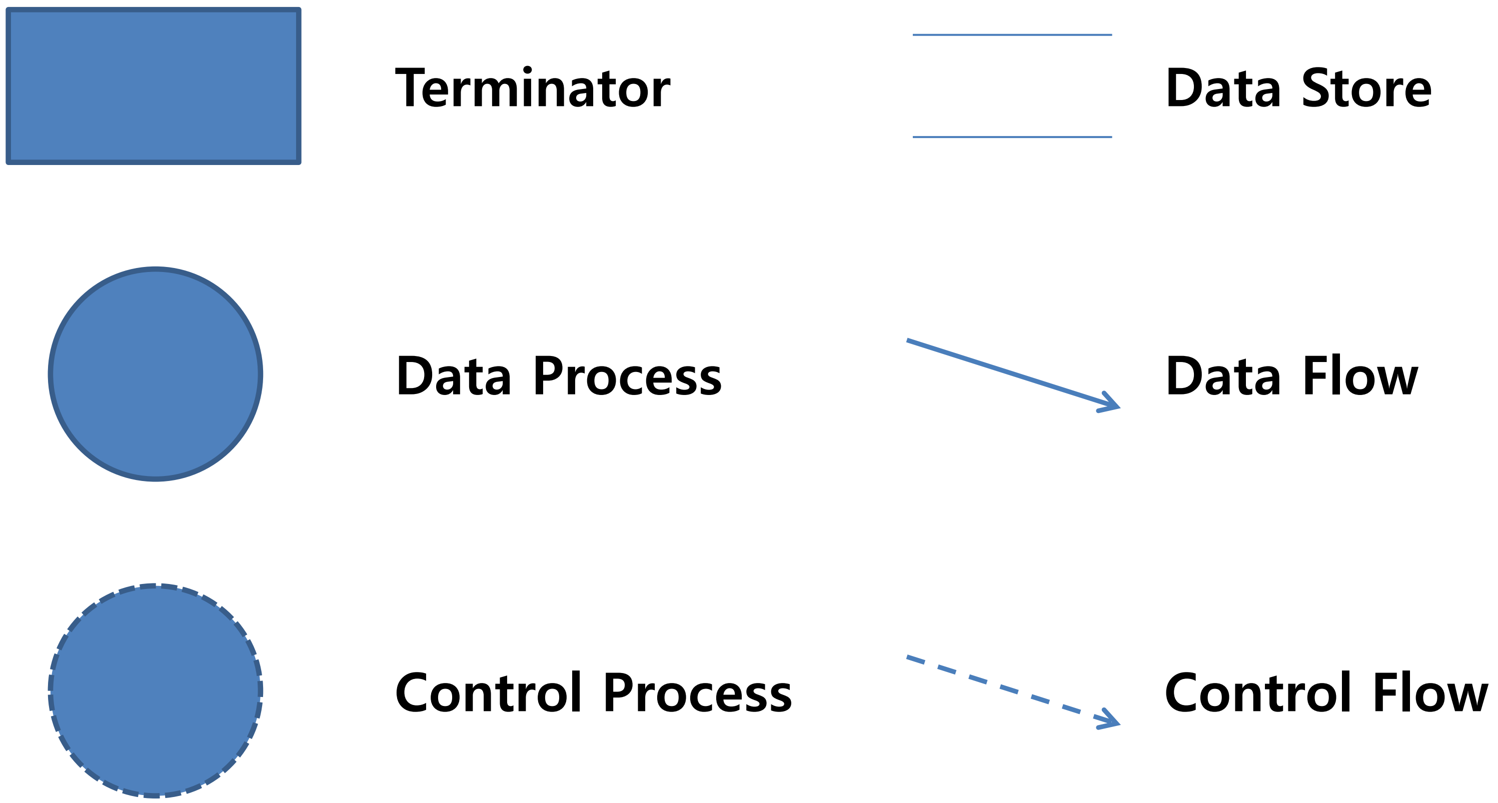
System Context Diagram & Event List



Input/Output Event	Description	Format/Type
Command	String includes Input File Name, Output File Name and Output Option. Ex) ./cfg [option] [input file name] [output file name]	string
C Code	C source code to convert CFG. The File must have *.c extension name. Source code produced by C standard .	*.c
CFG to File	CFG to be printed file, written by string format.	string
Console Message	CFG and Message to be outputted console. This message is about each situation.	CmdError / FileError / HelpMessage / GenerateMessage / CFG / OutFileName

