CORFU FBDK

An Engineering Support System compliant with the forthcoming IEC 61499 standard

Quick Start Guide

Version 0.7.0 – October 2003

Editor: Chris Tranoris
Contributions: Kleanthis Thramboulidis

Electrical and Computer Engineering, University of Patras, Greece 26500 Patras

Table of contents

1 Introduction 3

2 Terms of use 3

3 Installing 3

4 CORFU FBDK file structure 3

5 Using CORFU FBDK 4

5.1 The FB type library 4

5.2 The Function Block type editor 4
5.2.1 Editing an existing Function Block type 4
5.2.2 Creating a new Function Block type. 5

5.3 Function Block diagram Editor 5
5.3.1 Editing an existing Function Block diagram. 5
5.3.2 Creating a new Function Block diagram. 5

5.4 System Layer Editor 6

5.5 Transforming UML to Function Block diagrams: The Transformation Facility Manager 6

6 Developing an application: The Feed and Carry case study 7

6.1 Developing an application using CORFU FBDK and the CORFU development process 7
6.1.1 The CORFU development process 7
6.1.2 Creating a new project 7
6.1.3 Define use Cases 8
6.1.4 Create Sequence Diagrams for Use Cases 9
6.1.5 The design phase 11
6.1.6 Designing the underlying system 13
6.1.7 Distributing the application 18

6.2 Designing an application without an object model 21
6.2.1 Designing the underlying system 21
6.2.2 Designing the application 25
6.2.3 Distributing the application 26

7 Frequently Asked Questions 27

7.1 What are Industrial Process Terminators (IPTs)? 27

7.2 What is an Industrial Process Parameter (IPP)? 27

7.3 What is a Process Interface FB Diagram? 27

8 Additional Information and Comments 27

1 Introduction

Thank you for downloading the CORFU Function Block Development toolKit (CORFU-FBDK). CORFU FBDK is an Engineering Support System compliant with the forthcoming IEC 61499 standard.

This quick start guide will help you to explore the basic functionality of the tool. For more information on CORFU framework [1], CORFU Engineering process [2] and CORFU architecture [3], please refer to related conference papers as well as to white papers available in http://seg.clab.ee.upatras.gr/corfu/.
An extensive presentation of CORFU FBDK can be found in [4].

It is recommended to read section 6, where an example on using the CORFU FBDK is given and it would be of great assistance on understanding the use of the CORFU FBDK and the CORFU development process.

2 Terms of use

Please have in mind that the current version is just a prototype and it has been developed in our effort to examine the IEC 61499 standard and our CORFU development process; it is not a commercial product and it should not be used for the development of real applications.
Your comments are welcome at the e-mail addresses provided at the end of this guide.

3 Installing

The current version of CORFU-FBDK runs only on Windows platforms 9x, Win2k and WinXP.
It has the following requirements: Celeron 1000, 64MB, 8MB on your Hard disk.
CORFU FBDK has an automated installation procedure.
To install the tool you should:
Download the corfu_fbdk_distribution.exe file and double click on it to install it.
Run the file tools/msxml.msi, to install msxml 4.0 parser on your system. CORFU-FBDK requires the MSXML 4.0 parser to work properly.
[Optional:In case you want to use the Microsoft Wizard for help tips.] Execute the files: tools/MSagent.exe, and tools/Spchapi.exe.
To start the CORFU-FBDK double click on the file corfuess.exe.

4 CORFU FBDK file structure

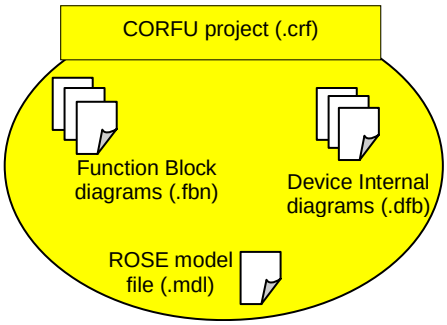


Figure 1

All files saved by CORFU FBDK are in XML format for readability and exchangeability. CORFU FBDK does not fully complies with IEC 61499 proposed file structure since it stores in the files additional information, except files with the .fbt extension which describe Function Block types where CORFU FBDK is fully compliant. CORFU project is saved in a file with .crf extension. A CORFU project file contains information about which Function block diagrams are used in the project, information regarding system layer objects and connections, and additional project information.
Function Block diagrams are saved separately in .fbn files. This enhances reusability since Function Block diagrams can be easily exchanged between projects. Process Interface FB Diagram are saved separately in .dfb files. These files keep information for the internal structure and connections of IEC devices. Process Interface FB Diagrams are presented and explained later on this guide. A UML model file from Rational ROSE (.mdl) is a file which keeps information for the Object model of your project. It is produced only if Rational ROSE is installed. Subdirectories after the “library” directory are consider as Function Block type libraries. Files that have the extension .fbt in these directories are considered as files describing Function Block types and they are imported automatically from CORFU FBDK while the application is loaded. If you want your library to be visible from CORFU FBDK you must copy it inside the “library” directory.
Figure 1 shows that a CORFU project is an aggregation of a UML model file from ROSE(.mdl), Function Block diagrams files (.fbn) and Process Interface FB Diagram files (.dfb). Thus when you save the project file all the other files are saved separately.

5 Using CORFU FBDK

The CORFU FBDK consists of the following components:

- a FB type library
- a FB type editor
- a FB diagram editor
- a System Layer (SL) editor
- a Transformation Facility Manager (TFM)

We next briefly describe the above components as well as the way they are used in the development process.
Follow the instructions below to discover the most basic functionality of CORFU-FBDK.

5.1 The FB type library

The FB type library is the repository of the CORFU FBDK for FB types. A number of predefined FB types is already contained in this library. A utility to import FB types defined by other vendors using the IEC61499 XML specification has been developed. This utility has already been used to import in our ESS all the FB types defined in Rockwell's FBDK. FB types are grouped in categoris/packages. FB types already contained in the CORFU FBDK are: E_RESTART, E_DELAY, FB_ADD_INT, PID_PRE, DERIVATIVE, TANK_MDL, etc.

5.2 The Function Block type editor

This editor is mainly used to:

- Edit existing FB types
- Create new FB types

5.2.1 Editing an existing Function Block type

To edit an existing Function Block type select from the menu FBType / Open. Then select from the CORFU-FBDK library (see fig.1) a FB to be loaded by the editor. You can select and open for example the PUMP_MDL.fbt from the folder library/process.
The Function Block Type Editor will open and load the selected FB type as shown in Figure 1.

You can edit the Function Block type or open another existing one available from the library. You can use the buttons Information to check:
-the available information and interface of the FB type
-XML Spec to view and edit the type directly by typing XML commands
-ECC editor to create an ECC diagram for the FB type,
-and Composition Editor (available on Composite FBs), to edit the internal of a Composite FB type

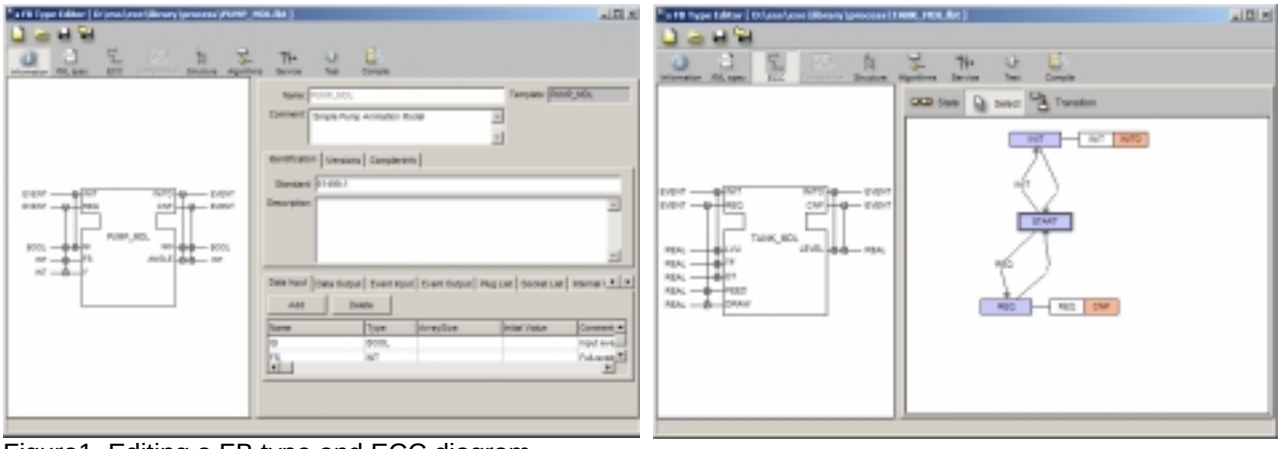


Figure1. Editing a FB type and ECC diagram

5.2.2 Creating a new Function Block type.

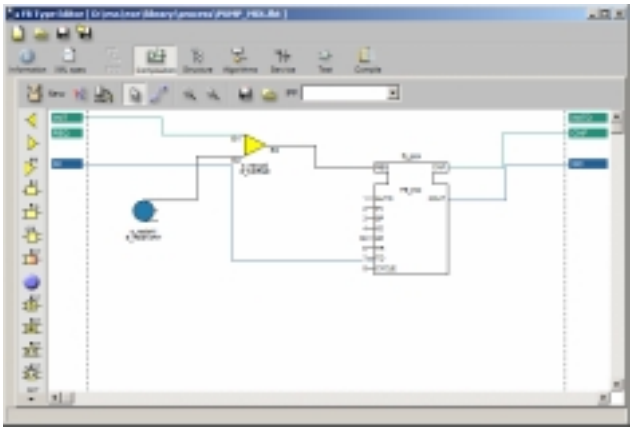


Figure 2. Editing a FB diagram of a Composite FB.

For the definition of a new FB type the following choices are supported: the default primitive FB template (New basic), the default composite FB template (New composite), or select any other FB from the FB type library to be used as template (New, based on...).
To define the structure of a composite FB you have to activate the Composition window (figure 2) by clicking the button Composition. On this window you can define the FB diagram or open an existing one, and then export Events and Data to the interface of your Composite FB, by drawing connections.

5.3 Function Block diagram Editor

5.3.1 Editing an existing Function Block diagram.

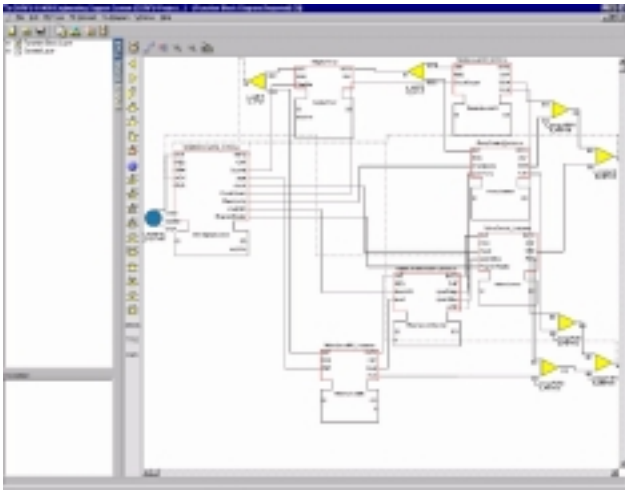


Figure 3. The Function Block Diagram editor

In order to edit a Function Block diagram you can start from the menu FBDiagram. Click on the menu FBDiagram/Open and then select the file Net1.fbn . On the appearing window (Figure 3) you can press on the button New in order to insert a new Function Block instance. The toolbar on the left displays the event function blocks in our stereotyped form. You can switch this by pressing the button Toggle Event Stereotypes. After inserting Function Block instances, you can interconnect them by pressing the button Make Connection. The FB diagram editor switches to Connection mode. You can click with the mouse-cross to an Event/Data output to an Event/Data input. Right clicking on the objects, displays useful information for the properties of the FB instance.

5.3.2 Creating a new Function Block diagram.

In order to create a new Function Block diagram you can start from the menu FBDiagram and click on the menu FBDiagram/New.

5.4 System Layer Editor

In CORFU FBDK you can design a first view of the underlying process. By clicking on the menu SystemLayerDiagram / New the System Layer Editor window appears. On this window you can drop IEC compliant devices, fieldbus networks, switches, etc and interconnect them. You can also make a preliminary distribution of your application (see figure 4), by dragging a Function Block Instance from a Function Block diagram and drop it on an IEC compliant device. Right click on any object to edit its properties.

