소프트웨어 모델링 분석

(SASD - SD단계)

1조

Concept : Smart Coffee Maker

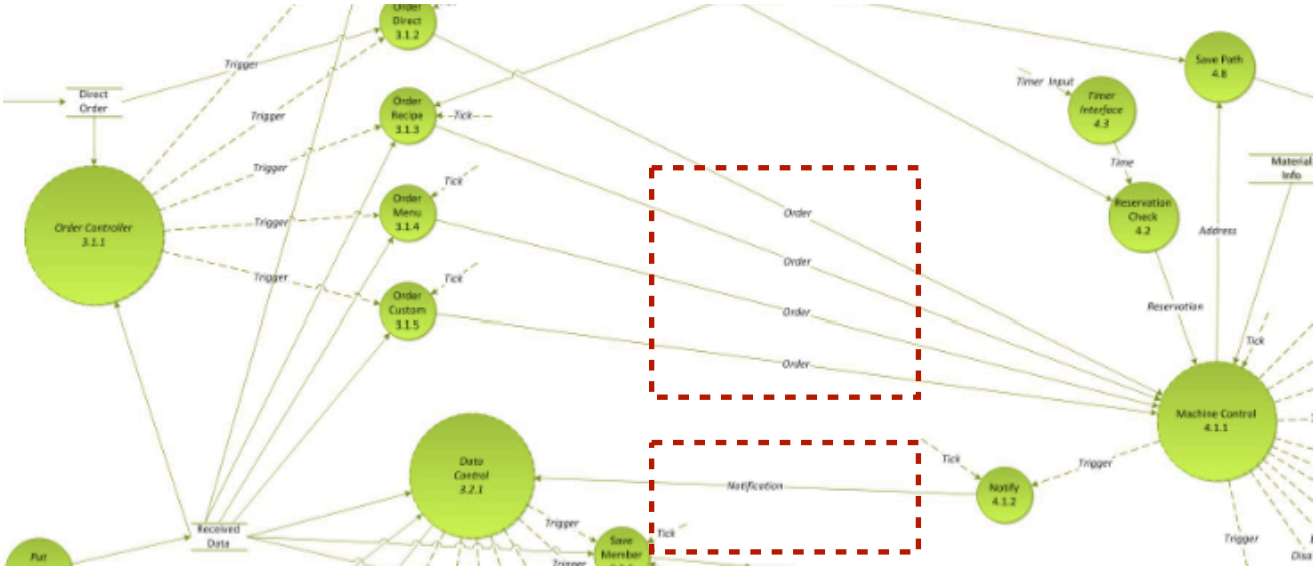
팀원 : 김정태 (200611460)

송준현 ()

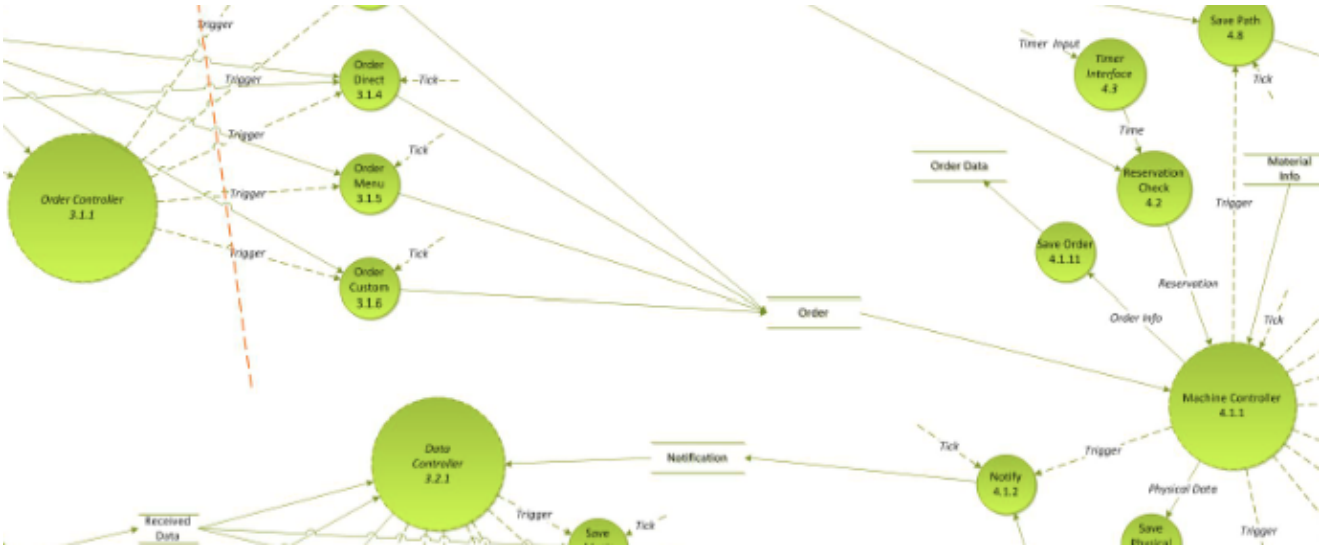
최정명 ()

이낙원 ()

SA 에서 변경 된 점



컨트롤이 컨트롤을 호출하는 구조가 되어서 데이터가 컨트롤 사이에서 순환이 발생하고 그리고 또 다른 문제는 컨트롤 사이에 동기화 문제가 발생한다.
이를 해결하기 위해 컨트롤이 호출하는 구조대신 버퍼를 사용해서 하나의 컨트롤은 계속 데이터를 버퍼에 저장하고 또 다른 컨트롤은 버퍼에 저장된 데이터를 소비하는 구조를 사용하여 컨트롤이 서로 영향을 주는 대신 버퍼를 통해서 컨트롤을 분리하였다.