Data Flow Diagram

Notation of Data Flow Diagram



Terminator

Data Store

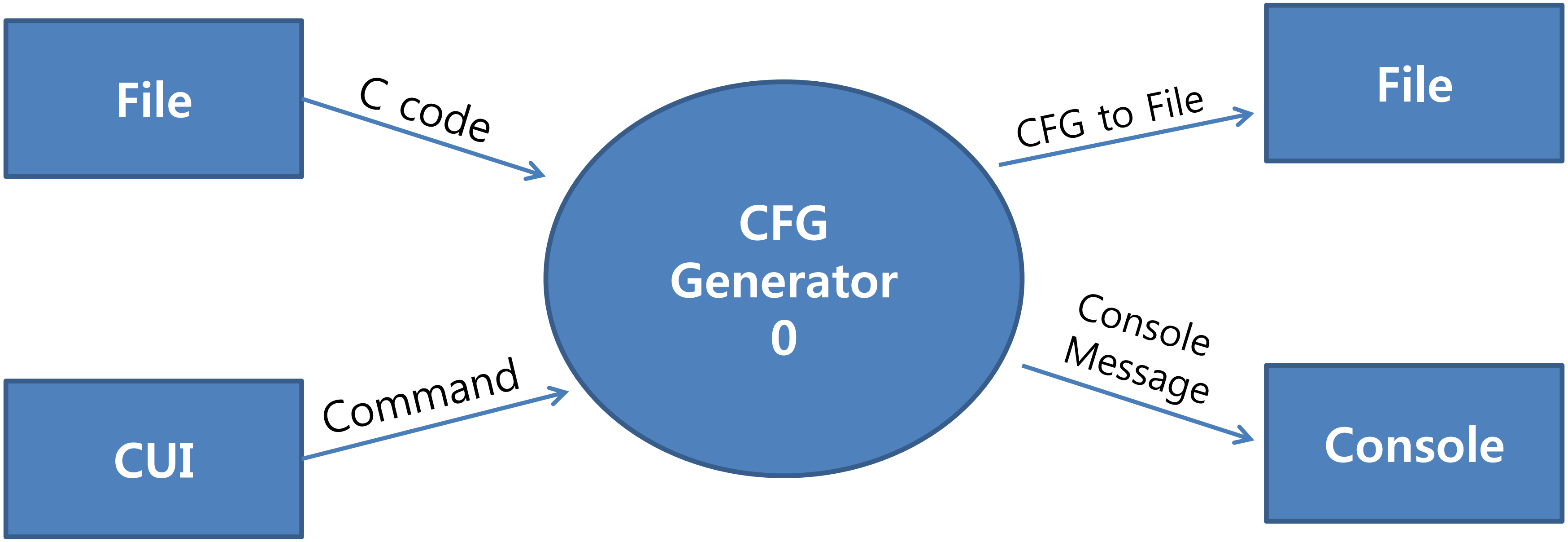
Data Process

Data Flow

Control Process

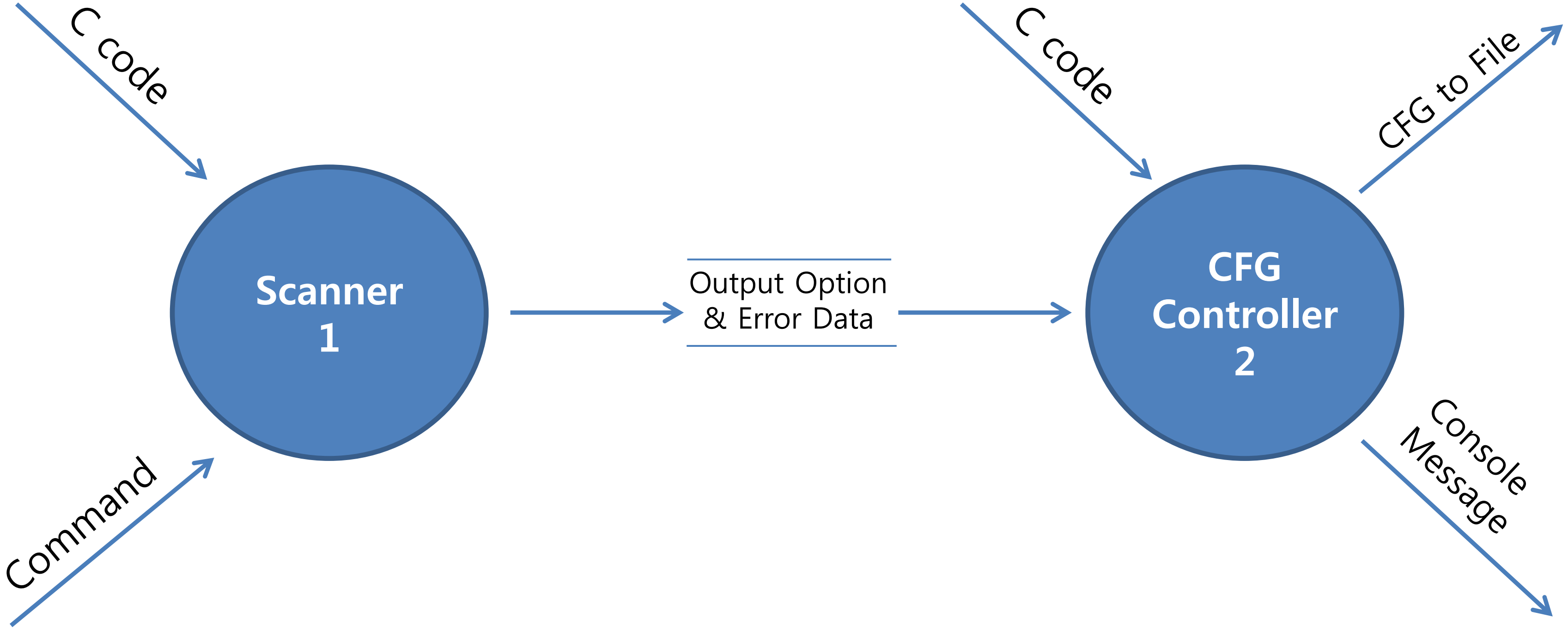
Control Flow

Data Flow Diagram[1/5] - Level 0



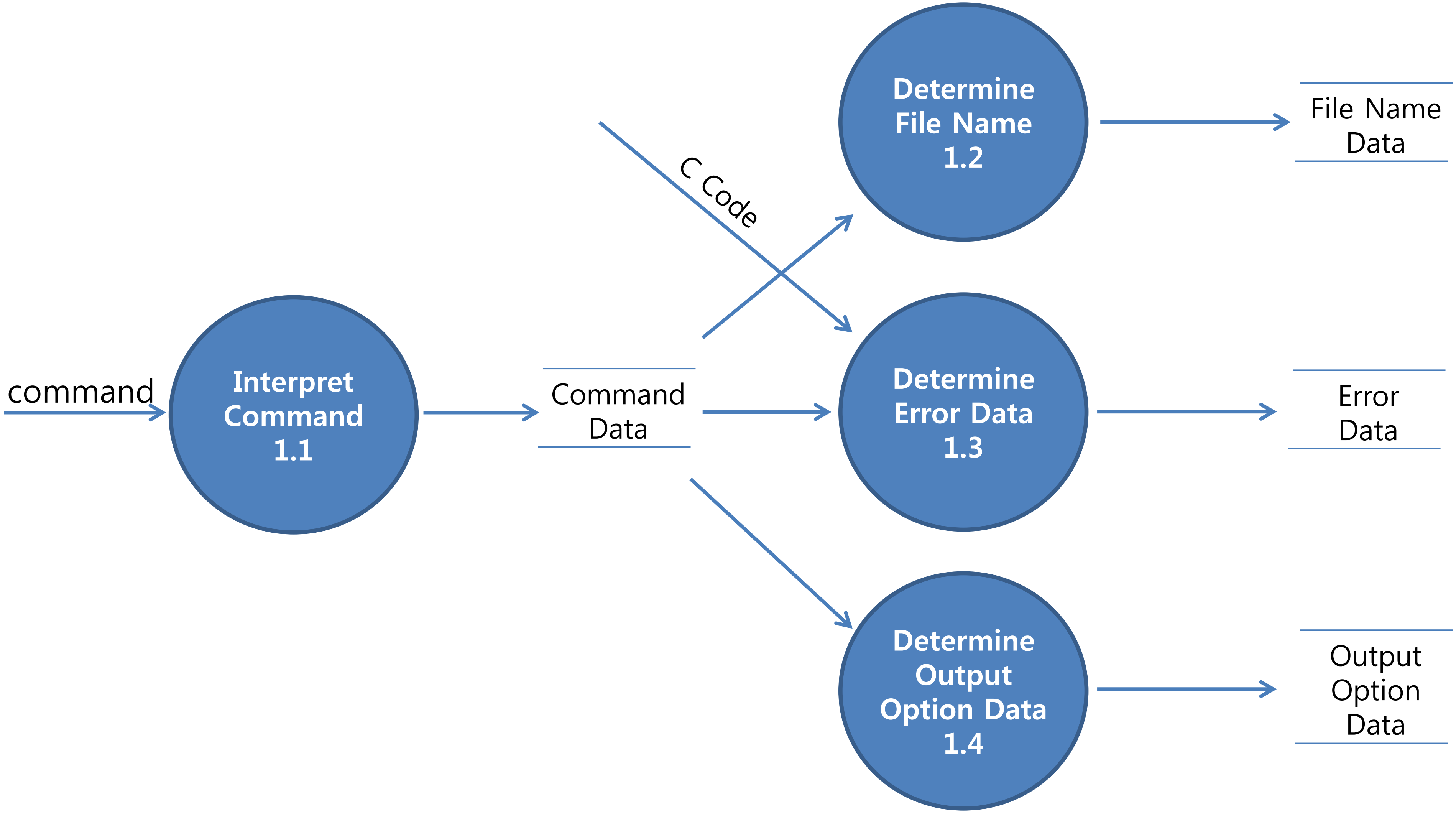
Data Dictionary

Data Flow Diagram[2/5] - Level 1



Data Dictionary

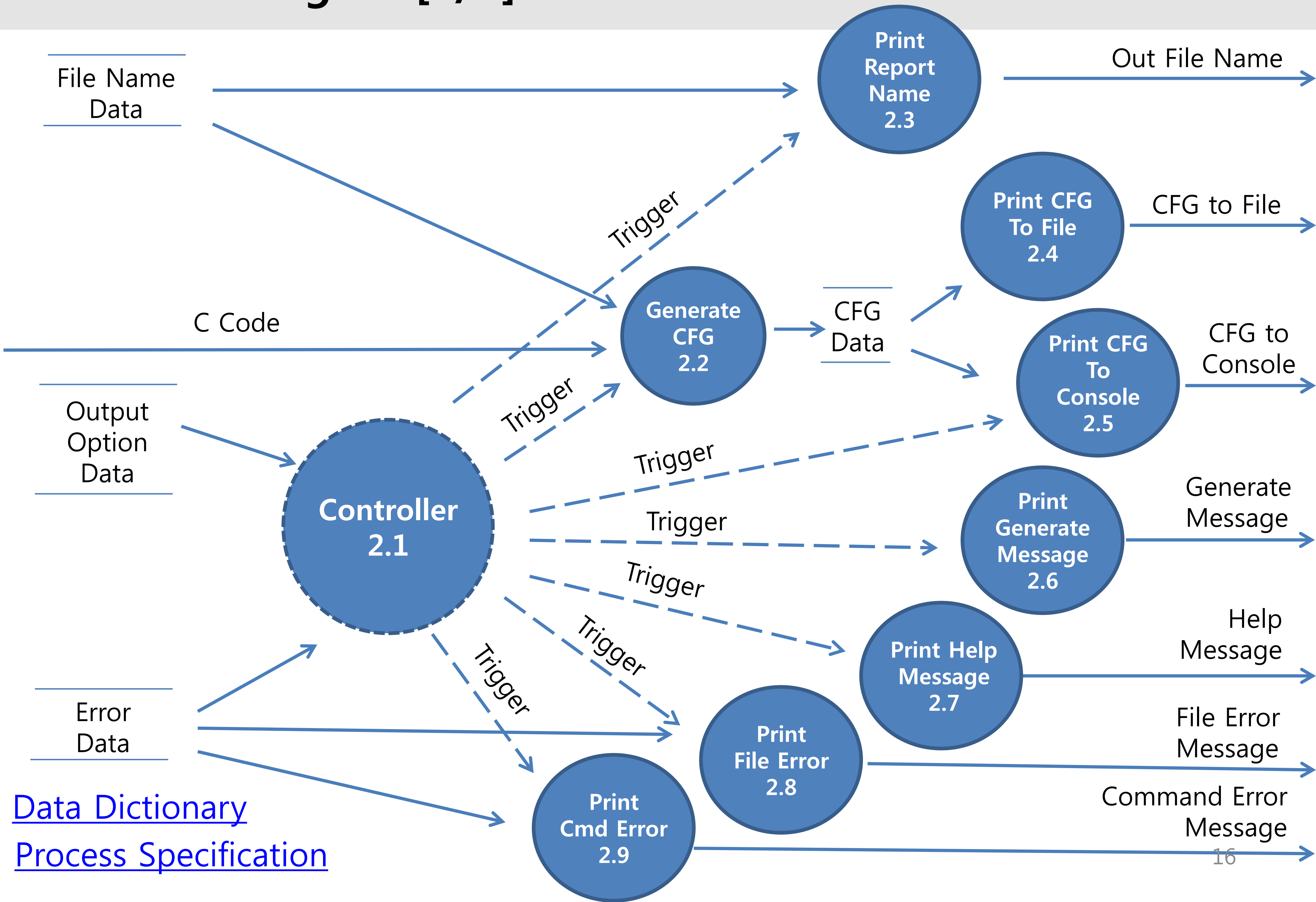
Data Flow Diagram[3/5] - Level 2



Data Dictionary   Process Specification

Data Flow Diagram[4/5] – Level 2

This is an original slide.
Next slide is a modified slide.

