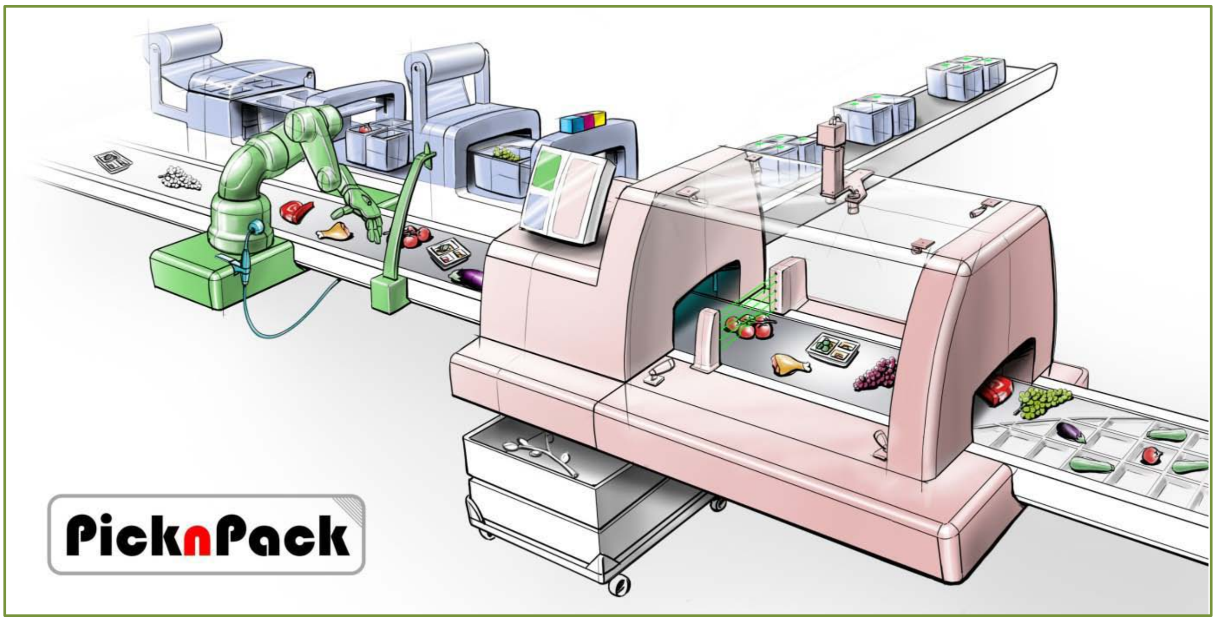
D3.6 – Report on Traceability System Integration

Zhipeng Wu (UM), Zhaozong Meng (UM) and John Gray (UM)
3/25/2015



Flexible robotic systems for automated adaptive packaging of fresh and processed food products

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n° 311987.

Dissemination level		
PU	Public	X
PR	Restricted to other programme participants (including the EC Services)	
RE	Restricted to a group specified by the consortium (including the EC Services)	
CO	Confidential, only for members of the consortium (including the EC Services)	

Table of Contents

1. Introduction

This document outlines the progress of the traceability system design and implementation for WP3 on traceability system integration and in particular the tasks defined in the WP3.4:

- Task 3.4: Integration (M30)
    - Integrate database, software development, RFID systems with sensor modules in WP4
    - Apply the hardware and software with intelligent production capability for automation control and robotic packaging, interact with WP4
    - Test, evaluate and optimise the traceability system

The associated milestone has been partially achieved and it is discussed in this report:

- M3.4: Optimised and complete integrated traceability system (M36)

The RFID enabled traceability system described in D3.3 and D3.4 is integrated to the line for command and data request with ZeroMQ(ZMQ or 0ZM), Life Cycle State Machine (LCSM), and JavaScript Object Notation (JSON).