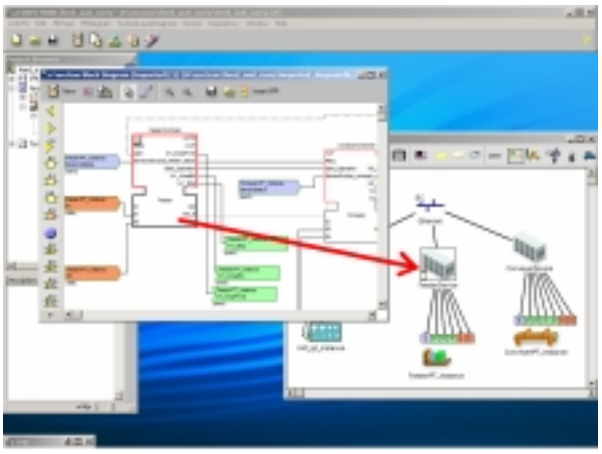
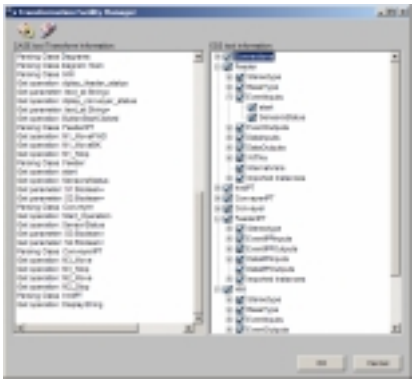


Figure 4. Dragging and dropping a FB instance to an IEC device

5.5 Transforming UML to Function Block diagrams: The Transformation Facility Manager



To automate the transformation process of UML diagrams to FB network diagrams, we have designed and implemented in the CORFU FBDK the Transformation Facility Manager (TFM) shown in figure 5. CORFU FBDK supports the CORFU Development process. You must have installed Rational Rose 2000 or above on your system in order to use the Transformation Facility Manager tool.

Figure 5 The Transformation Facility Manager

TFM is a core utility of our tool since it incorporates and applies the transformation rules, informs and guides the engineer during the transformation process. TFM implements all the interface of the CORFU FBDK with Rose. It is responsible for the creation of the proper new types, events, etc from the analysis model in Rose. The most important task of the TFM is the creation of new FB types in the CORFU FBDK, by properly parsing and transforming the class and interaction diagrams from Rose.

6 Developing an application: The Feed and Carry case study

The following paragraphs will present a step-by-step CORFU FBDK usage and will guide you on designing your own projects. The project that we will use in our example is the *Feed and Carry object* from Yamatake co. The presentation and description of the project, which it should be read before proceeding with this guide, can be downloaded from this address www.holobloc.com by selecting the feed_carry_test.pdf. The system consists of a Human Machine Interface (HMI) running on a PC computer, a Feeder machine which is controlled from a device, and two conveyers that are controlled from another device. An object must be moved from the feeder to the conveyer and carried until the end of the conveyer.
The HMI sends a command to the system to start the Feeder machine and the system sends the Feeder status. The Feeder starts moving the object forward, towards the Conveyer, while sending continuously the status of sensors S1 and S2 back to the system .When the Feeder is in front of sensor S1 the system sends a message to the Feeder to start moving backwards while it sends a message to the Conveyer to start operating M3 and reports the status to the HMI. The Conveyer starts moving while continuously sends the status of sensor S4. When the Feeder is in front of S2 the system sends a message to the Feeder to stop moving. When the object is in front of the sensor S4 the system sends a message to the Conveyer to move M2 and stop M3. When the object is in front of the sensor S3 the system sends a message to the Conveyer to stop M2. The system always report the status of the Conveyer to the HMI.

6.1 Developing an application using CORFU FBDK and the CORFU development process

6.1.1 The CORFU development process

In this paragraph we will describe on a step by step procedure, how we can design the *feed and carry* application with the CORFU development process. Figure 6 shows the workflows that we will follow in order to design the application. The analysis phase will be done in Rational ROSE [4] a Computer Aided Software Engineering (CASE) tool for modeling software applications and which fully supports the UML [5]. The design will be done with the CORFU FBDK tool. The CORFU FBDK communicates with Rational ROSE through COM automation. If you don't have installed Rational ROSE, then you can read the paragraph 8 where we describe the design of the feed and carry application without using the CASE tool, although it is recommended to read this paragraph since it fully presents a realization of the CORFU development process.

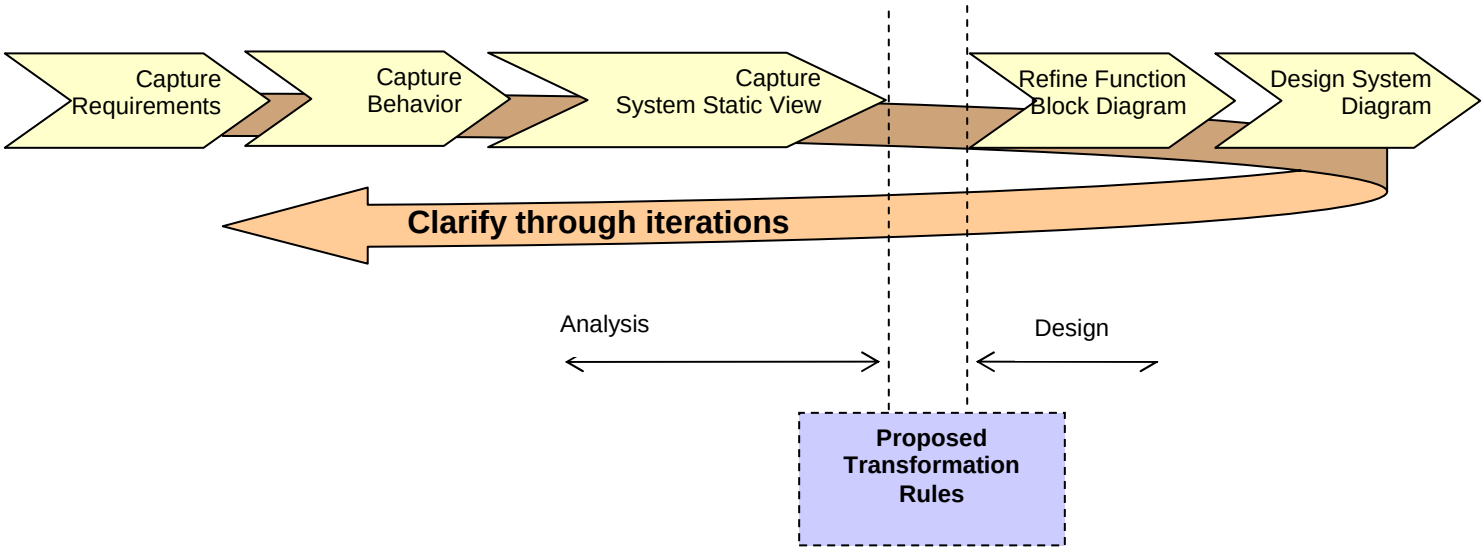


Figure 6. The CORFU development process

6.1.2 Creating a new project

Start CORFU FBDK and from the CORFU Start center select *New Project* or from the menu *CORFU/New CORFU project.* Select also where the filename **defaultstereotypes.ini** is located (usually in Rose directory). In the opened window type for the Project title: *feed_and_carry* and browse for a path to save the project file feed_and_carry.crf, for example D:\ess\feed_and_carry\feed_and_carry.crf, check the

Connect to ROSE checkbox and press the OK button. After a few seconds the files feed_and_carry.crf and feed_and_carry.mdl will be created in your folder. The window *Project Browser* on the left displays useful information. (figure 7)

6.1.3 Define use Cases

Open the IPMCS Requirements treeitem and you will see Use Case Diagrams, Actors, Use Cases and Interaction Diagrams. Right click on the tree and select from the menu *Add/Use Case.* In the window type for the new use case name: Feed and Carry and press OK. Rational ROSE will be displayed with the use case Feed and Carry created. Right click on the Use Case and select Open specification and write the following in the Documentation:

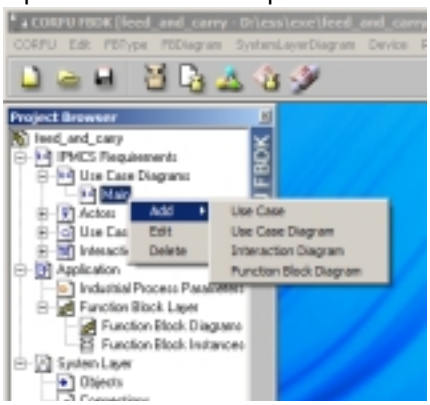


Figure 7. Project Tree View browser

USE CASE: Feed and Carry object

The user through HMI sends a command to the system to start the Feeder machine and the system sends the Feeder status. The Feeder starts moving the object forward, towards the Conveyer, while sending continuously the status of sensors S1 and S2 back to the system . When the Feeder is in front of sensor S1 the system sends a message to the Feeder to start moving backwards while it sends a message to the Conveyer to start operating M3 and reports the status to the HMI. The Conveyer starts moving while continuously sends the status of sensor S4. When the Feeder is in front of S2 the system sends a message to the Feeder to stop moving. When the object is in front of the sensor S4 the system sends a message to the Conveyer to move M2 and stop M3. When the object is in front of the sensor S3 the system sends a message to the Conveyer to stop M2. The system always report the status of the Conveyer to the HMI.

Close the Window by pressing OK. You also add an actor named: user. Figure 8 shows how Rational ROSE appears.

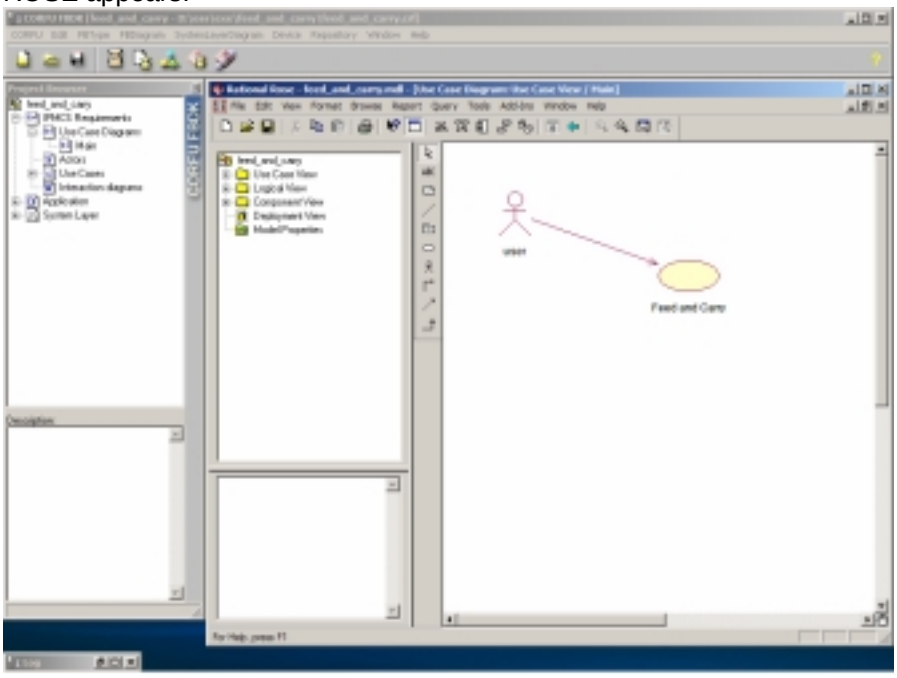


Figure 8. Add a Use Case.

You should go to ROSE and press the *Save button* in Rose in order to save occasionally your model.

6.1.4 Create Sequence Diagrams for Use Cases

Go again on the CORFU window *Project Browser* on the left and right click on the tree and select from the menu *Add/Interaction diagram*, type in the window *Feed and Carry Object* and press OK. In ROSE the interaction diagram will appear on an empty window. Before designing the interaction diagram we will go and design the class diagram of the application. Although the design of the class and interaction diagram usually is done from the software engineer in parallel, in our example for clarity we will first design the class diagram and then the interaction diagram.

Go to ROSE and on the tree, to the *Logical View* double click on item *Main*. The Class Diagram: Logical View/Main will appear. Add in ROSE a class named FeederIPT. Right click on the class Open Specification and select as a Stereotype IndustrialProcessTerminator. You must also add the following operations: M1_MoveFWD(), M1_MoveBK() and M1_Stop(). Press the OK button. You can read more for the IndustrialProcessTerminator stereotype in the section 7: Industrial Process Terminator and Industrial Process Parameters

Add a class named Feeder. Right click on the class Open Specification and select as a Stereotype FunctionBlock. Also Add the following operations: Start() and SensorsStatus(S1,S2:Boolean). The arguments S1, S2 you can add them on the tab Detail when you doubleclick on the SensorsStatus operation.

Add a class named ConveyerIPT. Right click on the class Open Specification and select as a Stereotype IndustrialProcessTerminator. You must also add the following operations: M2_Move(), M2_Stop(), M3_Move() and M3_Stop(). Press the OK button.