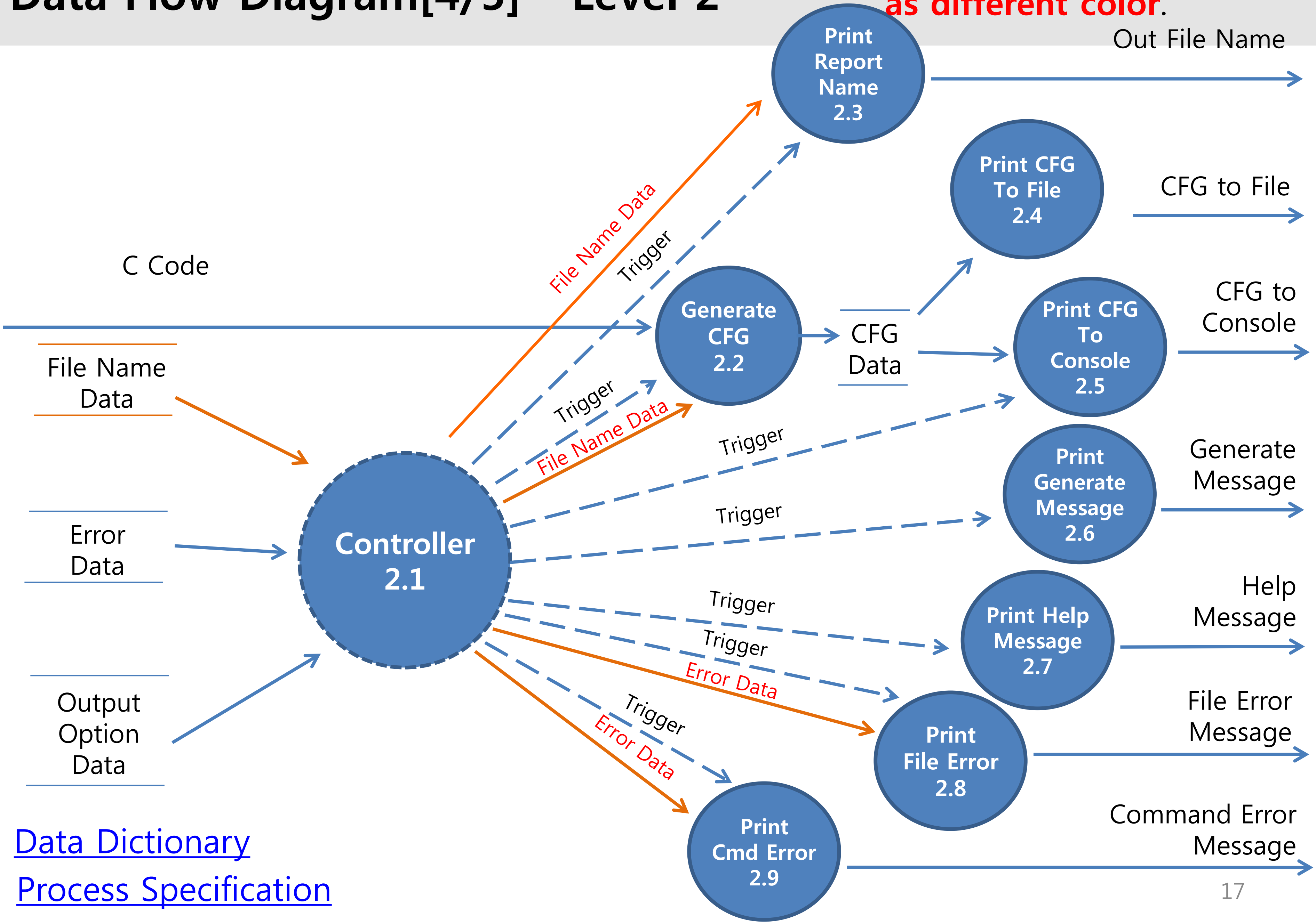


Data Dictionary

Process Specification

Data Flow Diagram[4/5] - Level 2

Modified details are shown as different color.



[Data Dictionary](#)

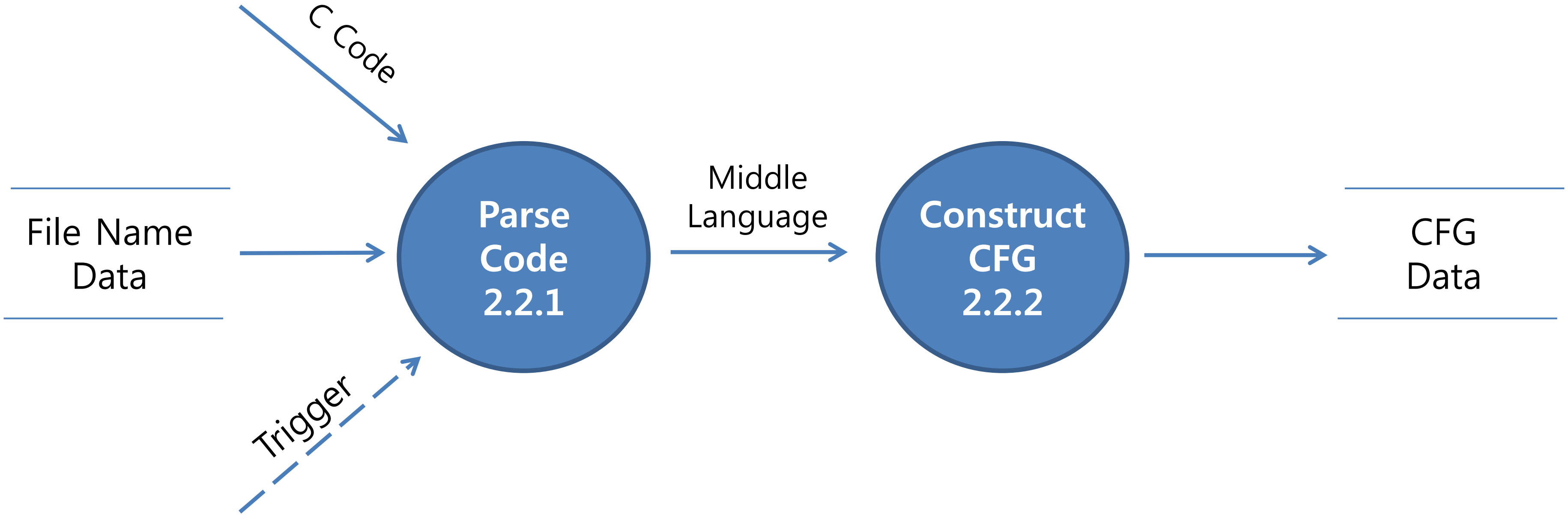
[Process Specification](#)

Explanation of modified slide (slide number 17)

On the DFD of the original slide, **the file name data** doesn’t go through the controller. It is connected with **‘Generate CFG**′ process and **‘Print report name’** process directly. But, on the Structured Chart of the SD, controller calls the **‘Determine file name’** process. **File name data** from the process**(Determine file name process)** is also delivered to controller.
It generate discordance between DFD and Structured Chart.
So we modify the DFD. On the DFD of the modified slide, **File name data** go through the controller, and it is delivered to **‘Generate CFG’** process and **‘Print report name’** process.

On the DFD of the original slide, **Error data** and **Output Option data** are delivered to controller. But it is not expressed where to go. Some connections between the **data store** and **process** are connected directly. We modified these things according to adequate flow of the data.