The functions for integration of the traceability system are implemented, such as application interface for other modules in the line to access information in WP3 Database, ID broadcasting, and device and line state monitoring for state machine operation. But real device test with other modules depends on the progress of the individual modules, which will be carried out in the next step.

Further details are provided in the following sections.

2. Technologies and Standards

Since the traceability system communicates with the line controller and other individual modules, it needs to follow the technologies and standards to pursue interoperability. The cross-platform software tools ZMQ and JSON are recommended by WP2 for the communication between the modules in the line. The LCSM concept is selected to handle the events and functions for the line and separate modules, and spotlight protocol is used to deal with the running state between the line and the modules.

2.1 ZMQ

ZMQ is a high-performance asynchronous messaging library aimed at use in scalable distributed or concurrent applications. It provides sockets that carry atomic messages across various transports like in-process (INROC), inter-process (IPC), TCP, and multicast (PGM) [1]. Detailed introduction and resources can be found at the official site of ZMQ[2].

For implementation and test of the traceability system, the ZMQ pattern - Paranoid Pirate Pattern (PPP), which is suitable for request-reply with heartbeating is employed. The diagram of PPP is as shown in Figure 1.

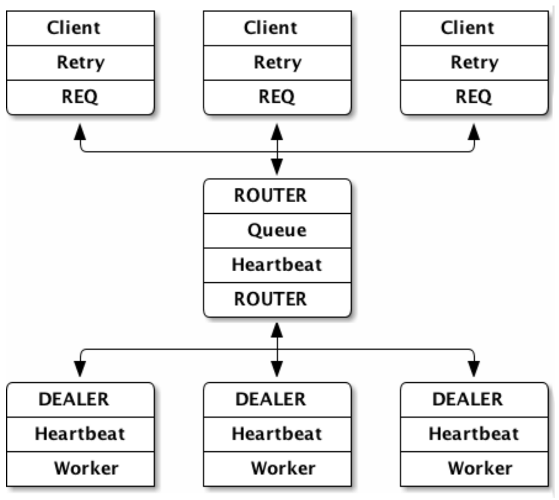


Figure 1 ZeroMQ Paranoid Pirate Pattern

In the diagram, the traceability system works as a DEALER. It sends heartbeats to ROUTER and monitors heartbeats from ROUTER. Other modules can access data from the Database of traceability system through the ROUTER.

Since the traceability system is developed with Microsoft Visual C# .NET, there are Dynamic Link Libraries (DLLs) for the implementation of ZMQ in this development platform [3].

2.2 JSON

JSON is a text format that facilitates structured data interchange between all programming languages. It is a light-weight data-interchange format which is easy for human to read and write and easy for machines to parse and generate[4].An example of JSON code snippet is given in List 1.

```
{"Products":[{
        "Product1":{
                "ProductName":"Tomato",
                "Weight":"250g",
                "Price":"£1.5",
                "BestBefore":"20/03 /2015"},
        "Product2":{
                "ProductName":"Grape",
                "Weight":"250g",
                "Price":"£2.5",
                "BestBefore":"21/03/2015"},
        "Product3":{
                "ProductName":"Apple",
                "Weight":"1000g",
                "Price":"£1.99",
                "BestBefore":"22/03/2015"}}]
}
```
List 1 Code Snippet of JSON Format

For the Microsoft Visual C# .NET, a popular high-performance JSON framework – Newtonsoft Json.NET [5] is employed. Json.NET is open source under the MIT license and free for commercial use.

2.3 LCSM

Finite State Machine(FSM) is a modelling approach for software development to reduce software development effort and improve quality[6].

The functions of the software can be divided into different states which transfer between the states according to the trigger signalsandconditions. The corresponding actions are executed when the state transfers from one to another.The events, actions, and states of the modules in the line can be modelled with the LCSM and then executed strictly with the pre-defined logic.

2.4 Spotlight Protocol

The spotlight protocol is an event-based protocol which is used to deal with the running state between the line and module[7].