Add a class named Conveyer. Right click on the class Open Specification and select as a Stereotype FunctionBlock. Also Add the following operations: Start_Operation() and SensorStatus (S3,S4:Boolean). The arguments S3, S4 you can add them on the tab Detail when you doubleclick on the SensorsStatus operation.

Add a class named HMIPT. Right click on the class Open Specification and select as a Stereotype IndustrialProcessTerminator. You must also add the following operation: DisplayString(text_st : String) . Press the OK button.

Add a class named HMI. Right click on the class Open Specification and select as a Stereotype FunctionBlock. Also Add the following operations: diplay_feeder_status(text_st : String), diplay_conveyer_status(text_st : String) and ButtonStartClicked(). The argument text_st you can add it on the tab Detail when you doubleclick on the diplay_feeder_status operation.

The class diagram in Rose should be as on figure 9. The icons represent stereotyped classes. The top 3 classes are IndustrialProcessTerminators and the bottom 3 FunctionBlocks.

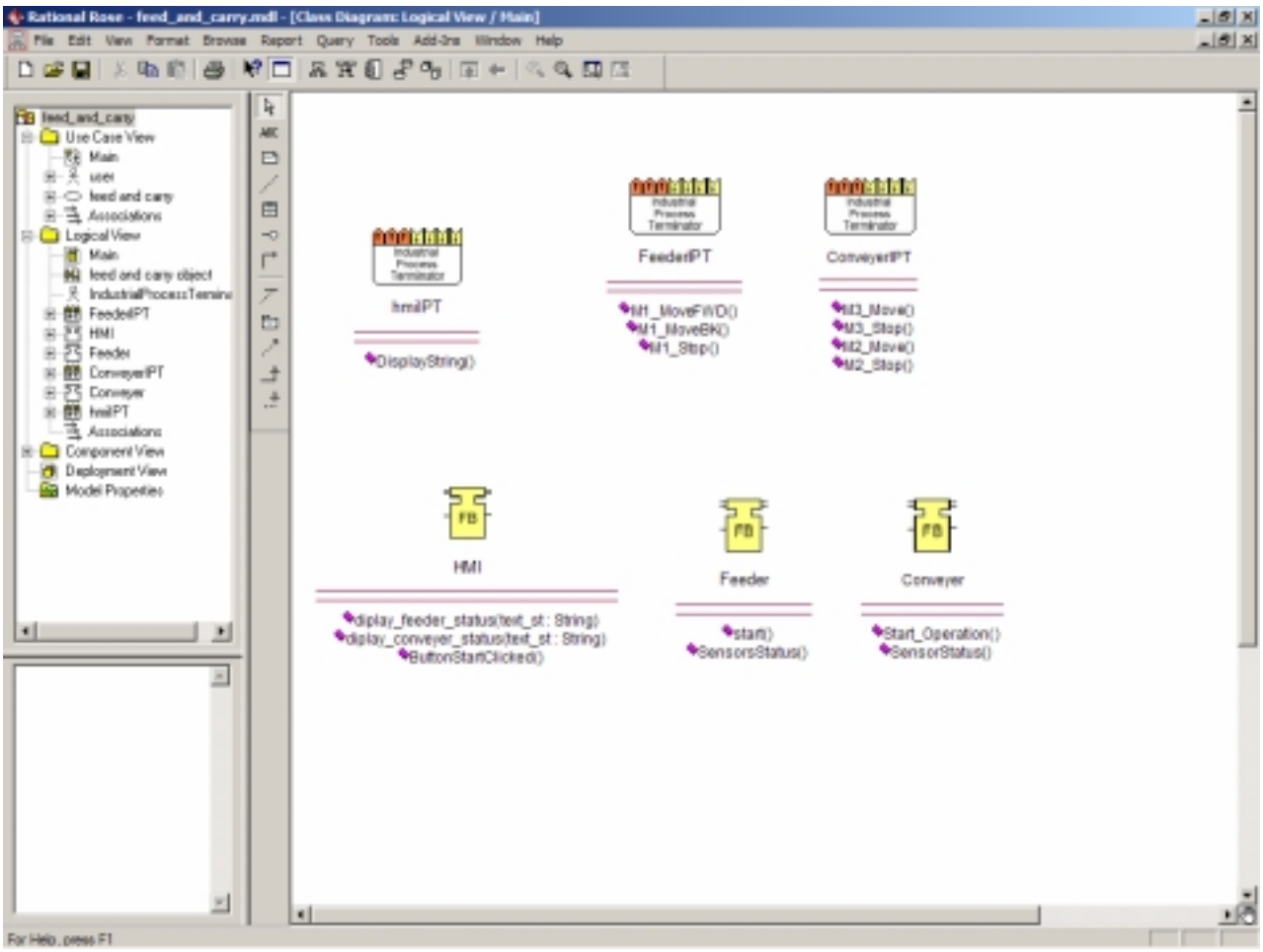


Figure 9. Class diagram with classes as stereotypes in Rose

On ROSE and on the tree, to the *Logical View* double click on interaction diagram item *feed and carry object.* We will design the interaction that implements the Use Case Feed and Carry. Create on Rose the diagram shown on Figure 10.

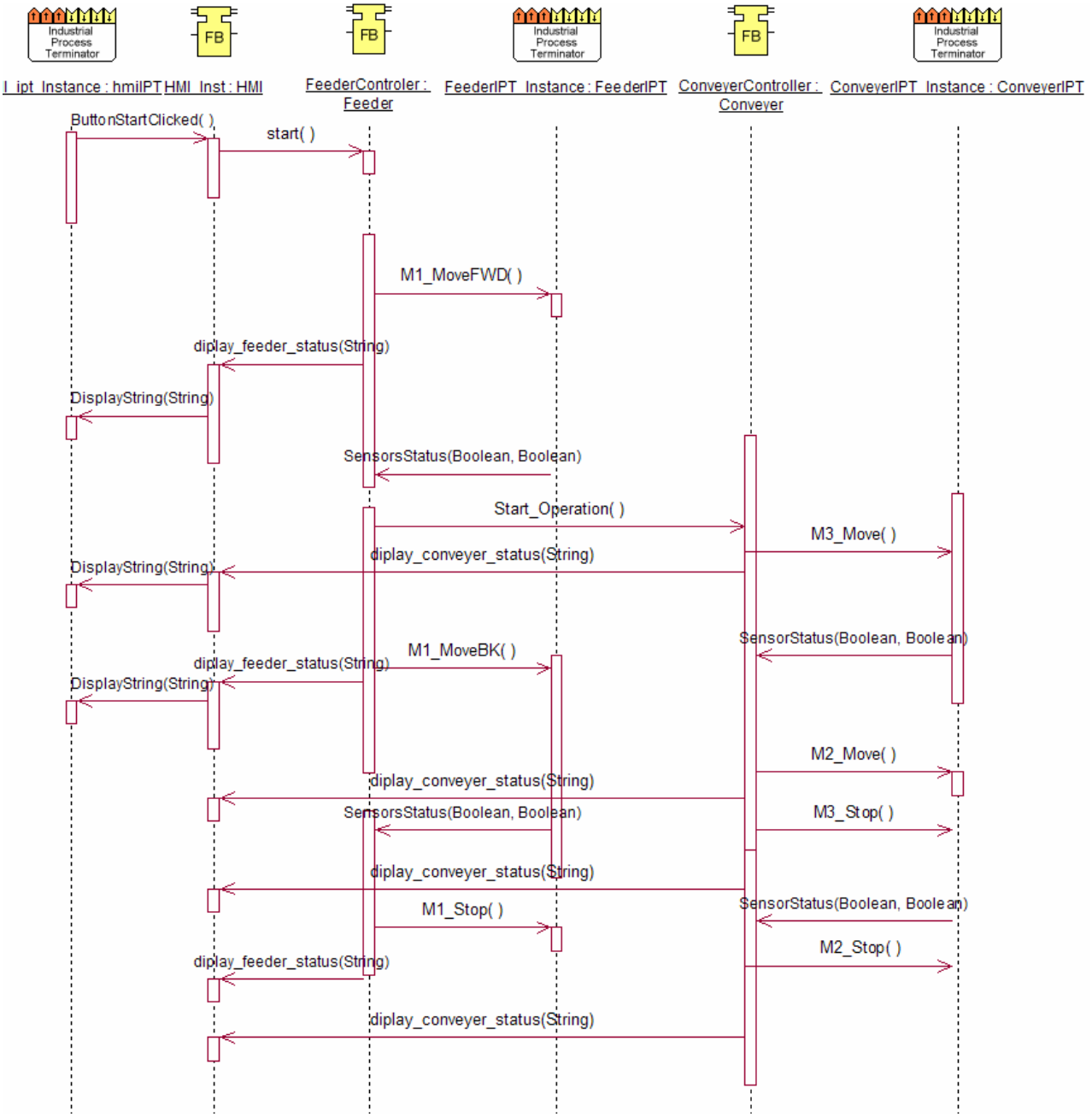


Figure 10. Interaction Diagram in Rose

After designing the interaction diagram you can hide Rose, by clicking in CORFU FBDK the 8th button on the toolbar *Show/Hide Rose*.

6.1.5 The design phase

We will proceed now to the next step by generating automatically Function Block diagrams from the class and interaction diagrams. Click in CORFU FBDK the 7th button on the toolbar *Transformation Facility Manager.* [Notice: Our transformation process does not currently support the transformation of Statechart diagrams. Future research and implementation will be done on this area, since it seems possible information from statechart diagrams to be included in ECC diagrams of Function Block types.] When the window appears, press the first button *Refresh Information.* The tool will start communicating with Rose by parsing the object model and will inform you for the process. When it will finish on the right part of the window you will have the objects that the tool automatically identified. You should see something similar as on figure 11.

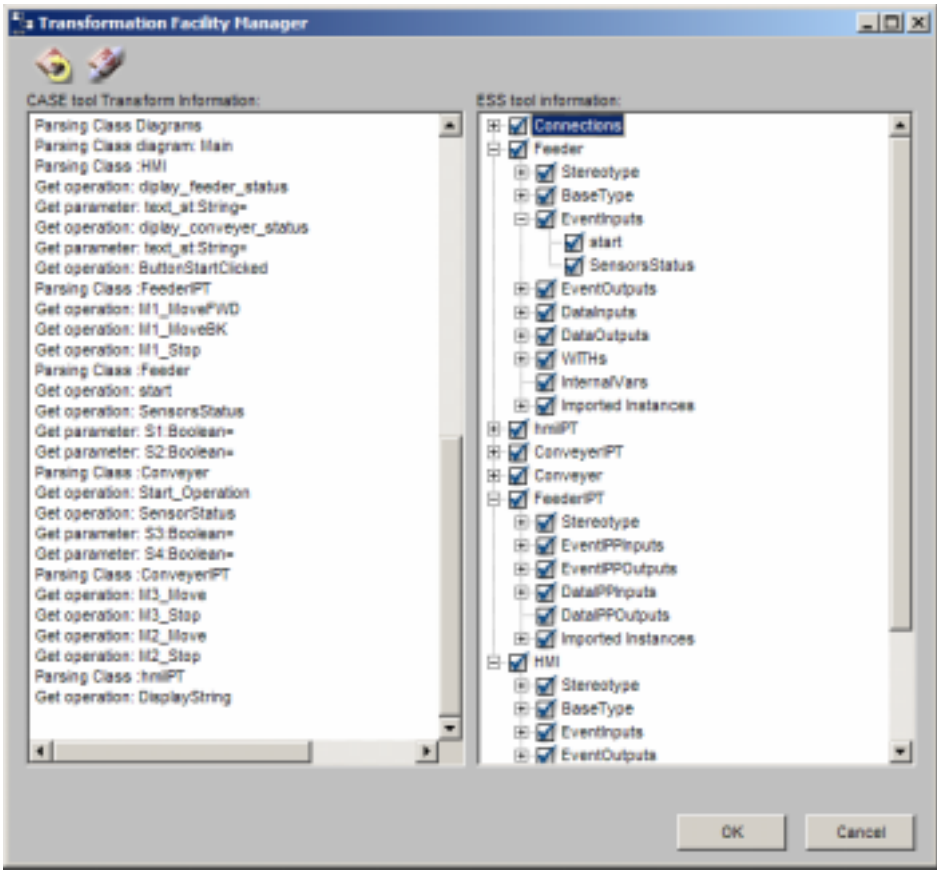


Figure 11. The Transformation Facility Manager window, after parsing information from Rose

Continue by pressing the OK button. Then the tool will produce automatically the Function Block diagram shown on figure 12.