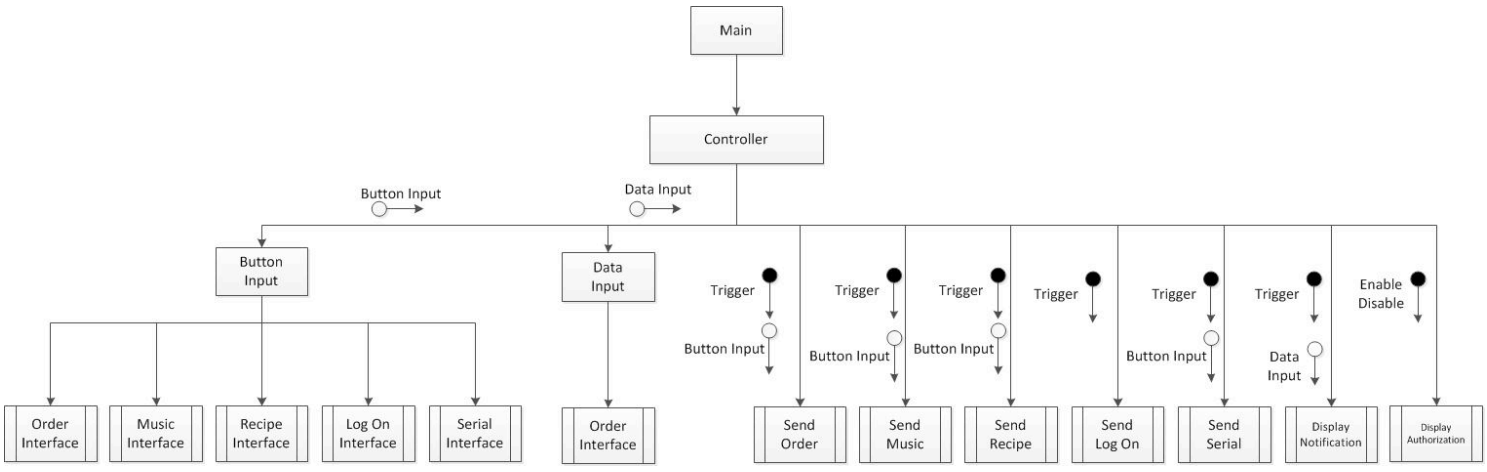


Structured Charts - Transform Analysis

- Remote Controller

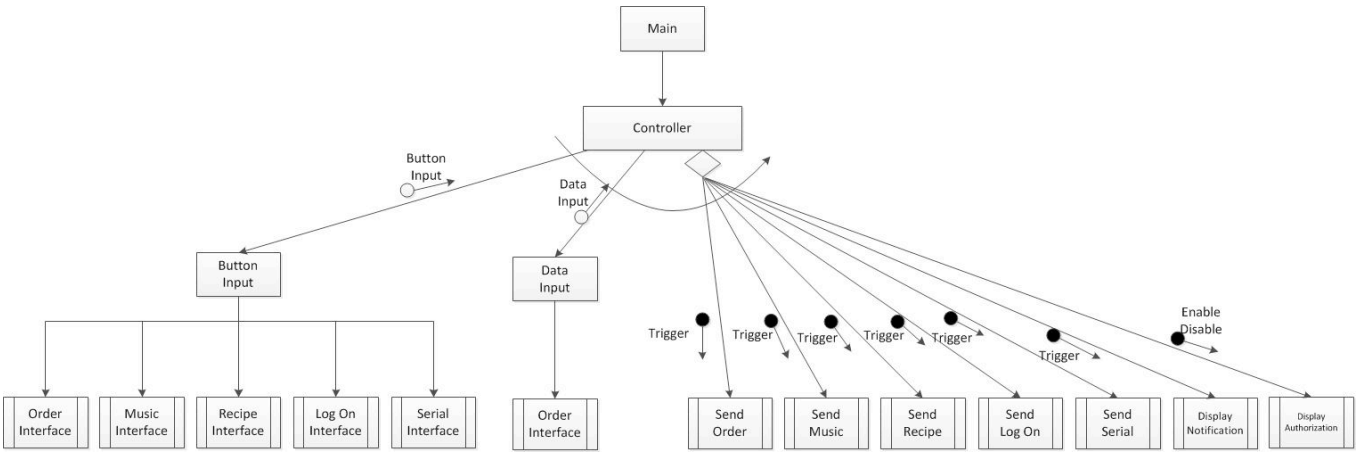
Structured Charts - Basic

- Remote Controller