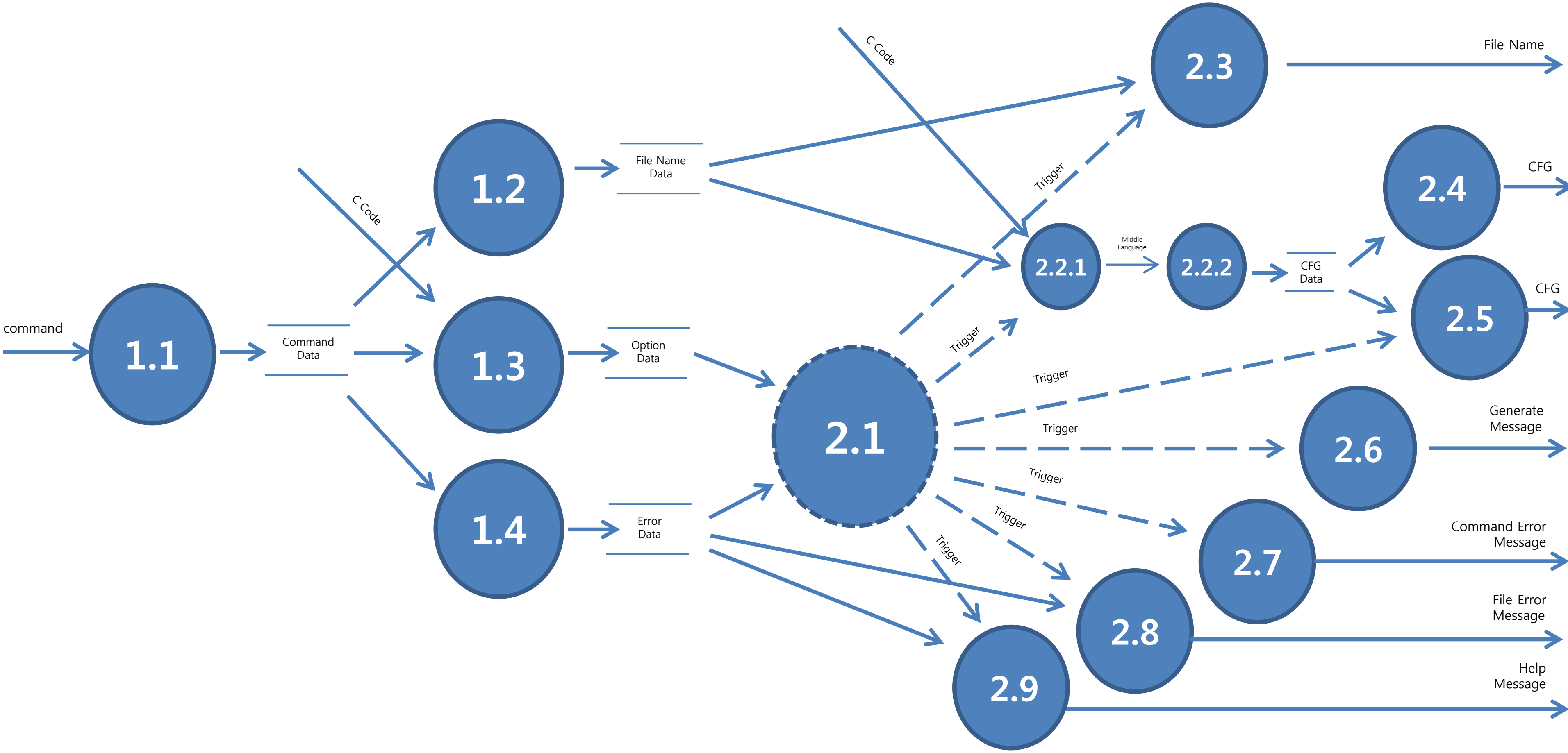
Data Flow Diagram[5/5] - Level 3



Process Specification

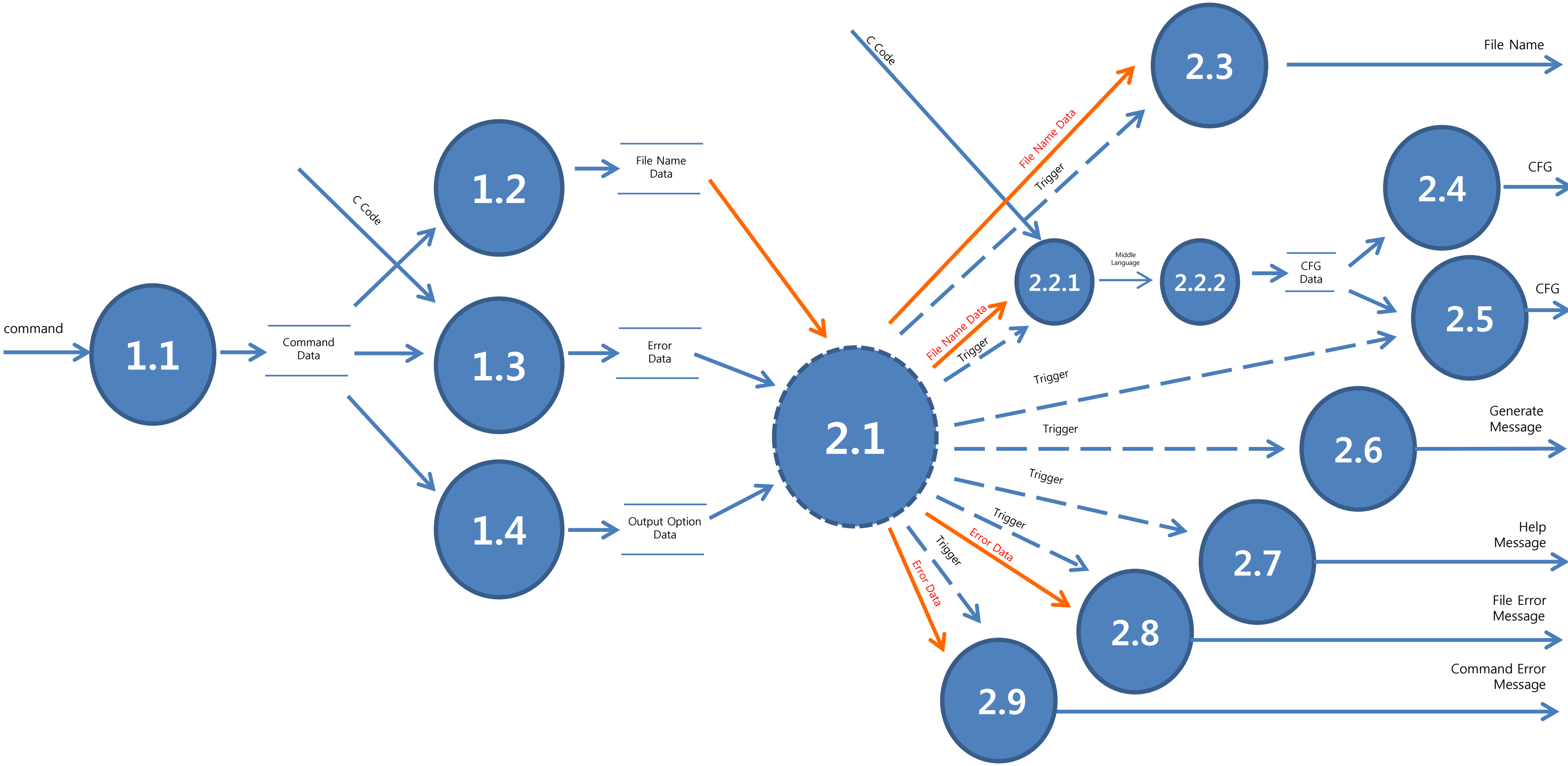
Data Flow Diagram - Final

This is an original slide.
Next slide is a modified slide.



Data Flow Diagram - Final

Modified details are shown as different color.



Explanation of modified slide (slide number 21)

On the DFD of the original slide, **the file name data** doesn’t go through the controller. It is connected with **‘Generate CFG’** process and **‘Print report name’** process directly. But, on the Structured Chart of the SD, controller calls the **‘Determine file name’** process. **File name data** from the process**(Determine file name process)** is also delivered to controller.
It generate discordance between DFD and Structured Chart.
So we modify the DFD. On the DFD of the modified slide, **File name data** go through the controller, and it is delivered to **‘Generate CFG’** process and **‘Print report name’** process.

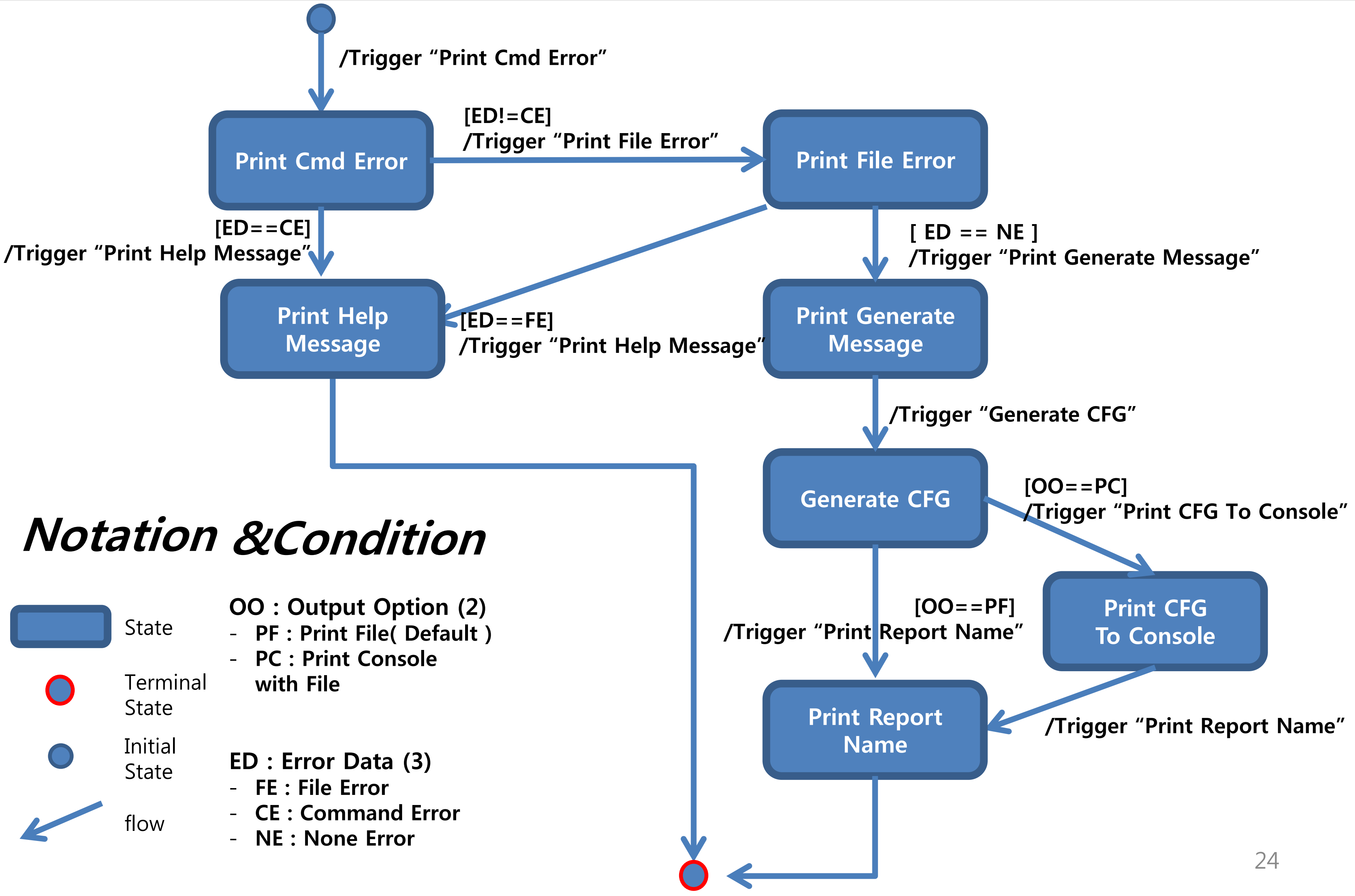
On the DFD of the original slide, **Error data** and **Output Option data** are delivered to controller. But it is not expressed where to go. Some connections between the **data store** and **process** are connected directly. We modified these things according to adequate flow of the data.

***This slide is the same with slide number 18.**

Final State Machine

Final State Machine

This is an original slide.
Next slide is a modified slide.