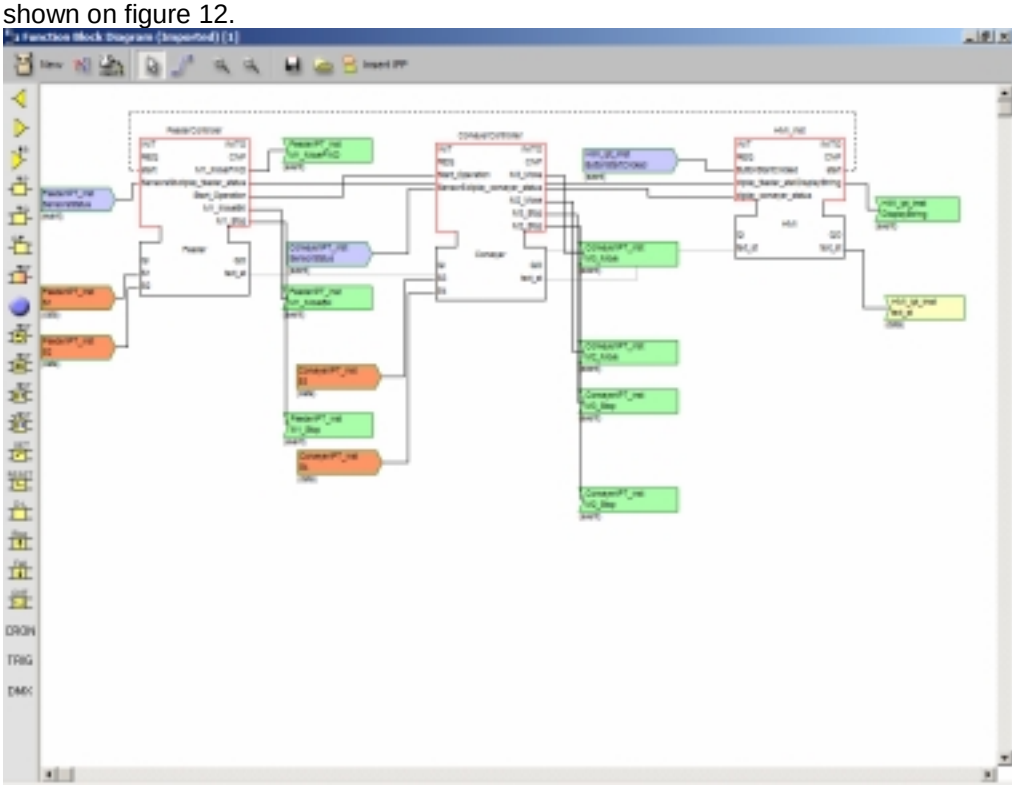


Figure 12. The automatically generated Function Block diagram

The automatically generated diagram on figure 7 is almost identical to the application Function Block diagram from Feed_and_carry document on page 42, presenting the effectiveness of the transformation process. The transformation process created 3 Function Block instances: FeederControler, ConveyerControler and HMI_Inst. Additionally, the transformation process created 3 new Function Block types, located in library “Imported”: HMI, Feeder and Conveyer. The exception here are the small blue, orange, green and yellow arrows. We call these items Industrial Process Parameters (IPP), which are parameters of the underlying process. For example we automatically identified that we need an **input event IPP (blue)** *SensorStatus* that will go as input to the Function Block Feeder WITH **input data IPP** S1 and S2 (orange). All three of them should come from the FeederIPT_Instance or in other words the actual industrial device that is connected to the Feeder machine. You can find more on IPTs and IPPs on section 7.
Now press the from the menu CORFU/ Save Project to save the project. It will appear a window to save the imported Function Block diagram. Save it as imported_diagram.fbn .
The following paragraphs will explain how we design and make available to the application the Industrial Process Parameters in CORFU FBDK.

6.1.6 Designing the underlying system

We must now design the underlying system and later on distribute the Function Block instances in devices.
Select from the menu *SystemLayerDiagram/New* and the System Layer Editor window will open. You will see that 3 IPTs are automatically inserted from the transformation process: HMI_ipt_instance, ConveyerIPT_Instance and FeederIPT_Instance. Select the ConveyerIPT_Instance, make a right mouse click and select from the menu *Properties of ConveyerIPT_Instance*. On the window change the type in the combobox from Undefined to Conveyer. On the same window you can also check the Industrial Process Parameters that the Conveyer has. Click in the OK button and ConveyerIPT_Instance will change its appearance. Repeat for HMI_ipt_instance and select as type Control Panel and for FeederIPT_Instance select type Feeder. You will have something like the images in figure 13.

Figure 13. The IPT instances on System layer.

These 3 objects are the components of the underlying process. Press now from the System Layer Editor toolbar the 4th button *IEC compliant device.* An icon that represents an IEC device will be inserted. Right click on it and select Properties of IEC_Device0. Name it FeederDevice and close the window. Repeat and add a device named PC and another names ConveyerDevice. Press now from the System Layer Editor toolbar the 3rd button *IEC compliant fieldbus*, right click on it and name it Ethernet. Now click on the toolbar the 2nd button and switch to connection mode. Connect the FeederDevice to the first IPP SensorStatus of FeederIPT_Instance. Repeat to produce a diagram similar to Figure 14.

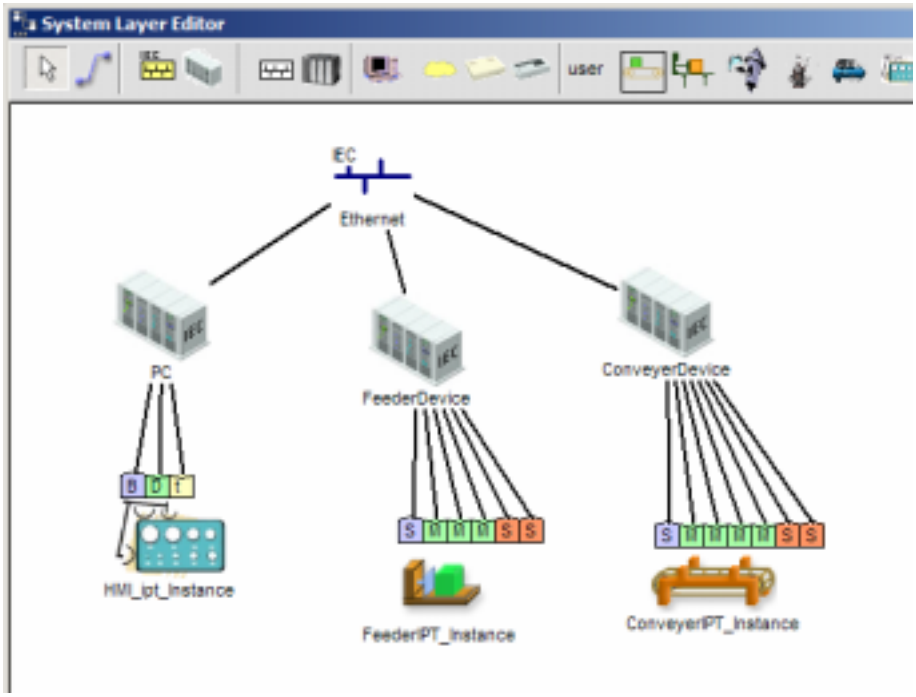


Figure 14. The system layer diagram of the application

Press the from the menu *CORFU/ Save Project* to save the project and the System Layer diagram.

Now we will go inside the FeederDevice and try to design a diagram similar to the Feed_and_Carry.pdf diagram on page 24 which describes the I/O control interface of the Feeder.
Select the FeederDevice, right click and select from the menu *Edit Process Interface FB Diagram.* The opened window now gives as the ability to design the internal I/O control interface diagram of the Feeder Control device (FeederDevice). On the left and right you see IPPs that are connected from the underlying process (FeederIPT) to the FeederDevice. On the left are Actuators and on the right are sensor IPPs. This signals (event and data) come from the underlying process. We will design now how exactly these signals come to the application.
Press the button New on the toolbar and bring the FB type DIO_8_8 from the library Yamatake. Right click on this and name the Function Block instance: DIO. Also add the FB types FLRT_BOOL_8_8 and FC_CTRL1_IOCTRL. Also from the vertical toolbar on the left insert in the diagram an E_SWITCH FB type (Event Switch 6th button), an E_CYCLE (Cyclic Event Generator 9th button), and E_DELAY type (Delayed Event Propagator 7th button). Although, the CORFU FBDK displays for clarity the Event FB types as small images, you can switch this by pressing on the toolbar the 3rd button *Toggle Event Stereotypes.* Press the 1st button *Save* and save the diagram with the name FeederDeviceInternal.dfb .
Press from the toolbar the 5th button *Make Connection.* Connect QO of DIO to QI of MSK in order to create a data connection between the Function Blocks. Connect the IND of MSK to the (blue) event-IPP SensorStatus. Continue to create the diagram shown on figure 15.
Pay attention to some IPPs that they don't exist in the FeederIPT_Instance. These are the actuator IPPs: INIT, QI , CT, DT and the sensor IPPs: QO, STATUS and INITO. These IPPs are created in the internal of the FeederDevice and they are needed to the Function Block diagram later. You create them by pressing the button *Insert Device IPP.* For example, to create the IPP INIT, press the button *Insert Device IPP*, in the name field type INIT, check *Is Event IPP,* select type *Actuator* and press OK.

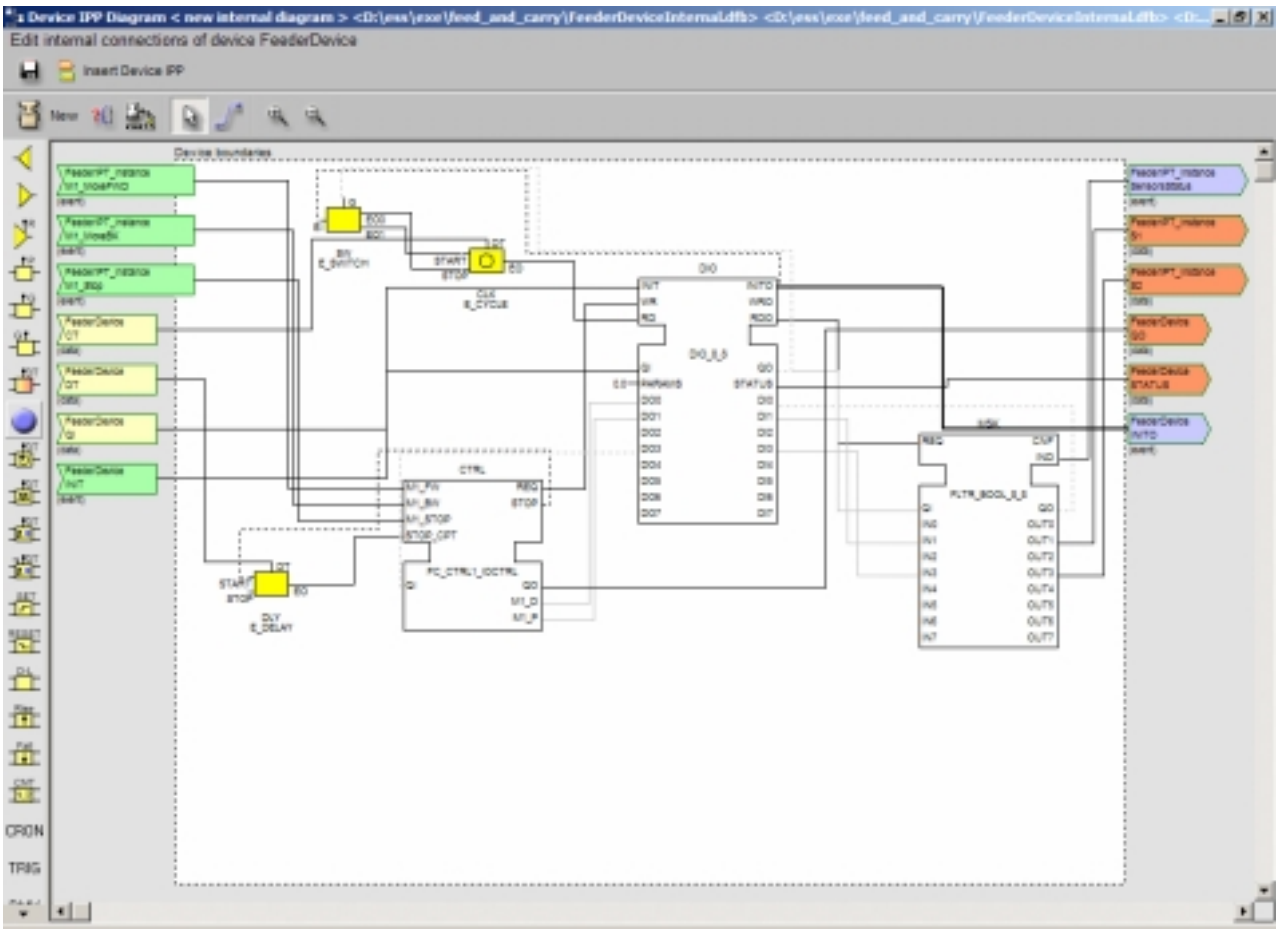


Figure 15. Process Interface FB Diagram of FeederDevice.