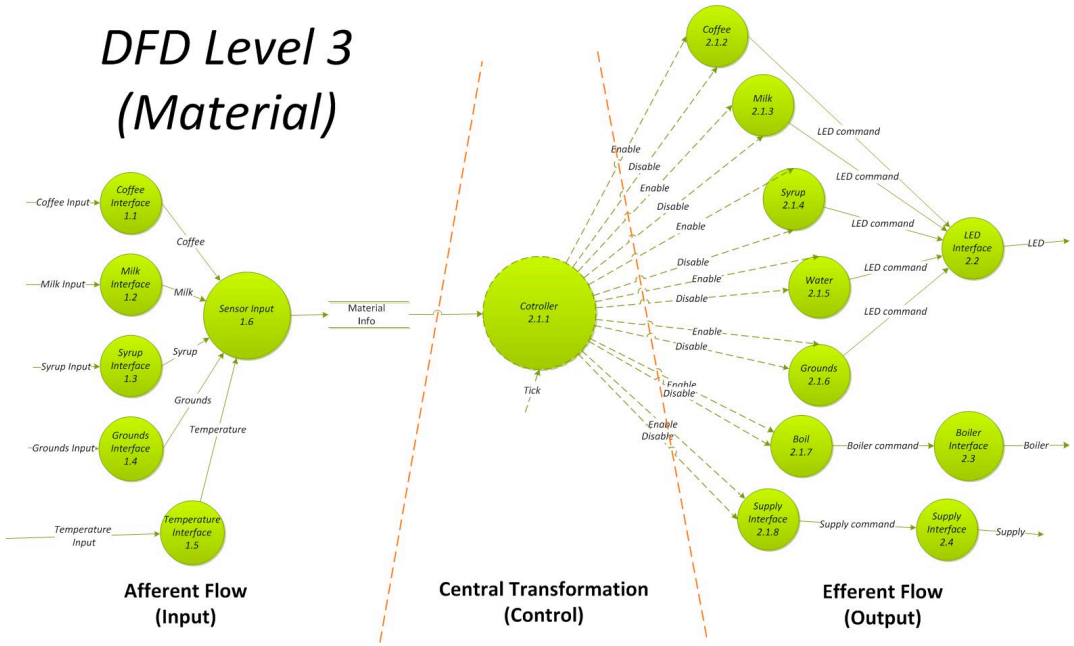
Structured Charts - Advance

- Remote Controller



Structured Charts - Transform Analysis

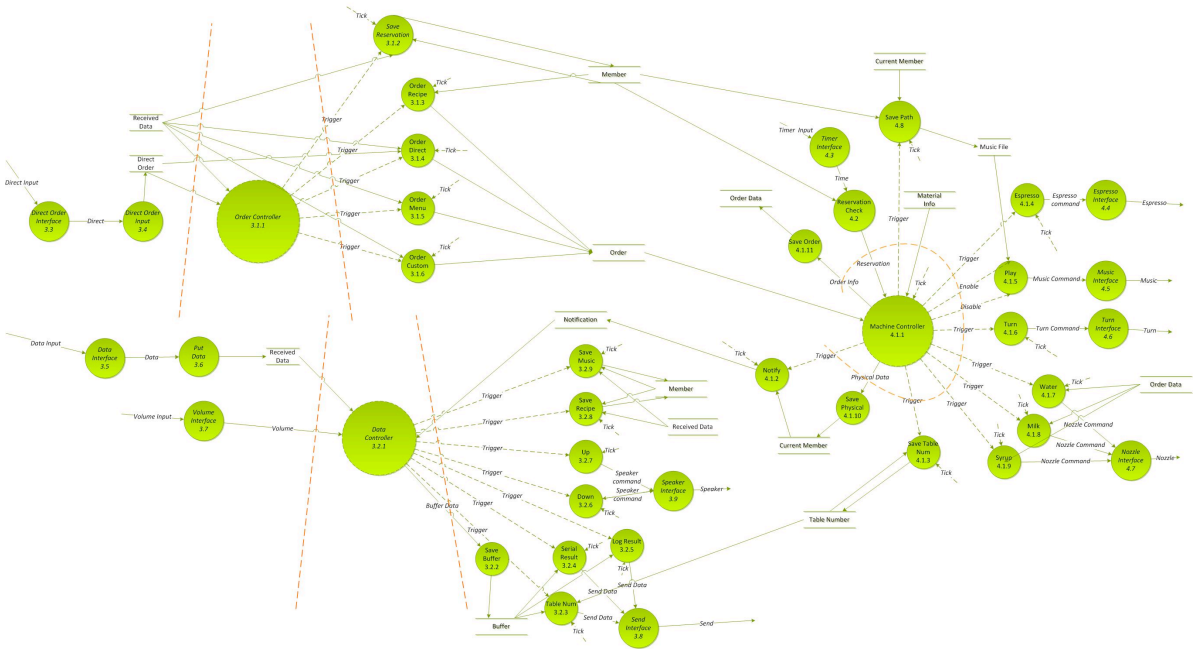