/Trigger "Print Cmd Error"
Print Cmd Error
[ED!=CE]
/Trigger "Print File Error"
Print File Error
[ED==CE]
/Trigger "Print Help Message"
[ED == NE]
/Trigger "Print Generate Message"
Print Help Message
[ED==FE]
/Trigger "Print Help Message"
Print Generate Message
/Trigger "Generate CFG"
Generate CFG
[OO==PC]
/Trigger "Print CFG To Console"
[OO==PF]
/Trigger "Print Report Name"
Print CFG To Console
Print Report Name
/Trigger "Print Report Name"

Notation &Condition

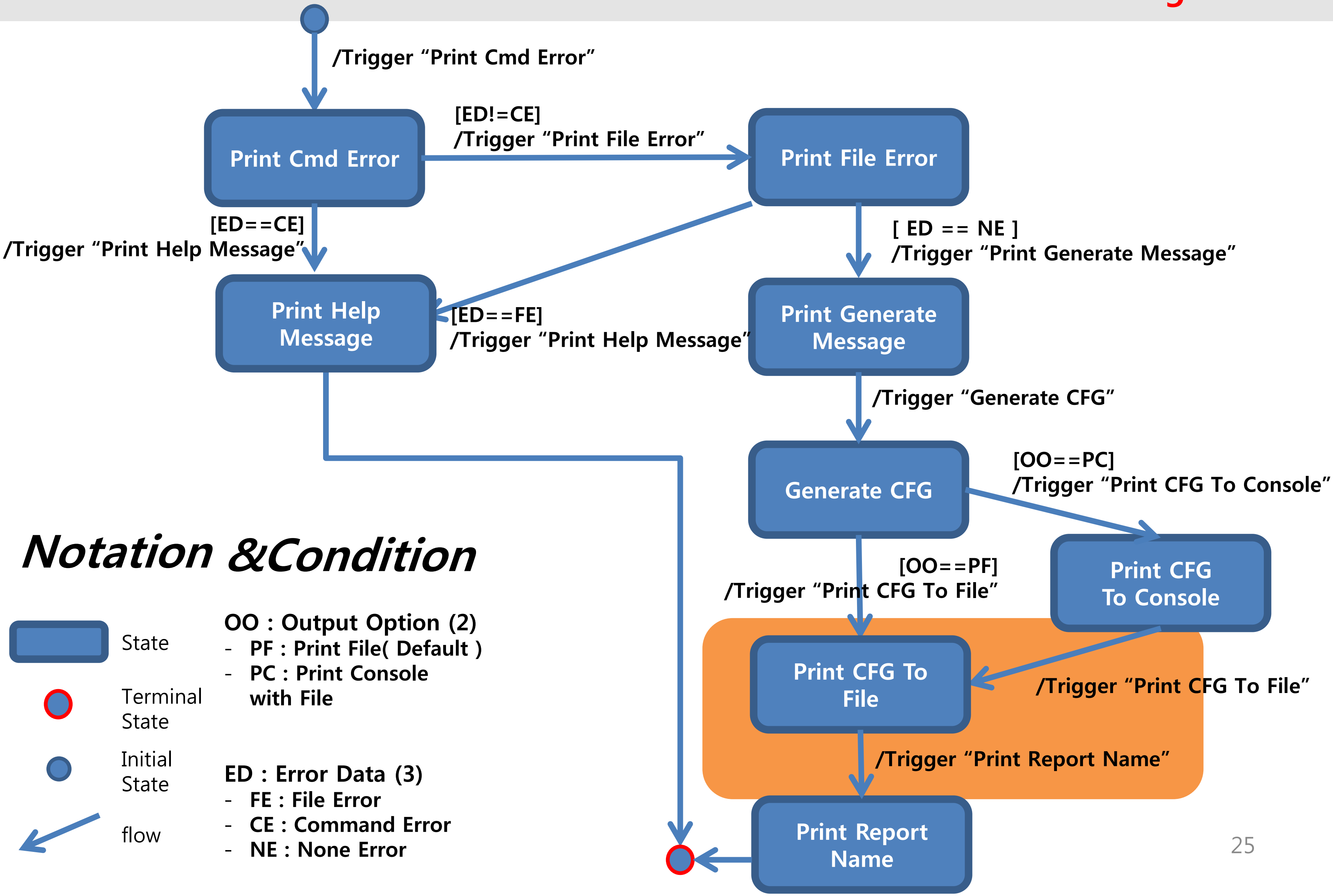
▬	State
●	Terminal State
●	Initial State
↙	flow

OO : Output Option (2)
- **PF : Print File(Default)**
- **PC : Print Console with File**

ED : Error Data (3)
- **FE : File Error**
- **CE : Command Error**
- **NE : None Error**

Final State Machine

Modified details are shown in different colored background.



/Trigger “Print Cmd Error”

Print Cmd Error

[ED!=CE]
/Trigger “Print File Error”

Print File Error

[ED==CE]
/Trigger “Print Help Message”

Print Help Message

[ED==FE]
/Trigger “Print Help Message”

[ED == NE]
/Trigger “Print Generate Message”

Print Generate Message

/Trigger “Generate CFG”

Generate CFG

[OO==PC]
/Trigger “Print CFG To Console”

Print CFG To Console

[OO==PF]
/Trigger “Print CFG To File”

Print CFG To File

/Trigger “Print CFG To File”

/Trigger “Print Report Name”

Print Report Name

Notation &Condition

▬ State

◉ Terminal State

● Initial State

↙ flow

OO : Output Option (2)
- PF : Print File(Default)
- PC : Print Console with File

ED : Error Data (3)
- FE : File Error
- CE : Command Error
- NE : None Error

Explanation of modified slide (slide number 25)

On the Final State Machine of the original slide, there is no process of **'print CFG to File'.** The FSM of original slide means that after finishing **'Generate CFG'** process, it doesn't print CFG to File. It is not satisfied with **basic requirement**.

So we add one more state(**'print CFG to File'**) in modified FSM.
The process of **'print CFG to File'** is necessary after **'Generate CFG'** process.
Regardless of **Output Option data**, this process must be operated.
(if the generating is successed correctly.)

Data Dictionary

Data Dictionary Level 0

DFD

Input/Output Event	Description	Format/Type
Command	String includes Input File Name, Output File Name and Output Option. Ex) ./cfg [option] [input file name] [output file name]	string
C Code	C source code to convert CFG. The File must have *.c extension name. Source code produced by C standard .	*.c
CFG to File	CFG to be printed file, written by string format.	string
Console Message	CFG and Message to be outputted console. This message is about each situation.	CmdError / FileError / HelpMessage / GenerateMessage / CFG / OutFileName

Data Dictionary Level 1

DFD

Input/Output Event	Description	Format / Type
Output option	Save the status of Output Option. String type variable OO contains Output Option Data. The variable OO may have two data type. *PF(print only file), PC(print console with file). Default data type is 'PF'. Ex string OO = "PC";	string
Error Data	Save the Error Data. String type variable ED contains Error Data. *CE (Command Error : When command line input has wrong syntax and inadequate form.), FE (File Error : When the format of the input file is wrong or the input file is not exist.) NE (None Error : When there is no error.) Ex) string ED = "NE";	string

Data Dictionary Level 2

DFD

Input/Output Event	Description	Format/Type
Command Data	Save the Command Data. If the command input is correct, it will be devided into several string type data and saved as struct type. Ex) ./cfg [option] [input file name] [output file name]	Struct
File Name Data	Input File Name and Output File Name from command data is saved in struct.	Struct

There were another sentence about command data('만약 적절하지 않은 명령어가 들어오면 모두 NULL값이 저장된다.') in the original slide.
If wrong command is inputted in CFG Generator, this program going to Help Message Process and quit. So, that requirement is not necessary for this program. So we remove the sentence.

Data Dictionary Level 2

DFD

Input/Output Event	Description	Format/Type
Error Data	After checking the Command Error and File Error from Command Data, adequate status of data will be saved as a string type data. CE (Command Error :When command line input has wrong syntax and inadequate form.), FE (File Error : When the format of input file is wrong or the input file is not exist.) NE (None Error : When there is no error) One of them will be saved in string type variable ED. Ex) string ED = “NE”;	string
Output Option Data	The string type variable OO contains output option data from Command Data. One of the two data(PF,PC) is saved in OO. Ex) string OO = “PC”;	string

Data Dictionary Level 2

DFD

Input/Output Event	Description	Format/Type
CFG Data	Save CFG Data. From inputted C code, node list & edge list are saved in struct-type-data. Based on this data, CFG will be outputted on File or Console.	Struct
Command Error	Whether the cmd error occurred or not.	String
Output File Name	Output file name for result CFG file.	string
Input File Name	Input file name for C code input file.	
File Error	Whether File exist or not.	String
Help Message	*Help Message* to help user for correct command input.	String
Generate Message	The complete message of generating.	String
CFG (Console)	CFG that will be outputted to console.	string

Data Dictionary Level 3

DFD

Input/Output Event	Description	Format/Type
Middle Language	It stores Middle-Language produced by parsing process . It is a data for producing CFG component by analyzing the source code. It stores each branch information as a list or array in struct.	Struct List / Array

Process Specification

Process Specification Level 2

DFD

Reference	1.1
Name	Interpret Command
Input	Command
Output	Command Data
Process Description	"Command" come in this Process from CUI. The data in the command consist of string is divided into three part of string by this process, "Interpret Command". After that work, this process save Command Data Systematically .

Process Specification Level 2

DFD

Reference	1.2
Name	Determine File name
Input	Command Data
Output	File Name Data
Process Description	"Command Data" come in this Process from The storage. The data have "File Name Data" that is consist of Report Name(Output File Name) and C Code File. This Process extract the data from "Command Data" And save File Name data Systematically for convenience of Controller.

Process Specification Level 2

DFD

Reference	1.3
Name	Determine Error Data
Input	Command Data
Output	Error Data
Process Description	"Command Data" come in this Process from The storage. The data have "Error Data" that is consist of File Error Data and Cmd Error Data. This Process try to open the C Code File to check the File's validity. And save Error data Systematically for convenience of Controller.

Process Specification Level 2

DFD

Reference	1.4
Name	Determine Output Option Data
Input	Command Data
Output	Output Option Data
Process Description	"Command Data" come in this Process from The storage. The data have "Output Option Data" that have two condition, Print File and Print Console. The Print File is default option when any option does not income. After extract Output Option Data from Command Data, This process save output option data for convenience of Controller.

Process Specification Level 2

DFD

Reference	2.1
Name	Controller
Input	Error Data, Output Option Data
Output	Trigger
Process Description	“Error Data” and “Output Option data” come in this process for divergence. Controller can make a decision with that data for p rocess of Program. The Operation of this Process is represented to “state machine” specifically.