Close the window and press Yes to save the diagram.

From the tree view Project Browser, select Application/Function Block Layer/Function Block Diagrams/ Function Block Diagram (Imported) [1] and double click it to open the diagram. Observe the Feeder Function Block (like in figure 16) how it is connected with the IPPs. Now, it should be clear how these IPPs are available to the application. For example: The event M1_MoveFWD of the Function Block instance FeederControler, is connected to the (event) IPP FeederIPT_Instance.M1_MoveFWD. In a subsequent step the (event) IPP FeederIPT_Instance.M1_MoveFWD, comes from the Process Interface FB Diagram of the FeederDevice and is actually the event input M1_FW of the CTRL Function Block instance. The idea of the Process Interface FB Diagram comes to enhance the encapsulation of information and helps the engineer to focus on the design of the application. It is possible in future that these internal diagrams are available directly from the device vendor thus the engineer will focus only in the interface.

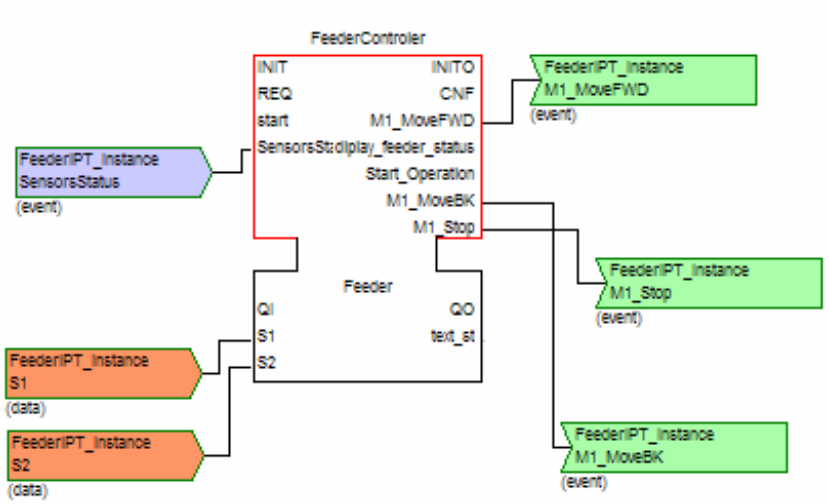


Figure 16. The Feeder Function Block and the connected IPPs

Continue now by designing the Process Interface FB Diagrams of PC and ConveyerDevice as shown in figures 17 and 18.

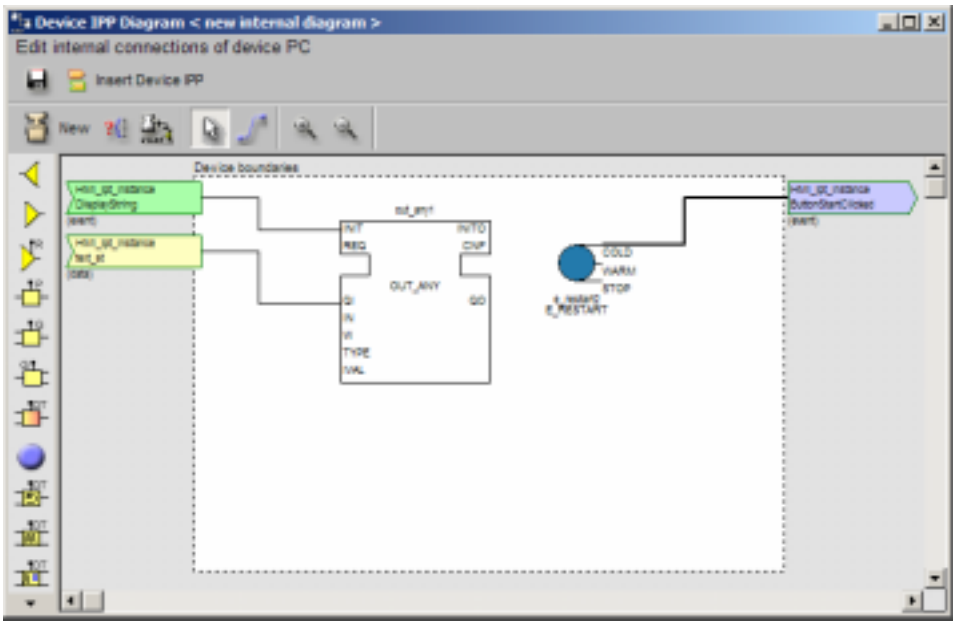


Figure 17. Process Interface FB Diagram of PC

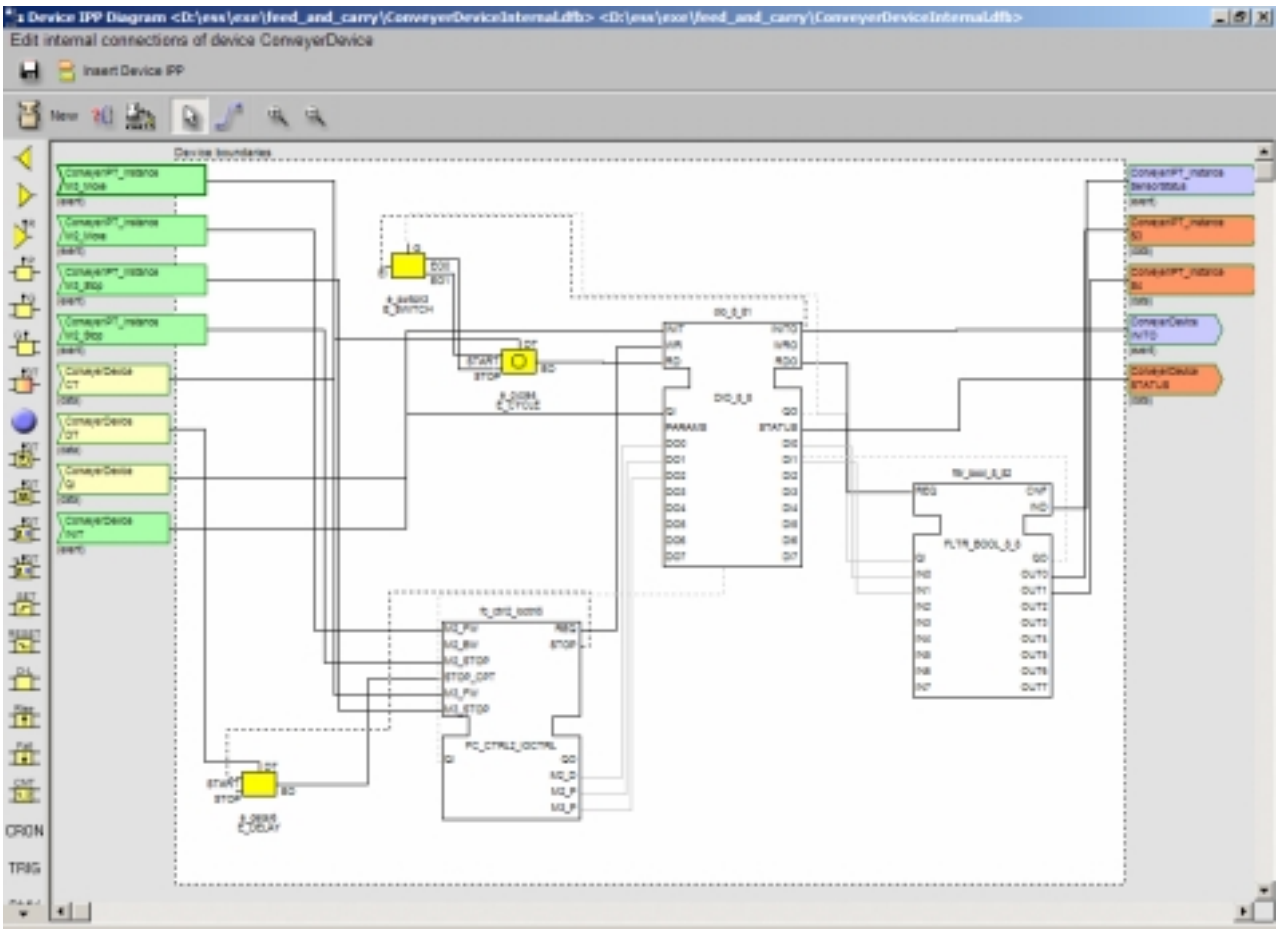


Figure 18. Process Interface FB Diagram of ConveyerDevice

After designing the internal diagrams of the devices, we must go back to the Function Block diagram and insert the IPPs that we created in the devices and have not been imported and connected automatically. We must do this process in order our application to be as much compete. From the tree view Project Browser, select Application/Function Block Layer/Function Block Diagrams/ Function Block Diagram (Imported) [1] and double click it to open the diagram. Press from the toolbar in the FB editor window, the button *Insert IPP.* Select from the tree the item ConveyerDevice and open it, as show in figure 19 Then select the item QI and press the button *Add Selected*. The IPP ConveyerDevice.QI will appear in the diagram. Connect it to the QO of the FB ConveyerController.

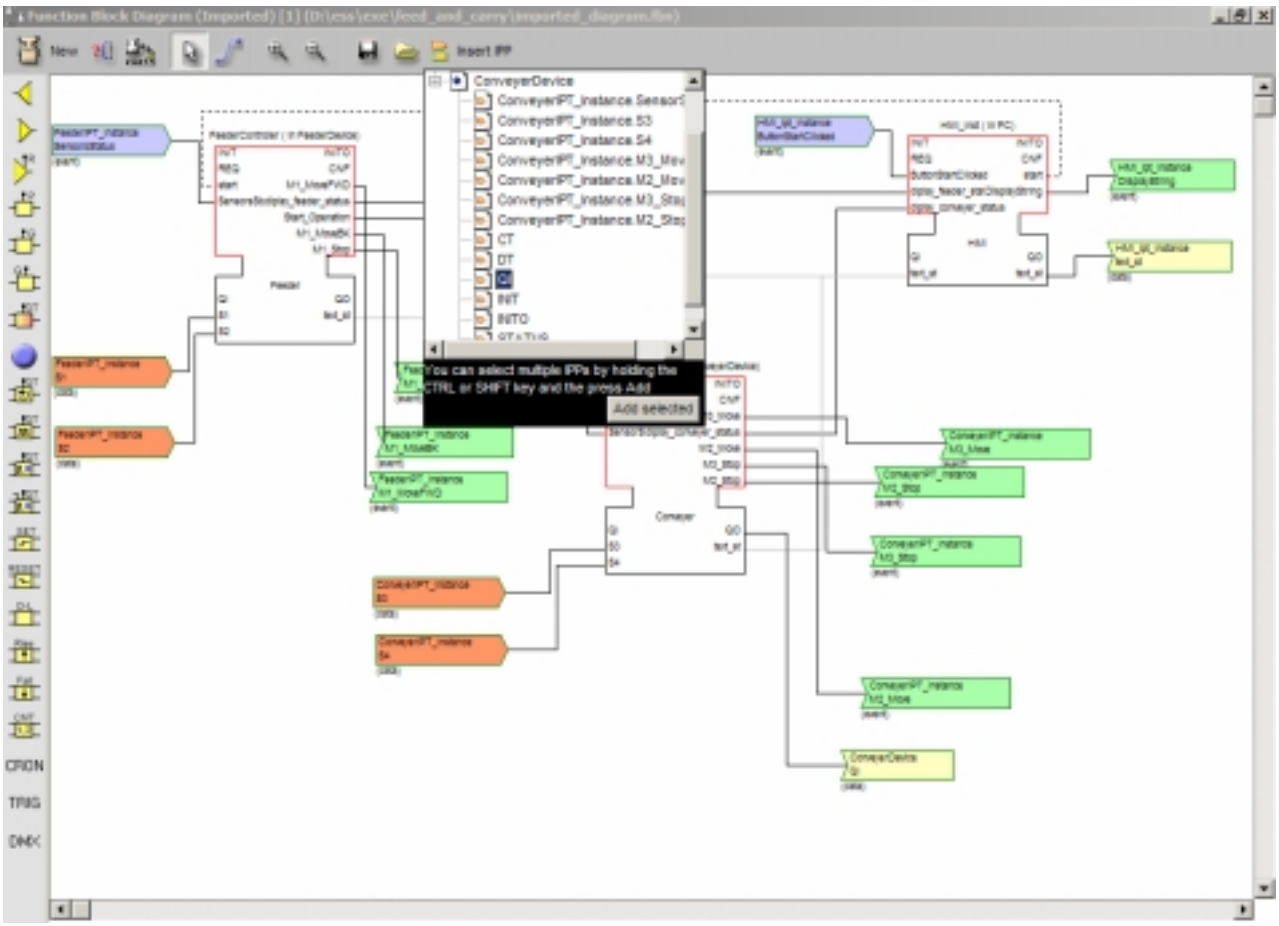


Figure 19. Adding an IPP to the Function Block editor

6.1.7 Distributing the application

The final step is to distribute the Function Block instances of our application to the actual devices. Open the Function Block diagram Imported and the System Layer editor. Then with holding the CTRL on your keyboard drag the FeederControler Function Block instance and drop it to the FeederDevice on the System Layer editor as shown on Figure 20. With this action the system assumes that the FeederControler instance will be downloaded to the FeederDevice. You can continue with the ConveyerControler and download it to the ConveyerDevice and the HMI_inst instance to the PC device.