- Coffee Maker Part 1



Structured Charts - Transform Analysis

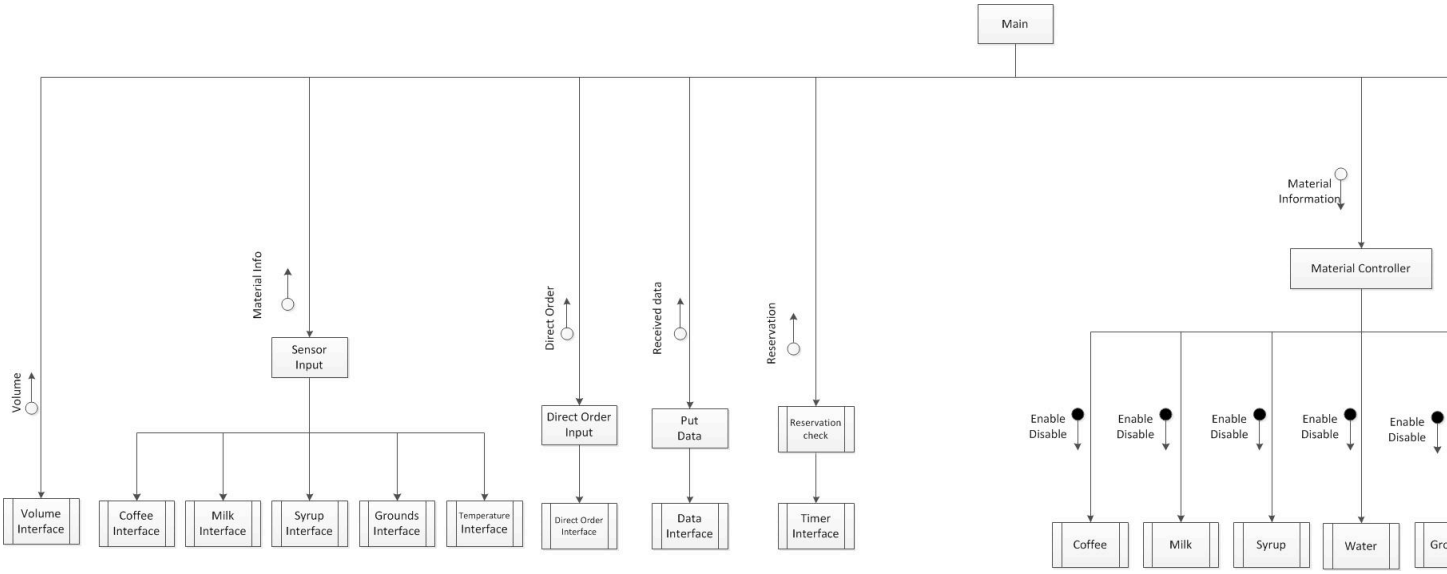
- Coffee Maker Part 2

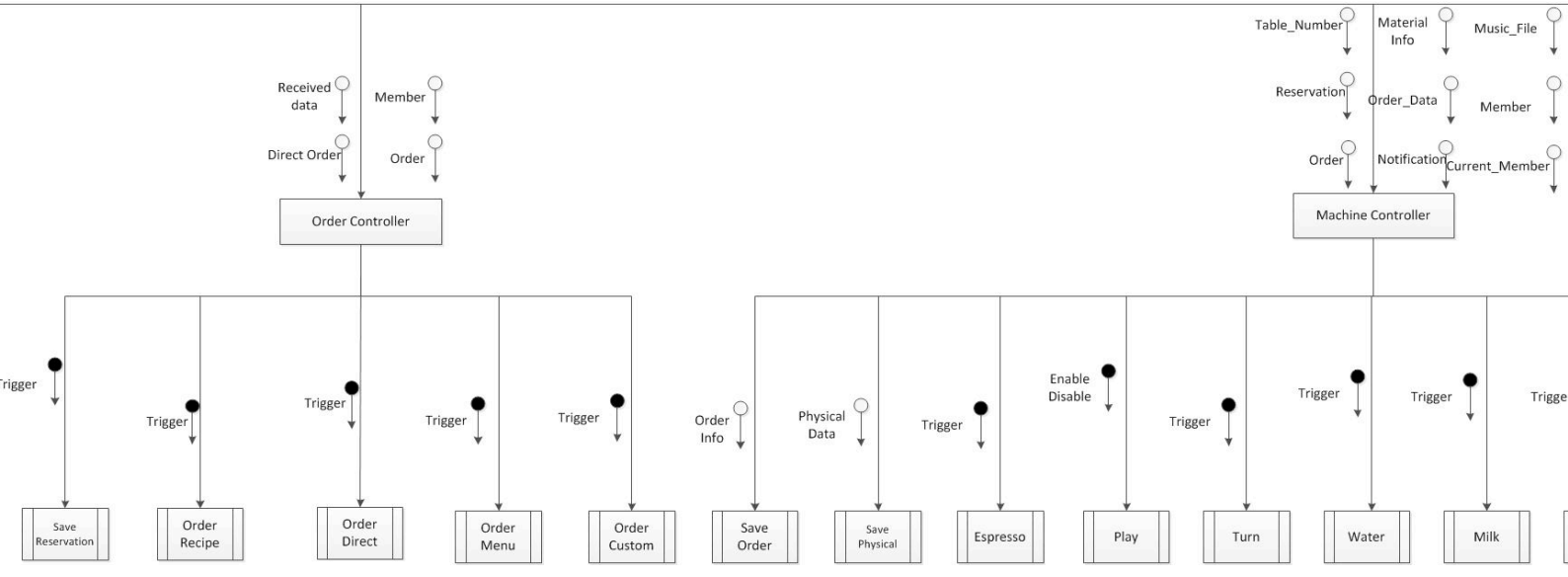
DFD Level 3(Machine)