Process Specification Level 2

DFD

Reference	2.3
Name	Print Report Name
Input	Trigger, File Name Data
Output	File Name
Process Description	After Triggered, this process print report name as meaning of finishing print CFG and succeeding this program.

Process Specification Level 2

DFD

Reference	2.4
Name	Print CFG To File
Input	Trigger, CFG Data
Output	CFG
Process Description	After Triggered, this process, “Print CFG To File” print CFG Data to File. Since CFG can be printed on two stream, File and Console, this process is named ‘Print CFG To File’ not ‘Print CFG’. If the CFG generating is completed, this process must be operated regardless of ‘Output Option’.

Process Specification Level 2

DFD

Reference	2.5
Name	Print CFG To Console
Input	Trigger , CFG Data
Output	CFG
Process Description	After Triggered, this process ,'Print CFG To Console', print CFG Data to Console. This process operate selectively, the contrary "Print CFG To File" operate unconditionally after generate CFG.

Process Specification Level 2

DFD

Reference	2.6
Name	Print Generate Message
Input	Trigger
Output	Generate Message
Process Description	After Triggered, this process print Generate Message meaning of ‘No Error’ and starting ‘Generate CFG Data’

Process Specification Level 2

DFD

Reference	2.7
Name	Print Cmd Error
Input	Trigger
Output	Command Error Message
Process Description	When Cmd Error occur, The controller trigger this process to print command error message. After print command error message, controller trigger 'print help' process . If there is no error in command, this process does not print error message.

Process Specification Level 2

DFD

Reference	2.8
Name	Print File Error
Input	Trigger
Output	File Error Message
Process Description	When File Error occur, The controller trigger this process to print file error message. After print command error message, controller trigger 'print help' process. If there is no error in command, this process does not print error message.

Process Specification Level 2

DFD

Reference	2.9
Name	Print Help Message
Input	Trigger
Output	Help Message
Process Description	Whenever Error occur, this process is triggered. This process help user to operate ‘CFG Generator’ correctly. After print help message, this program is terminated.

Process Specification Level 3

DFD

Reference	2.2.1
Name	Parse Code
Input	Trigger, File Name data, C Code
Output	Middle Language
Process Description	When Error isn't occurred, this process read 'C code' from File. This process translates C code to middle language, reading each line of c code.

Process Specification Level 3

DFD

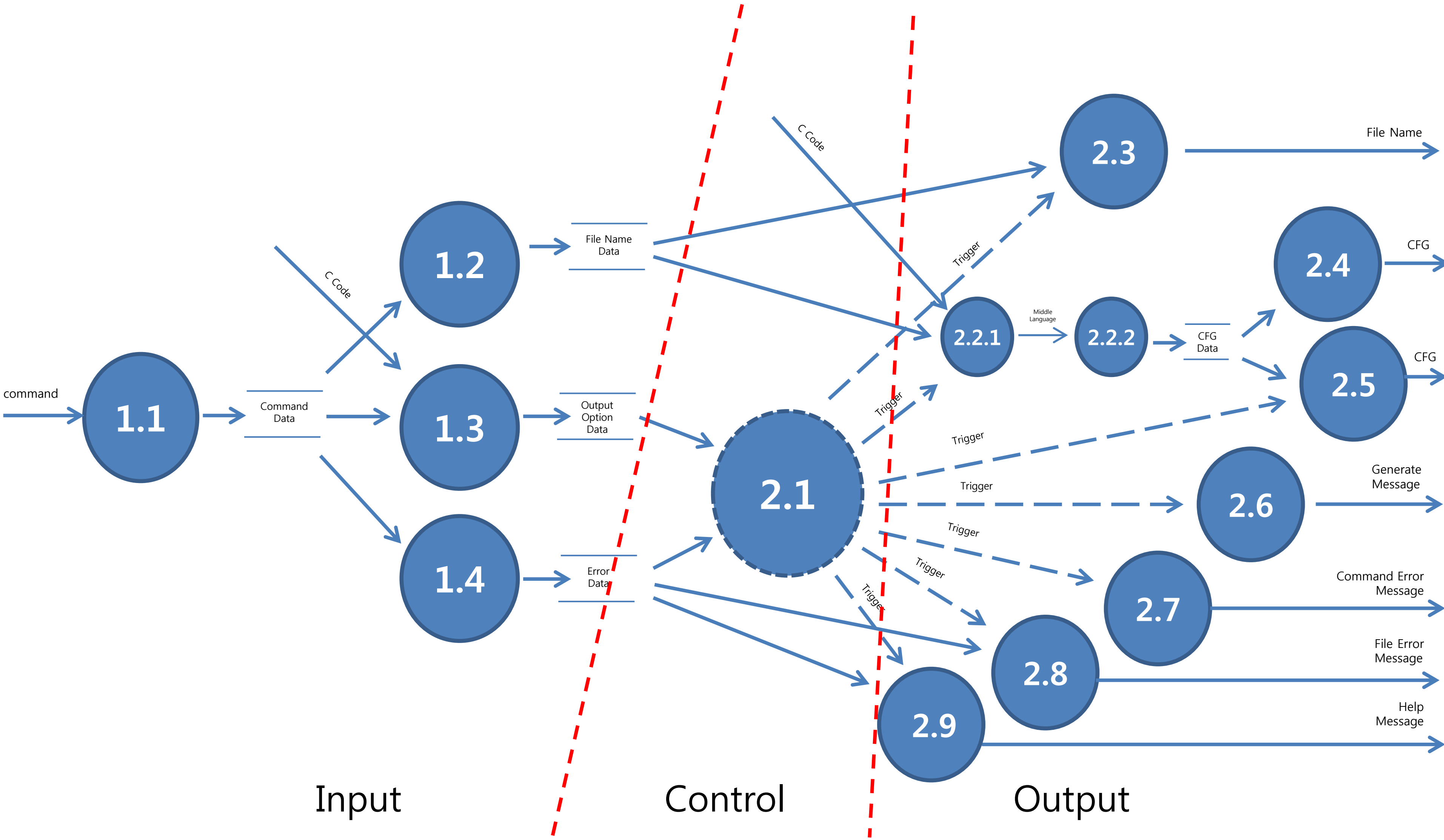
Reference	2.2.2
Name	Construct CFG
Input	Middle Language
Output	CFG Data
Process Description	After Middle Language is generated, This process construct CFG with middle language. Finishing the constructing, this process save CFG Data for print report and console.

Structured Design

- Structured Charts

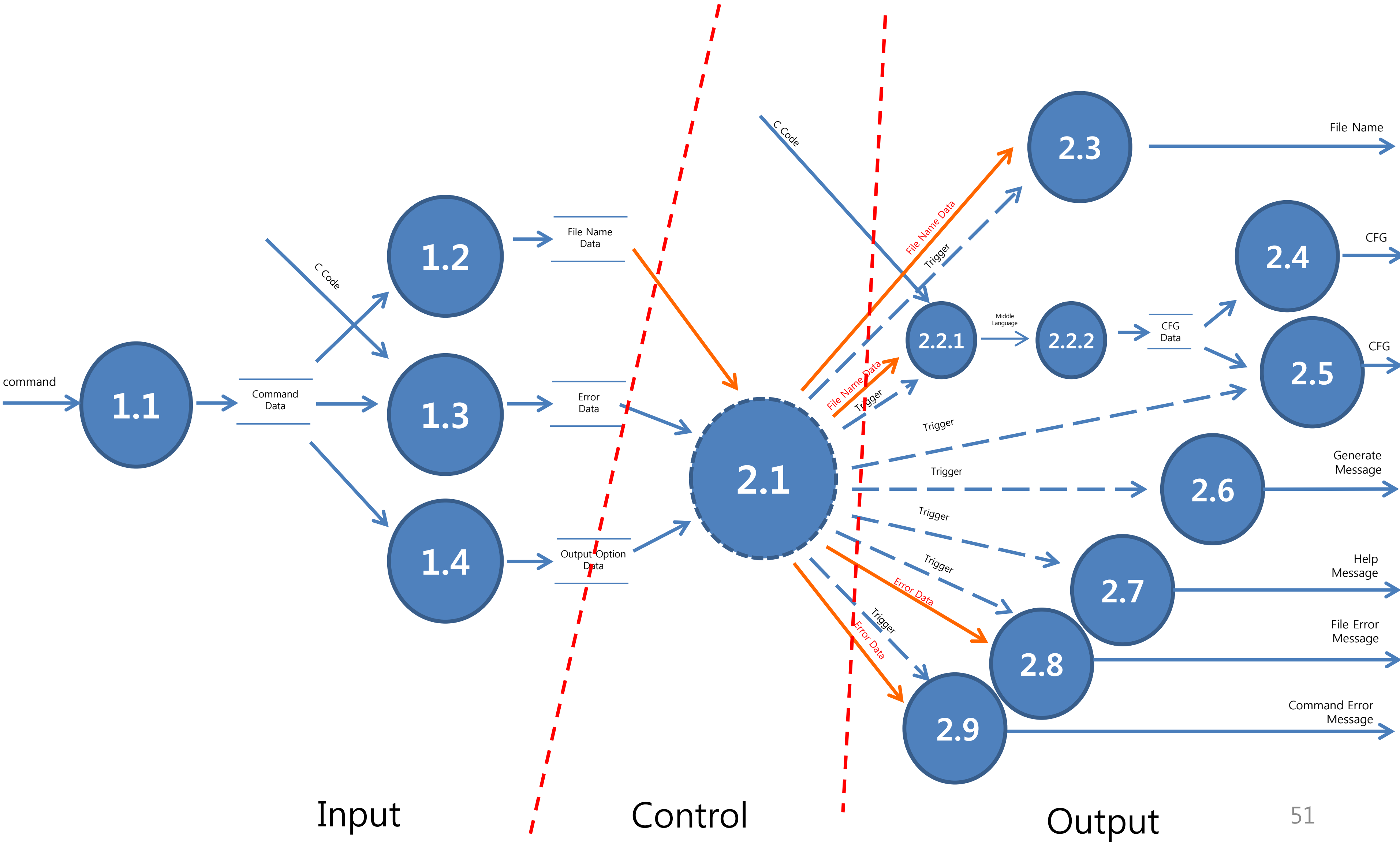
Transform Analysis

This is an original slide.
Next slide is a modified slide.