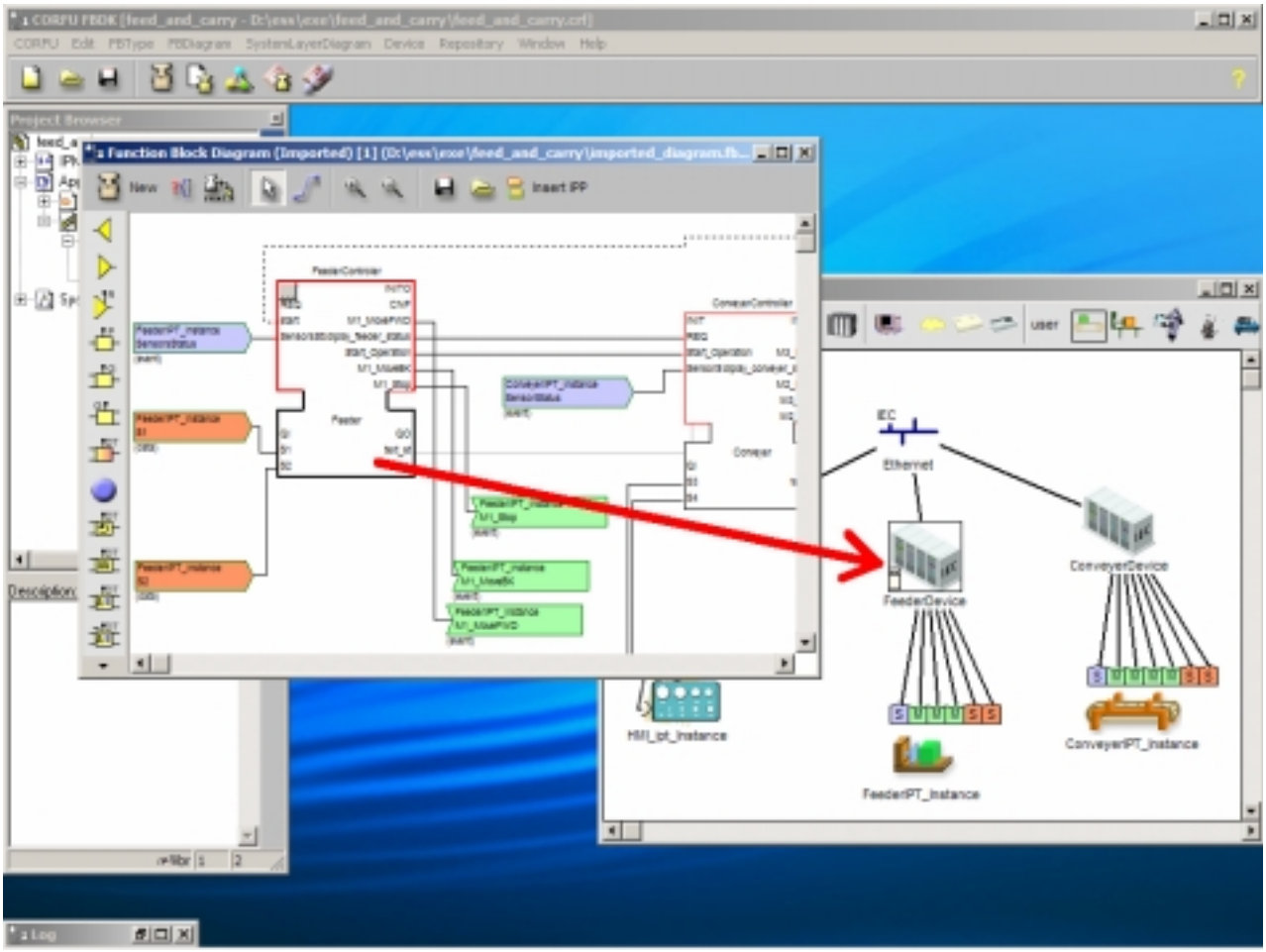


Figure 20. Downloading a Function Block instance to a device.

Double click on the FeederDevice and the properties window will appear. On the Downloaded FBs tab you can see the FeederControler instance. Select from the drop down list the type of the FeederDevice as *Ehternet TCP/IP compatible* and switch to the tab Advanced (see figure 21). Then enter the IP address of the device and the port to communicate. You can press the button Test Communication to check if the device is listening to that port.

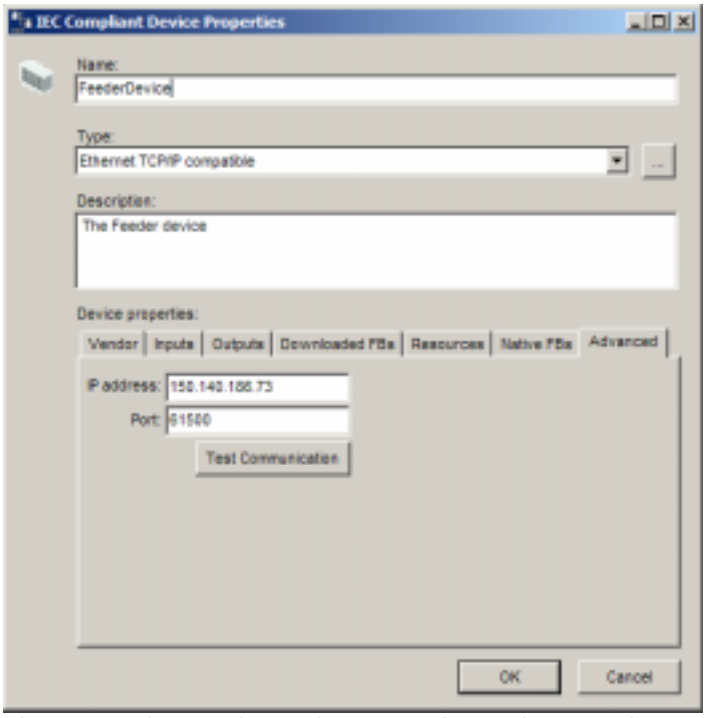


Figure 21. The FeederDevice properties and connection settings.

Press OK and close the window. Continue with the FeederDevice and PC.

After configuring the devices, press from the main toolbar the 9th button *Distribute Application* and the window Download Application Manager will open as shown in figure 22.

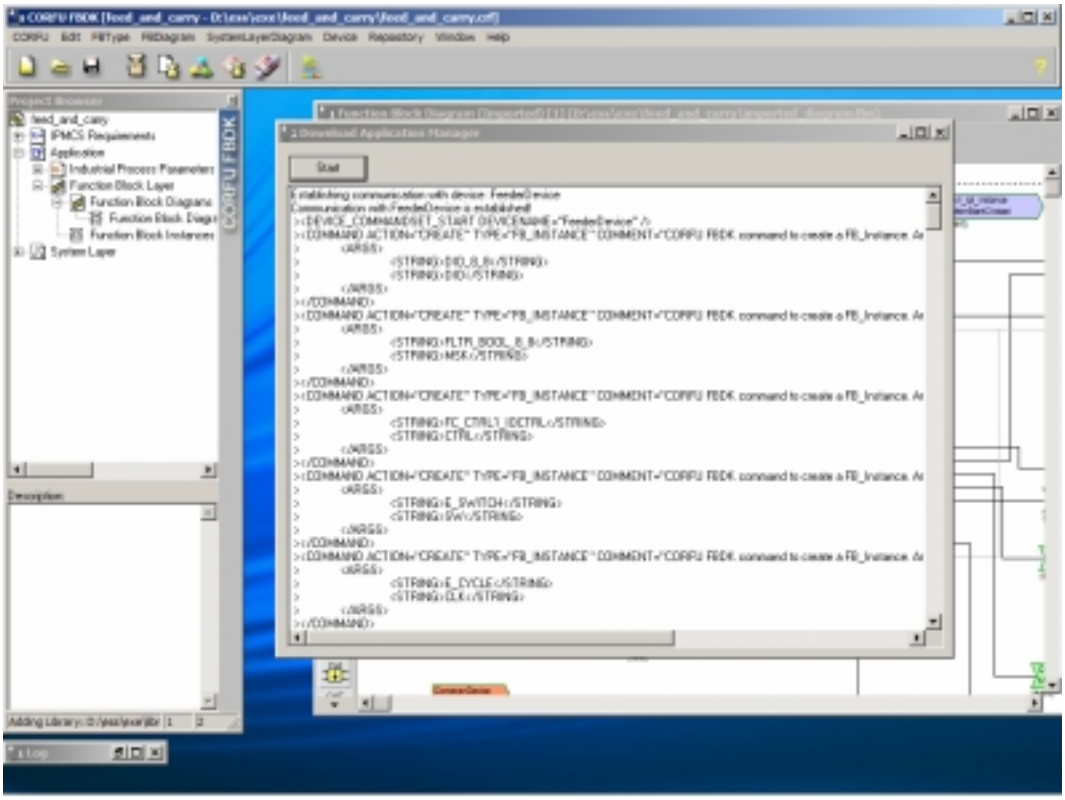


Figure 22. Communicating with the devices

Press the Start button and CORFU FBDK will start communicating with the devices. In order to test this example there is a small server application on our website, which you can download. Although this is experimental, the commands are in XML format and if a device can understand them, it is possible to create the application. You will also notice, if you run the example, that we don't create Publisher or Subscriber Function Blocks, but instead we send commands to the device, to publish or subscribe certain events or data. We assume that the devices are clever enough to create properly the connections.

The application can be found ready in the directory feed_and_carry and open the file **feed_and_carry.crf** or the open file **feed_and_carry without parsing.crf** which has ready only the object model in Rose.

6.2 Designing an application without an object model

In this paragraph we will describe on a step by step procedure, how we can design the *feed and carry* application. If you wish to now more about how to design it with the CORFU development process then read paragraph 6.1
We will start by creating a new project. Start CORFU FBDK and from the CORFU Start center select *New Project* or from the menu *CORFU/New CORFU project.* In the opened window type for the Project title: *feed_and_carry* and browse for a path to save the project file feed_and_carry.crf, for example D:\ess\feed_and_carry_no_model\feed_and_carry.crf, UNCHECK the *Connect to ROSE* checkbox and press the OK button. After a few seconds the file feed_and_carry.crf will be created in your folder. The window *Project Browser* on the left displays useful information. (figure 23)

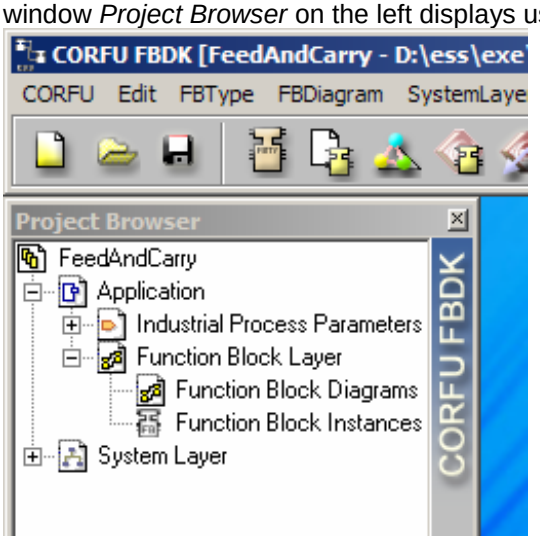


Figure 23. Project Browser

6.2.1 Designing the underlying system

Next, we will design the underlying process that our application will be executed. Click from the toolbar the 6th button to open the *System Layer Editor* window. Our example consists of an HMI control, a Feeder machine and Conveyer machine. Click from the *System Layer Editor* window the button Control Panel (3rd from the end). A small icon will appear that represents a Control Panel. Right click on it (or double click it) and select from the menu *Properties of ControlPanel1*. The Properties window will appear. Name the Control Panel as HMI_IPT (Human Machine Interface, Industrial Process Parameter). More information about IPTs and IPPs can be found on Section 7. Press the button Add to add an IPP named ButtonStartClicked, check the IsEventIPP, select datatype Boolean, select type as Sensor and press OK. Again press the button Add, to add an IPP named DisplayString, check the IsEventIPP, select datatype Boolean, select type as Actuator and press OK. Add finally an IPP named text_st, **don't** check the IsEventIPP (will be data IPP), select datatype String, select type as Actuator and press OK
Click from the *System Layer Editor* window the button *Feeder*. Name it Feeder_IPT and add the following IPPs:

SensorsStatus, IsEventIPP=YES, Datatype=Integer,type=Sensor
M1_MoveFWD, IsEventIPP=YES, Datatype=Boolean,type=Actuator
M1_MoveBK, IsEventIPP=YES, Datatype= Boolean,type= Actuator
M1_Stop, IsEventIPP=YES, Datatype= Boolean,type= Actuator
S1, IsEventIPP =NO, Datatype= Boolean,type=Sensor
S2, IsEventIPP=NO, Datatype= Boolean,type=Sensor

Press OK to close the properties of Feeder_IPT.