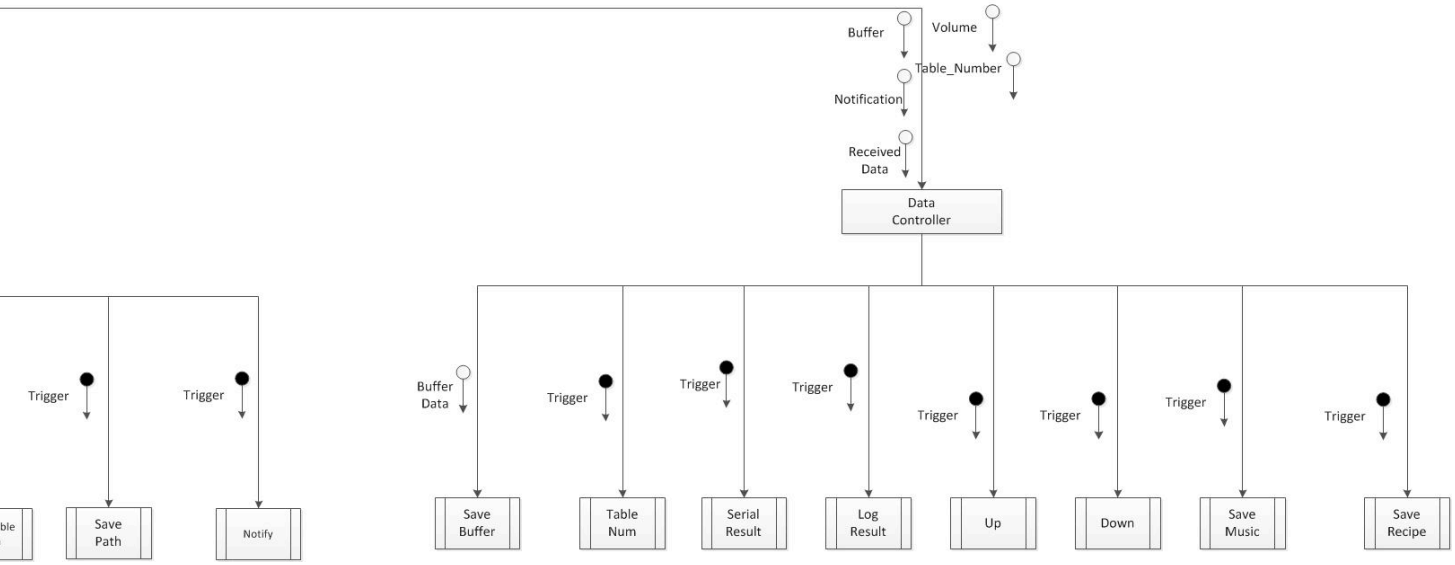


Structured Charts -Basic

- Coffee Maker

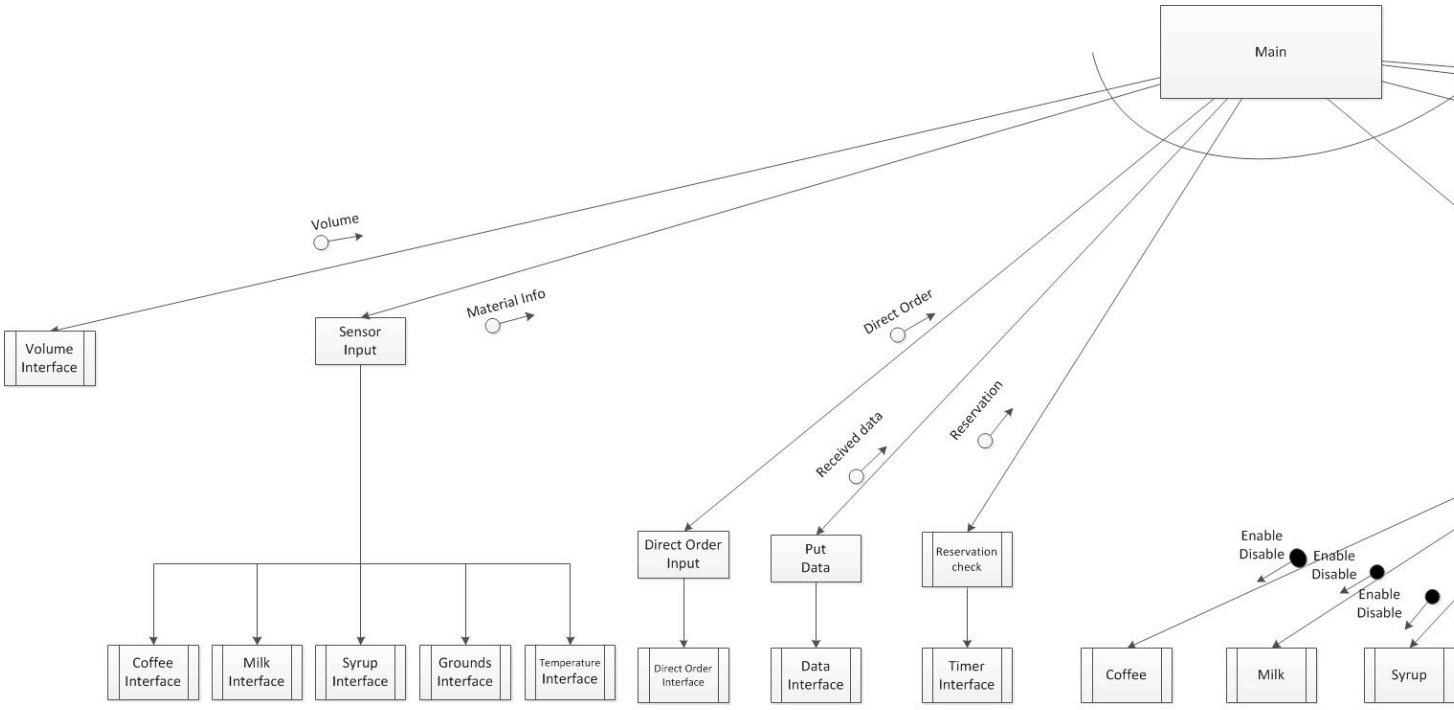


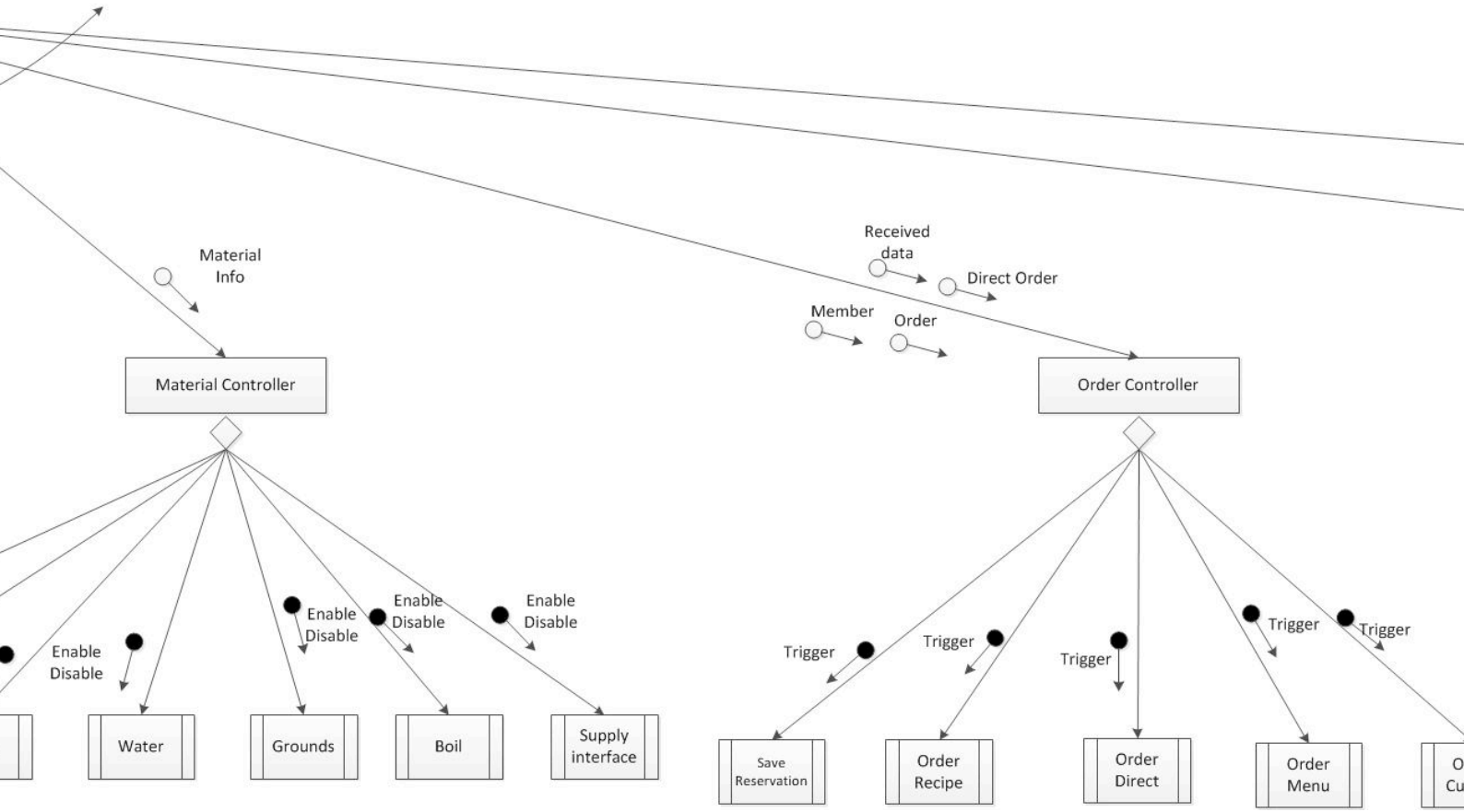


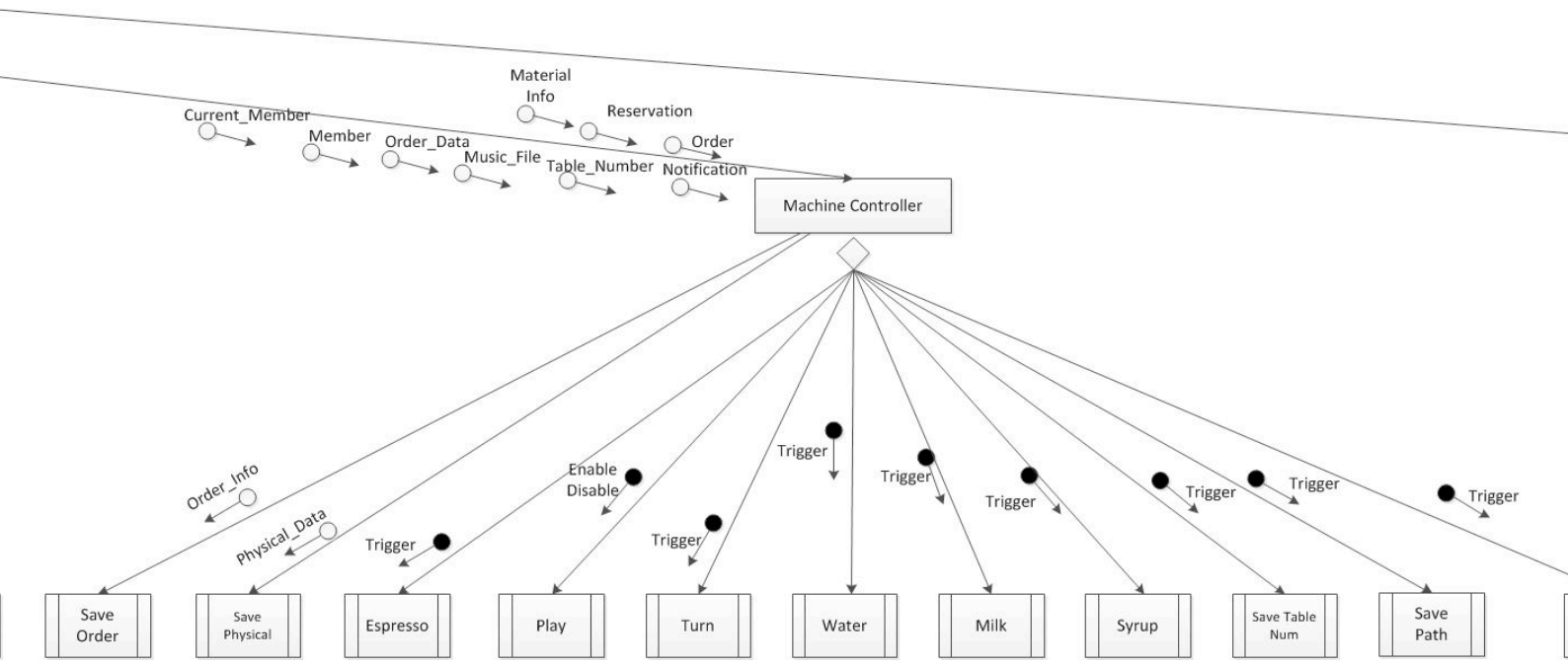


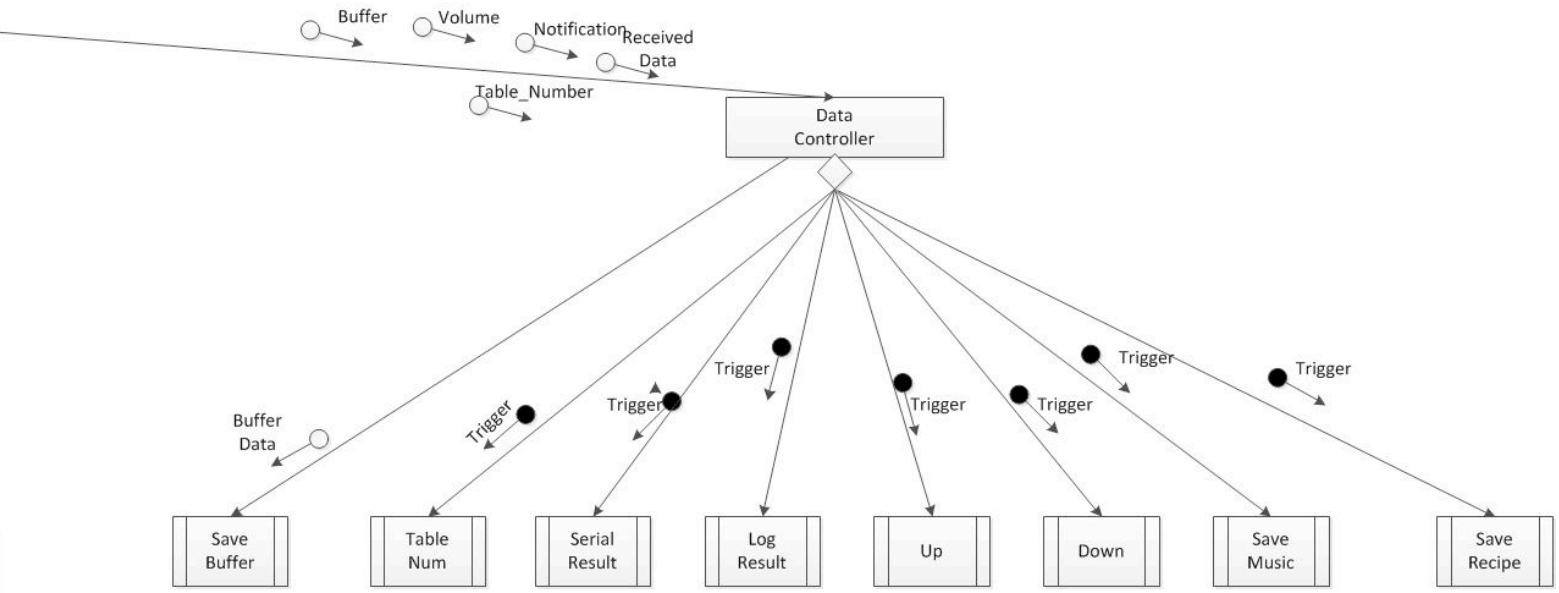
Structured Charts -Advance

- Coffee Maker




Material Info
Material Controller
Enable Disable
Enable Disable
Enable Disable
Enable Disable
Enable Disable
Water
Grounds
Boil
Supply interface
Received data
Direct Order
Member
Order
Order Controller
Trigger
Trigger
Trigger
Trigger
Trigger
Save Reservation
Order Recipe
Order Direct
Order Menu


Material Info
Reservation
Current_Member
Member
Order_Data
Music_File
Table_Number
Order
Notification
Machine Controller
Order_Info
Physical_Data
Trigger
Enable Disable
Trigger
Trigger
Trigger
Trigger
Trigger
Trigger
Trigger
Save Order
Save Physical
Espresso
Play
Turn
Water
Milk
Syrup
Save Table Num
Save Path



Pseudo code

- Remote Controller

```
void main()
{
    struct Button_Input, Data_Input;

    declare Remote's State;

    set Remote's State to initial state