Transform Analysis

Modified details are shown as different color.



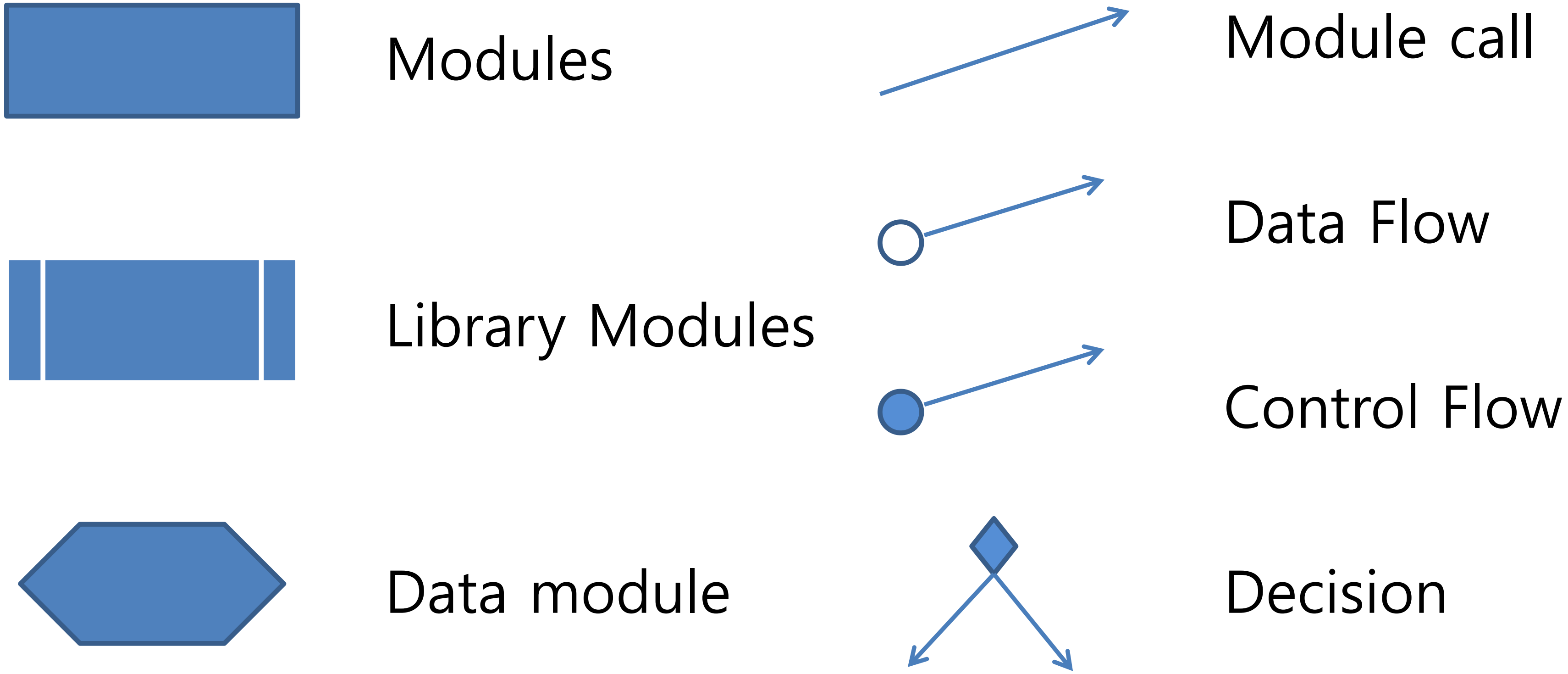
Explanation of modified slide (slide number 50)

On the Transform Analysis of the original slide, **the file name data** doesn’t go through the controller. It is connected with **‘Generate CFG**′ process and **‘Print report name’** process directly. But, on the Structured Chart of the SD, controller calls the **‘Determine file name’** process. **File name data** from the process**(Determine file name process)** is also delivered to controller.
It generate discordance between Transform Analysis and Structured Chart.
So we modify the Transform Analysis. On the Transform Analysis of the modified slide, **File name data** go through the controller, and it is delivered to **‘Generate CFG’** process and **‘Print report name’** process.

On the Transform Analysis of the original slide, **Error data** and **Output Option data** are delivered to controller. But it is not expressed where to go. Some connections between the **data store** and **process** are connected directly. We modified these things according to adequate flow of the data.