Click from the *System Layer Editor* window the button *Conveyer*. Name it Conveyer_IPT and add the following IPPs:

SensorsStatus, IsEventIPP=YES, Datatype=Integer,type=Sensor
M3_Move, IsEventIPP=YES, Datatype=Boolean,type=Actuator
M2_Move, IsEventIPP=YES, Datatype=Boolean,type=Actuator
M3_Stop, IsEventIPP=YES, Datatype=Boolean,type=Actuator
M2_Stop, IsEventIPP=YES, Datatype=Boolean,type=Actuator
S3, IsEventIPP =NO, Datatype= Boolean,type=Sensor
S4, IsEventIPP=NO, Datatype= Boolean,type=Sensor

Press OK to close the properties of Coneyer_IPT. You should have by now something similar like figure 24.

Figure 24. The IPT instances on System layer.

These 3 objects are the components of the underlying process. Press now from the System Layer Editor toolbar the 4th button *IEC compliant device.* An icon that represents an IEC device will be inserted. Right click on it and select Properties of IEC_Device0. Name it FeederDevice and close the window. Repeat and add a device named PC and another names ConveyerDevice. Press now from the System Layer Editor toolbar the 3rd button *IEC compliant fieldbus*, right click on it and name it Ethernet. Now click on the toolbar the 2nd button and switch to connection mode. Connect the FeederDevice to the first IPP SensorStatus of FeederIPT_Instance. Repeat to produce a diagram similar to Figure 25.

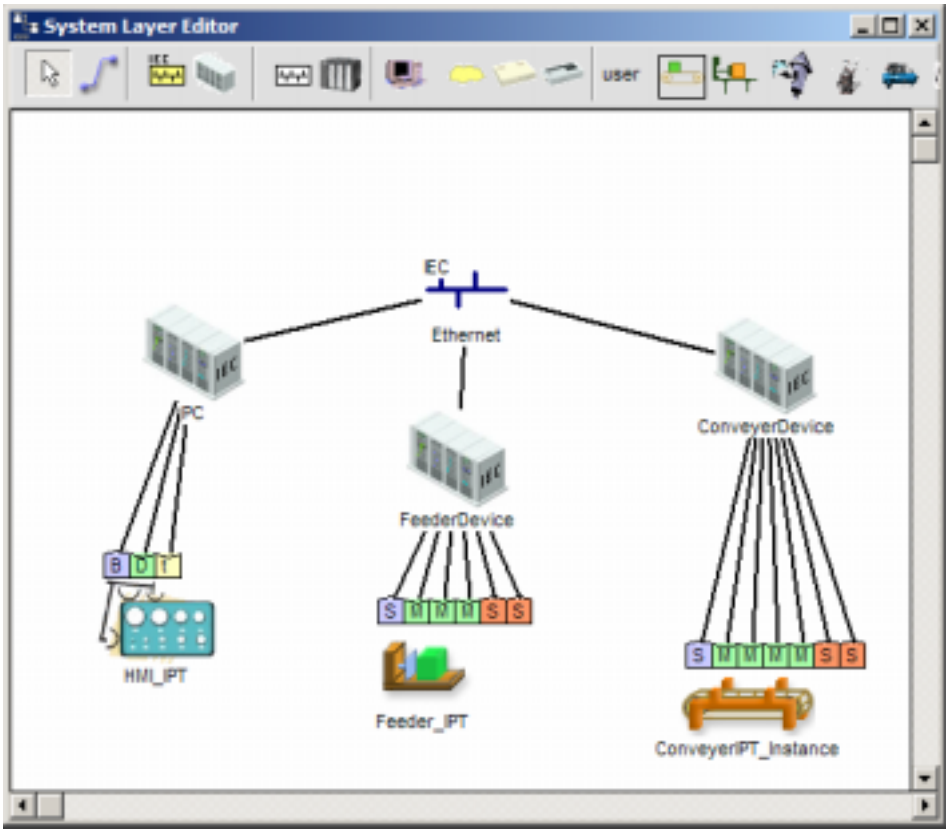


Figure 25. The system layer diagram of the application

Press the from the menu *CORFU/ Save Project* to save the project and the System Layer diagram.

Now we will go inside the FeederDevice and try to design a diagram similar to the Feed_and_Carry.pdf diagram on page 24 which describes the I/O control interface of the Feeder.
Select the FeederDevice, right click and select from the menu *Edit Process Interface FB Diagram.* The opened window now gives as the ability to design the internal I/O control interface diagram of the Feeder Control device (FeederDevice). On the left and right you see IPPs that are connected from the underlying process (FeederIPT) to the FeederDevice. On the left are Actuators and on the right are sensor IPPs. This signals (event and data) come from the underlying process. We will design now how exactly these signals come to the application.
Press the button New on the toolbar and bring the FB type DIO_8_8 from the library Yamatake. Right click on this and name the Function Block instance: DIO. Also add the FB types FLRT_BOOL_8_8 and FC_CTRL1_IOCTRL. Also from the vertical toolbar on the left insert in the diagram an E_SWITCH FB type (Event Switch 6th button), an E_CYCLE (Cyclic Event Generator 9th button), and E_DELAY type (Delayed Event Propagator 7th button). Although, the CORFU FBDK displays for clarity the Event FB types as small images, you can switch this by pressing on the toolbar the 3rd button *Toggle Event Stereotypes.* Press the 1st button *Save* and save the diagram with the name FeederDeviceInternal.dfb .
Press from the toolbar the 5th button *Make Connection.* Connect QO of DIO to QI of MSK in order to create a data connection between the Function Blocks. Connect the IND of MSK to the (blue) event-IPP SensorStatus. Continue to create the diagram shown on figure 26.
Pay attention to some IPPs that they don't exist in the FeederIPT_Instance. These are the actuator IPPs: INIT, QI , CT, DT and the sensor IPPs: QO, STATUS and INITO. These IPPs are created in the internal of the FeederDevice and they are needed to the Function Block diagram later. You create them by pressing the button *Insert Device IPP.* For example, to create the IPP INIT, press the button *Insert Device IPP*, in the name field type INIT, check *Is Event IPP,* select type *Actuator* and press OK.

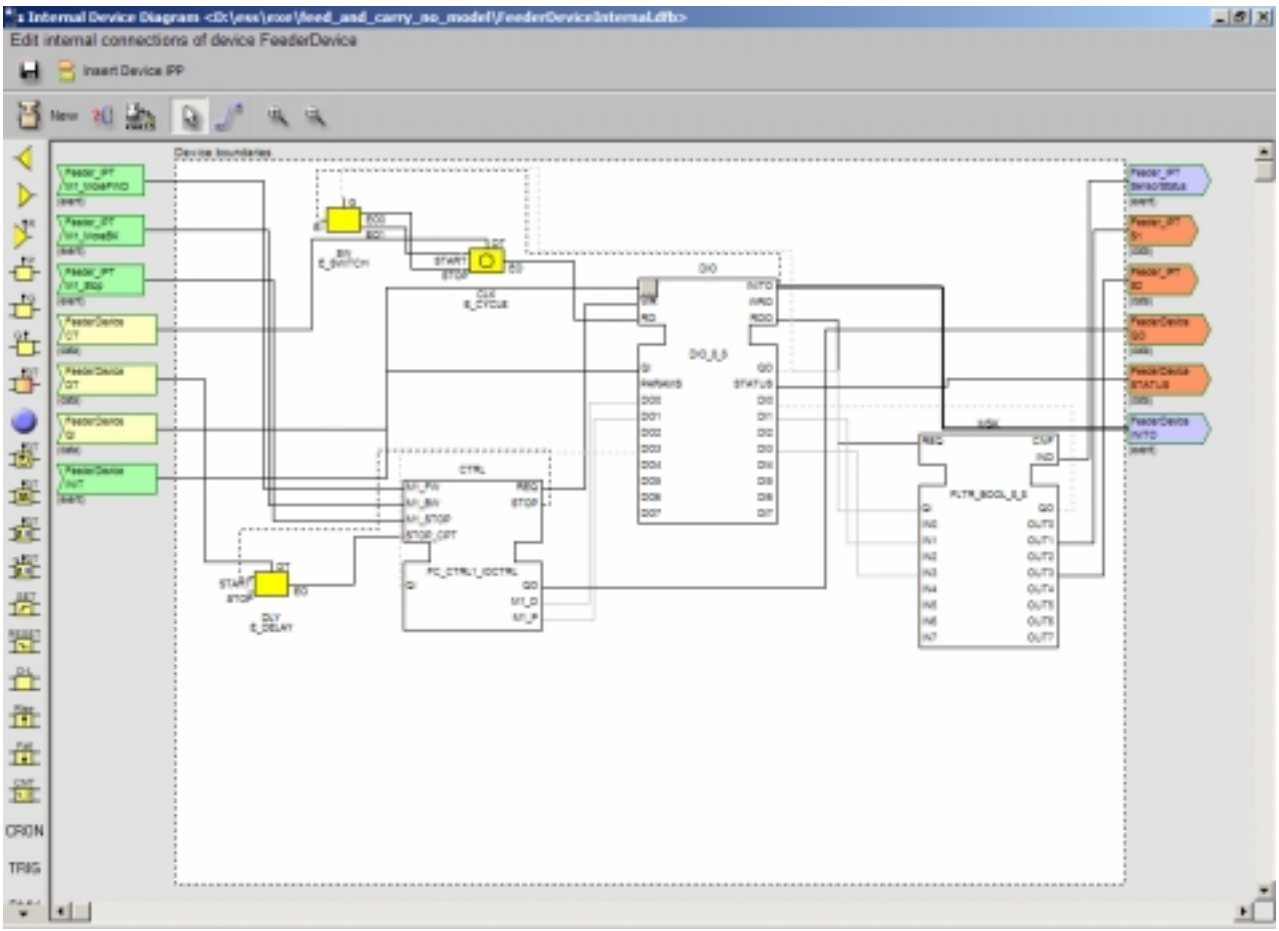


Figure 26. Process Interface FB Diagram of FeederDevice.

Close the window and press Yes to save the diagram.
Continue now by designing the Process Interface FB Diagrams of PC and ConveyerDevice as shown in figures 27 and 28.