    initialize structure data

    while(1)
    {

        Button_Input <- ButtonInput()
        Data_Input  <- DataInput();

        switch(Remotes State)
        {
            case Initial:   //Ω√¿€
                if((BF == 4) &&(SF == 0)
                {
                    Call Send_Log_On();
                    go to "Wait" state
                }
                break;
```

```
                case Wait:   //Wait
                     if((SF==1)&&(!LR))
                     {
                          call Display_Authorization(1);
                          go to "Try Authorization" state

                     }
                     else if((SF==1)&&(LR))
                     {
                          go to "Log On" state
                     }
                     break;

                case Try Authorization:   // Try Authorization
                     if(BF ==5)
                     {
                          Call Display_Authorization(0);
                          Call Send_Serial() with "Data Input"
                          go to "Ready Authorization" state
                     }
                     break;

                case Log On:
                     if((SF!=3)&&(BF==2))
                     {
                          Call Send_Music();
                     }
                     else if((SF!=3)&&(BF==1))
                     {
                          Call Send_Order();
                     }
                     else if((SF!=3)&&(BF==3))
                     {
                          Call Send Recipe();
                     }
                     else if(SF==3)
                     {
                          Call Display Notification();
                          Wait 5 second;
                     }
                     break;

                case Ready Authorization:
                     if((SF==2)&&(!SR))
                     {
                          exit(1);
                     }
                     else if((SF==2)&&(SR))
                     {
                          go to "Log On" state
                     }
                     break;
           }
     }
}
```