***This slide is the same with slide number 18.**

Notation



Modules

Library Modules

Data module

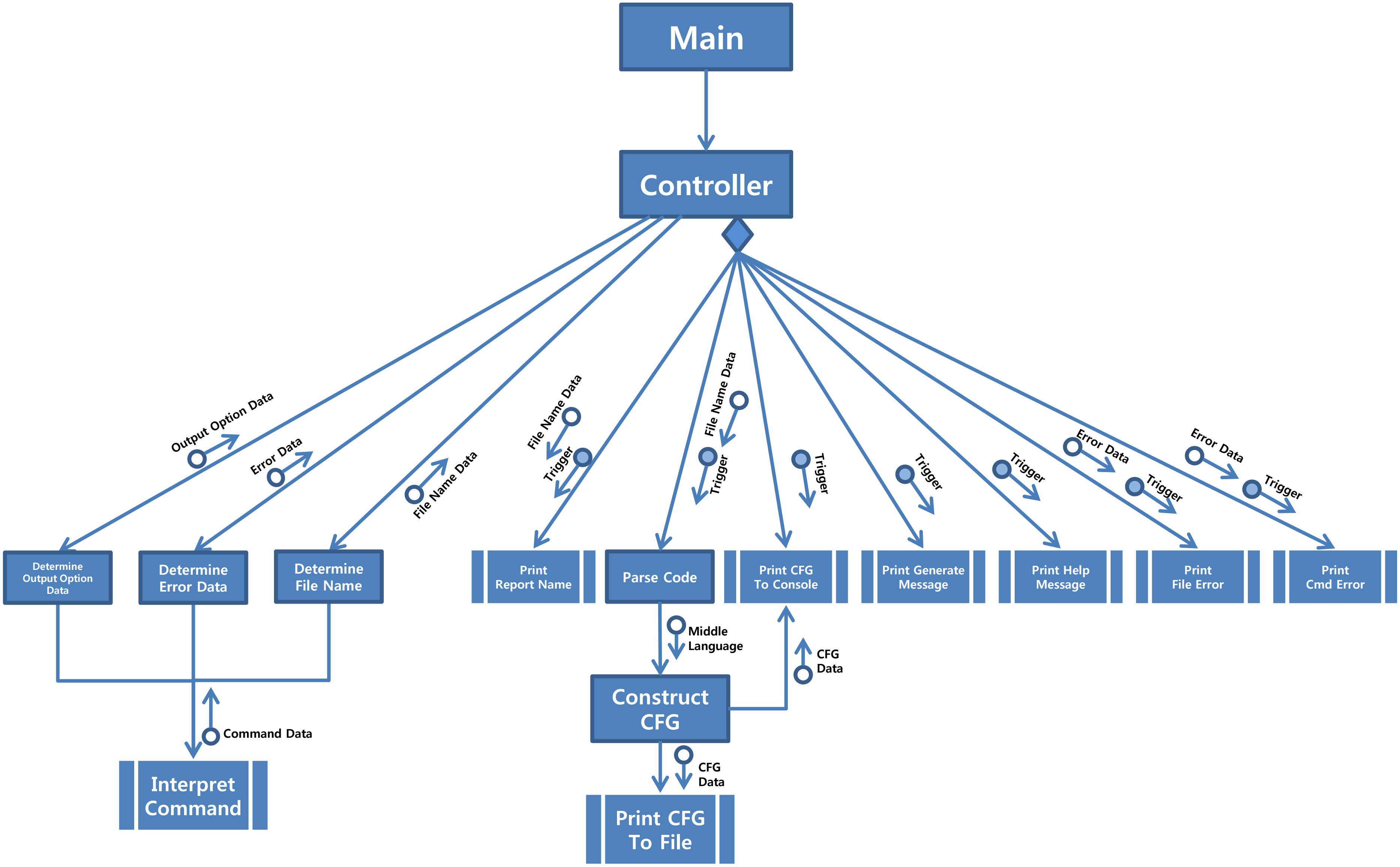
Module call

Data Flow

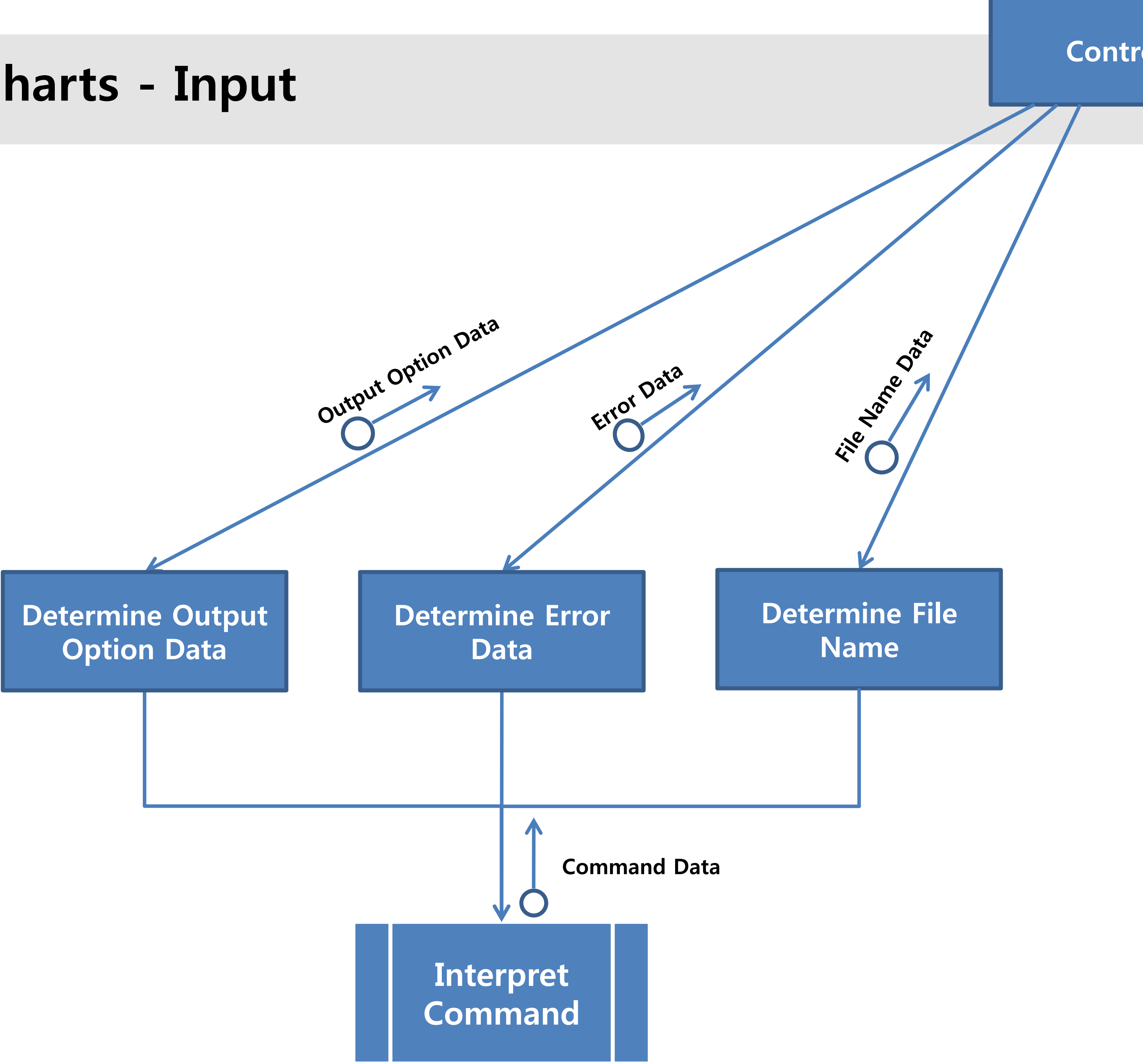
Control Flow

Decision

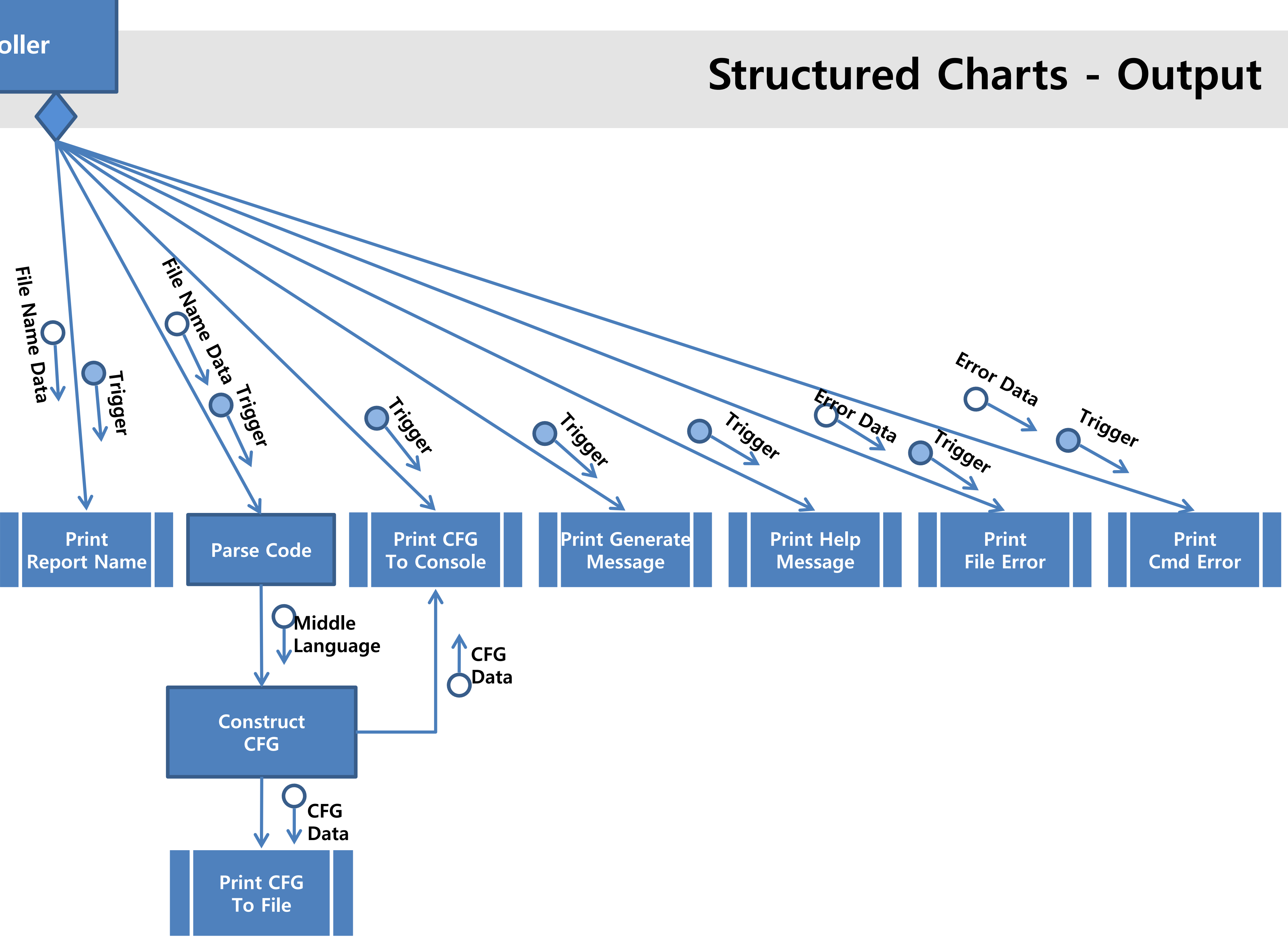
Structured Charts



Structured Charts - Input



Structured Charts - Output



Appendix

- Middle Language
- Parse Code
- Construct CFG

Middle Language

Middle Language

Middle Language is modified language from C-Code for convenience of CFG output.

Middle Language is used for following three reasons.

1. When convert C code to CFG, unify overlapping concepts into one thing.

ex) ‘for’ function and ‘while’ function has the same structure in CFG. Middle Language can express this two different function code into the same term.

2. Using this, it can express the CFG more structural.

ex) As specifying jump to where, it is better to realize the flow of the Code in CFG.

3. When programming, it is convenient to modularize and divide each task.

ex) In Parse Code process, make Middle Language from C code. In Construct CFG process, make CFG from Middle Language. This division of task reduces project delay.

Parse Code

Parse Code

Parser process makes Middle Language according to following ways.

1. Find the branch occurring spot in C code, add branch condition.
2. After numbering at each of branches, array these according to the numbering.
3. Add the matched [body] in Numbered branch.

Middle Language

Middle Language - Keyword

Middle Language is composed of following keywords.

n#COND $condition$ -> checkup condition, specify block that make branch.

n#BODY $code$ -> specify block that has no condition and branch.

#JUMP n -> specify relation of block and order of block.

** Between '$', code is lapped. It's for classifying Middle language.

** 'n' is for numbering each block.

** '#' is for distinguishing keyword from other things.

Middle Language

Middle Language – example of branch

```
 *** while(condition) { code }
3#COND $condition$
#JUMP $6$          // if not satisfy condition.
4#BODY $code$      // execute code.
#JUMP $3$          // checkup condition again.
6#BODY $code$      // out of while connection.
......
```

```
 *** do{ body } while ( condition )
11#BODY $code$
12#COND $condition$   // checkup condition.
#JUMP $11$            // if satisfy the condition, go to 11.
13#BODY $code$        // out of do while.
....
```

Middle Language

Middle Language – example of branch

```
 *** if(condition){code} else if(condition){code} else{code}
25#COND $condition$
26#BODY  $code$
#JUMP $30$
27#COND $condition$
28#BODY  $code$
#JUMP $30$
29#BODY  $code$
30#BODY  $code$
…
```

Middle Language

Middle Language – example of branch

```
switch(variable) case status1: code break; case status2:  code case status3: ....
25#COND $status1$      //if variable satisfy status.
26#BODY  $code$
#JUMP $31$             // escape switch because break.
27#COND $condition$
28#BODY  $code$        // execute next condition because non-break.
29#COND $condition$
30#BODY  $code$
31#BODY  $code$        // out of switch code .
...
```

```
for(init condition; condition ; increase and decrease){body}
43#COND$condition$
44#BODY  $code$
#JUMP $43$
45#BODY  $code$
...
```

Construct CFG

Construct CFG

Construct CFG analyze the Middle language. It prints as follows format.

```
1#
code....
2#
condition .....
    3#
    code..
4#
code......
5#
condition......
    6#
    condition....
        7#
        code....
    8#
    code....
9#
code.....
..
```

Thank You!