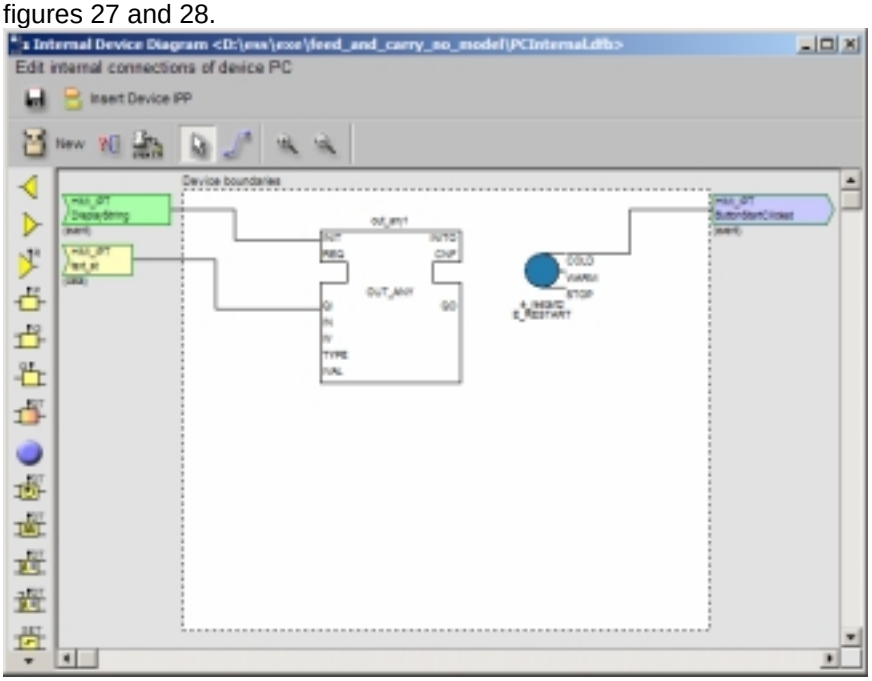


Figure 27. Process Interface FB Diagram of PC

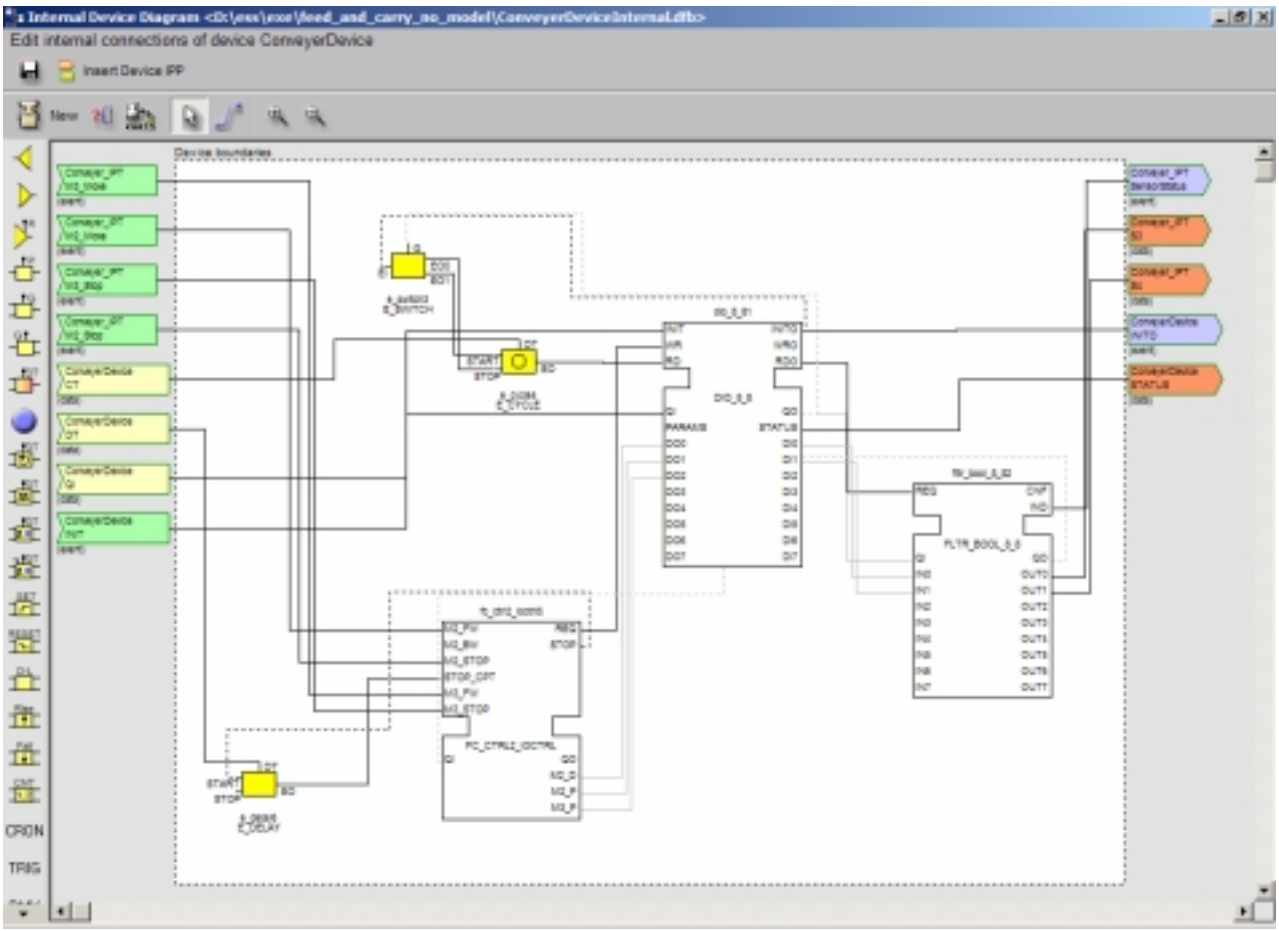


Figure 28. Process Interface FB Diagram of ConveyerDevice

6.2.2 Designing the application

After designing the underlying process we can go and design our application. Close the System Layer Window and select from the menu *FBDiagram/New,* and the Function Block Diagram editor will open. Press the button *New* (the menu displays available FB libraries) and from *Imported* select HMI. A function block instance HMI1 of FB type HMI will be created. Right click on it and name it *HMI_INST.* Insert also from library *Imported* a Feeder (name it FEEDER_INST) and a Conveyer (name it CONVEYER_INST). Press the 5th button *Make Connection* and make a connection between the event start of HMI_INST and event start of FEEDER_INST. Also the event Start_Operation of FEEDER_INST and event Start_Operation of CONVEYER_INST.
Another important task is to bring values (parameters) from the underlying process to our FB diagram. For example to bring the SensorStatus from the feeder machine to the FB FEEDER_INST. In order to do this, press from the menu the button *Insert IPP* and Select from the item FeederDevice the item Feeder_IPT.SensorStatus, press the button Add Selected and the IPP Feeder_IPT.SensorStatus (with blue color) will be added to the diagram. Then make a connection from the Feeder_IPT.SensorStatus to the event. Until now you will have something similar to figure 29.

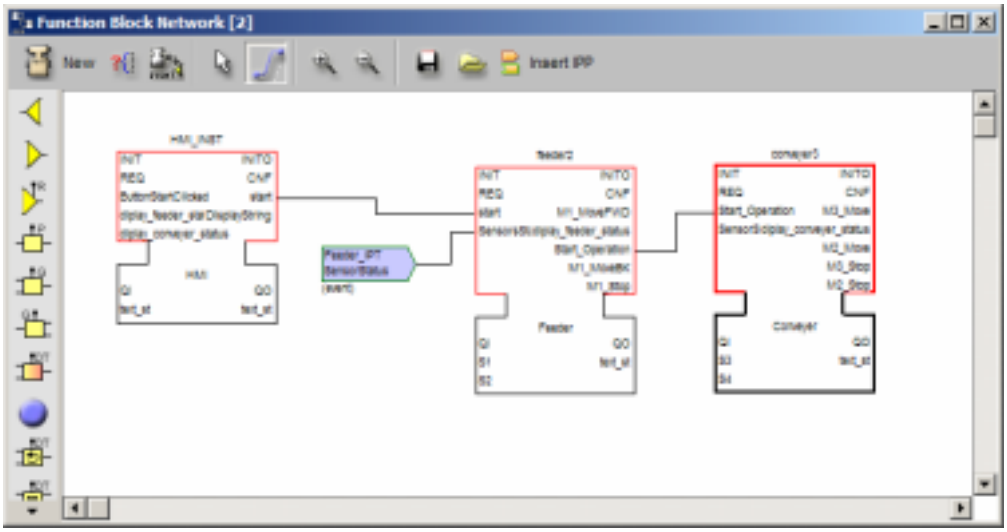


Figure 29. Making the FB diagram of the application

Press the 8th button *Save FB Network…* and save it with the name **main_fb_diagram .**
Continue until you create the FB diagram shown on figure 30.

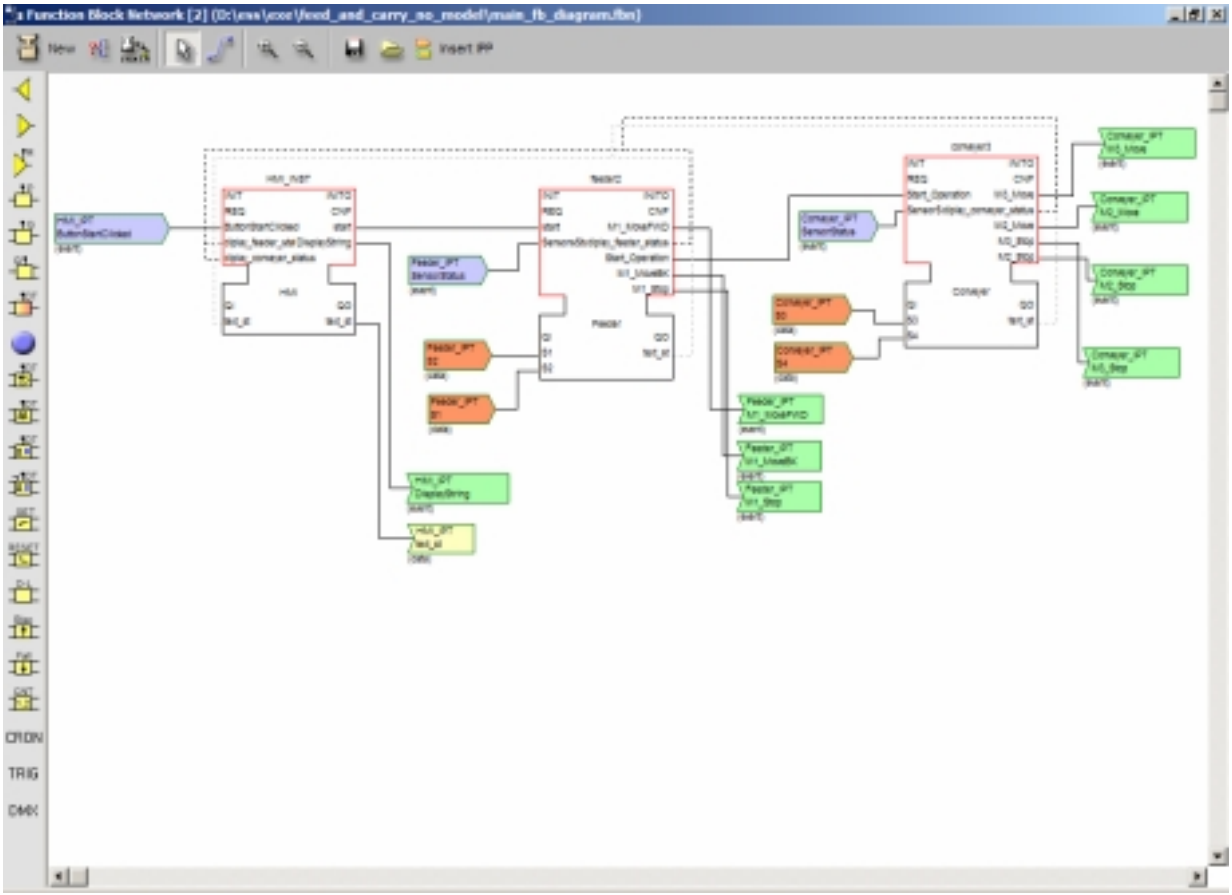


Figure 30. Final FB diagram

6.2.3 Distributing the application

The distribution of the application is similar to section 6.1.5, so refer to this section.

7 Frequently Asked Questions

7.1 What are Industrial Process Terminators (IPTs)?

Industrial Process Terminators are actually devices of the underlying industrial process, like Conveyers, HMIs, Computers, Boilers, etc. An Industrial Process Terminator is characterized by its Industrial Process Parameters.

7.2 What is an Industrial Process Parameter (IPP)?

An Industrial Process Parameter is a parameter of the underlying industrial process like a boiler's Temperature, speed of conveyer, Start/Stop events of a drill etc. We assume that IPPs are "high level" parameters, meaning that they are already measured values of the underlying process and not low-level values (like 8 or 16bits). IPPs can be located and produced either from a "clever" IPT or from an IEC 61499 compliant Device.

7.3 What is a Process Interface FB Diagram?

A Process Interface FB Diagram is like a black box with its interface IPPs of the underlying process. The idea of the Process Interface FB Diagram comes to enhance the encapsulation of information and helps the engineer to focus on the design of the application. It comes to hide information about specific transformation of the process values, that is not necessary to the engineer. For example the engineer wants to know a Temperature as an Integer and not that it comes from four digital outputs of a device. It is possible in future that these internal diagrams are available directly from the device vendor for wide area of applications, thus the engineer will focus only on interfacing his application with the underlying industrial process.

8 Additional Information and Comments

For more information, recommendations or problems found, please contact us on:
tranoris@ee.upatras.gr
thrambo@ee.upatras.gr

References

[1] K. Thramboulidis, "Development of Distributed Industrial Control Applications: The CORFU Framework", 4th IEEE International Workshop on Factory Communication Systems, August 2002, Vasteras, Sweden.
[2] C. Tranoris, K. Thramboulidis, "From Requirements to Function Block Diagrams: A new Approach for the design of industrial applications", 10th IEEE Mediterranean Conference on Control and Automation, MED'02.
[3] K. Thramboulidis and C. Tranoris, "Developing a CASE Tool for Distributed Control Applications", International Journal of Advanced Manufacturing Technology (forthcoming).
[4] C. Tranoris, and K. Thramboulidis, "An IEC-compliant Engineering Tool for Distributed Control Applications", 11th Mediterranean Conference on Control and Automation - MED'03. Rodos, Greece.
[5] www.rational.com
[6] www.omg.org