Pseudo code

- Coffee Maker

```
void main()
{

    struct Volume, Material_Info, Received_Data, Reservation, Order,
Member, Order_Data, Notification, Current_Member
         Table_Num, Buffer, Music_File, Direct_Order;

    declare contoller's states : Material_Controller, Order_Controller,
Machine_Contorller, Data_Controller;
    set 4 state's to initial state

    initialize structure data

    while(1)
    {

           //get input data from input interface.
           Volume <- Volume_Interface();
           Material_Info <- Sensor_Input();
           Direct_Order <- Direct_Order_Input();
           Received_Date <- Put_Data();
           Reservation <- Reservation_Check();

           //call controller with sequence order
           Call Material_Controller() with "Material_Info";

           Call Order_Controller() with "Recieved_Data" "Direct_Order"
"Member" and "Order";

           Call Machine_Controller() with "Member" "Current_Member"
"Material_Info" "Order_Data" "Order"
                                                "Notification"
"Table_Number" "Music_File" and "Reservation";

           Call Data_Controller() with "Received_Data" "Buffer" "Member"
"Volume" "Table_Number" and "Notification"

    }
}


//Materail_Controller
void Material_Controller() with "Material_Info"
{

    switch(Material_Controllers State)
    {
           case Initial:
                 if(C)
```

```
        {
                call Coffee(1);
        }
        else
        {
                call Coffee(0);
        }
        go to "Coffee_Checked" state;
        break;

case Coffeed_Checked:
        if(M)
        {
                call Milk(1);
        }
        else
        {
                call Milk(0);
        }
        go to "Milk_Checked";
        break;

case Milk_Checked:
        if(S)
        {
                call Syrup(1);
        }
        else
        {
                call syrup(0);
        }
        go to "Syrup_Checked" state;
        break;

case Syrup_Checked:
        if(G)
        {
                call Grounds(1);
        }
        else
        {
                call Gorunds(0);
        }
        go to "Grounds_Checked" state;
        break;

case Grounds_Checked:
        if(W==0)
        {
                call Supply(1);
                call Water(1);
        }
        else if(W==1)
        {
                call Water(0);
        }
```

```
                else if(W==2)
                {
                        call Supply(0);
                        call Water(0);
                }
                go to "Water_Checked" state;
                break;

        case Water_Checked:
                if(T==0)
                {
                        call Boil(1);
                        call Water(1);
                }
                else if(T==1)
                {
                }
                else if(T==2)
                {
                        call Boil(0);
                }
                go to "Temperature_Checked" state;
                break;

        case Temperature_Checked:
                go to "Initial" state;
                break;
    }
}

//Order_Controller
void Order_Controller() with "Recieved_Data" "Direct_Order" "Member" and
"Order"
{
    switch(Order_Controllers State)
    {
        case Ready_Order:
                if(F)
                {
                        go to "Remote_Order" state;
                }
                else if(!F&&DF)
                {
                        call Order_Direct();
                }
                break;

        case Remote_Order:
                if(Time!=2400)
                {
                        call Save Reservation();
                }
                else if(Time==2400 && What == 2)
                {
                        call Order_Menu();
                }
```

```
                else if(Time==2400 && What == 1)
                {
                      call Order_Custom();
                }
                else if(Time==2400 && What == 0)
                {
                      call Order_Recipe();
                }

                go to "Readay_Order" state;
                break;
    }

}

//Machine_Controller
void Machine_Controller() with "Member" "Current_Member" "Material_Info"
"Order_Data" "Order"
                                        "Notification" "Table_Number"
"Music_File" and "Reservation"
{
    struct Physical_Data, Order_Info;

    initialize data structure with input data

    Save_Physical() with "Physical_Data";
    Save_Order() with "Order_Info";

    switch(Machine_Controllers State)
    {
          case Idle:
                if(RF)
                {
                      call Save_Path() with "Music_File" and
"Current_Member";
                      go to "Music_Saved" state;
                }
                else if(!RF && OF)
                {
                      go to "Receive_Order" state;
                }
                break;

          case Music_Saved:
                call Play(1) with "Music_File";
                call Espresso();

                go to "Make_Espresso" state;
                break;

          case Make_Espresso:
                wait 120 seconds;

                call Water();
                call Milk();
                call Syrup();
```

```
                go to "Making_Done" state;
                break;

          case Making_Done:

                if(O_Physical == NULL)
                {
                      call Turn();
                      call Save_Table_Num() with "Table_Num";
                }
                else
                {
                      call Play(0);
                      call Turn();
                      call Save_Table_Num() with "Table_Num";
                      call Notify() with "Current_Member";
                }

                go to "Idle" state;
                break;

          case Receive_Order:
                if(O_Physical!=NULL)
                {
                      call Save_Path() with "Music_File" and
"Current_Member";
                      go to "Indirect_Order" state
                }
                else if(O_Physical == NULL)
                {
                      call Espresso();
                      go to "Make_Espresso" state;
                }
                break;

          case Indirect_Order:
                call Play(1) with "Music_File";
                call Espresso();
                go to "Make_Espresso" state;
                break;
     }
}

//Data Controller
void Data_Controller() with "Received_Data" "Buffer" "Member"
"Table_Number" "Volume" and "Notification"
{
     struct Buffer_Data;

     initialize data structure with input data;

     call Save_Buffer() with "Buffer_Data";

     switch(Data_Controllers State)
     {
```

```
    case Idle:
          if(Notify)
          {
               go to "Notify" state;
          }
          else if(!Notify && F==0 && VF)
          {
               go to "Volume" state;
          }
          else if(!Notify && F!=0 && F!=1)
          {
               go to "Not_Order" state;
          }
          break;

    case Notify:
          call Table_Num() with "Table_Num" and "Buffer";
          go to "Idle" state;
          break;

    case Volume:
          if(UP)
          {
               call Up();
          }
          else
          {
               call Down();
          }
          go to "Idle" state;
          break;

    case Not_Order:
          if(F==2)
          {
               call Save_Recipe() with "Member";
          }
          else if(F==3)
          {
               call Save_Music() with "Member";
          }
          else if(F==4)
          {
               call Log_Result() with "Buffer";
          }
          else if(F==5)
          {
               call Serial_Result() with "Buffer";
          }

          go to "Idle" state;
    }
}
```