- GREEN: continue running with business as usual
- ORANGE: continue running when triggered externally
- RED: pause and only restart when triggered by GREEN light

The traceability system needs to follow the spotlight protocol to synchronise its state with the line.

The integration work of the traceability system is based on the technologies listed above. The command message and data request message are implemented with ZMQ, command and value information are wrapped and parsed with JSON format. The state control and event handling are modelled with FSM. The design and implementation are illustrated in sections 3 and 4.

3. State Machine Design and Implementation

In order to synchronous the module of traceability system with the line, the events, activities, and states are modelled with a finite state machine. The design of the FSM and its implementation in Visual C# .NET platform is described in this section.

3.1 State Machine Design

The design of the FSM is based on its events and activities of the traceability application. Forthe traceability systemof current version, it is separated into five states: Configure, Ready/Waiting, Running, Pause, and Interrupt:

- Configure: the system initiates and configures hardware devices, such as ZMQ connection, RFID reader initiation and configuration, packaging setting, job batch setting, input/output container registration, etc.
- Ready/Waiting: waiting when hardware devices and settings are ready
- Running: working
- Pause: stopped because of container empty, needs user to restart
- Interrupt: stopped because container removed, auto-start when container is found

The events in traceability system can be classified into four types:

- Device and setting state event
- Line command event
- User UI command event
- RFID detected container event

The transfer of states of the state machine is triggered by these events.In the state machine, the states are set with transitions "e[g]/a" [7,8], where

- e=event
- [g] = guard condition
- a=action

A diagram of the state machine is shown in Figure 2.

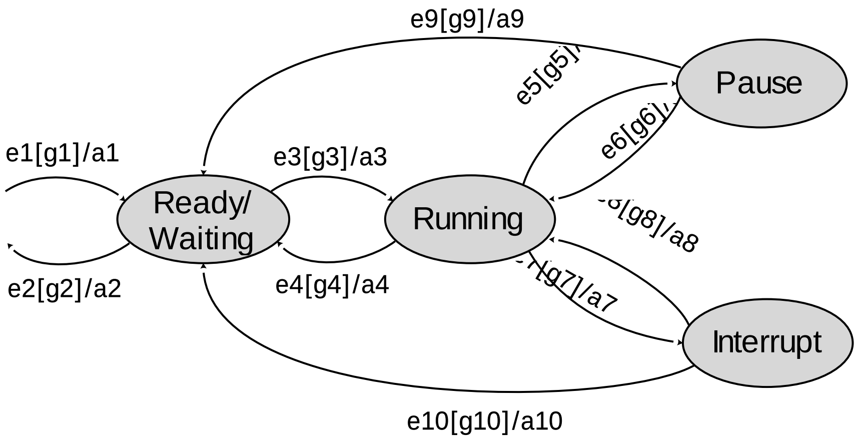


Figure 2 FSM of the Traceability System

The events, guard conditions, and actions in Figure 2 are listed in Table 1.

Table1. States, Events, Conditions, and Actions of Traceability System State Machine

States		Events	Guard Conditions	Actions
Configure	e1[g1]a1	RFID reader1 event RFID reader2 event Line state event Packaging setting event Job batch setting event	RFID reader1 state==true AND RFID reader2 state==true AND Line state==true AND Packaging setting state==true AND Job batch setting state==true	(1)Set stateReady/Waiting (2)Notify line (3)Monitor device events (4)Monitor setting events (5)Monitor line command (6)Reply data request
Ready/ Waiting	e2[g2]a2	RFID reader1 event RFID reader2 event Line state event Packaging setting event Job batch setting event	RFID reader1 state==falseOR RFID reader2 state==falseOR Line state ==falseOR Packaging setting state ==falseOR Job batch setting state==false	(1)Set stateConfigure (2)Notify line (3)Monitor device events (4)Monitor setting events (5)Reply data request
	e3[g3]a3	Start button click event Line command event	Line state == RunningAND Containers ready == true	(1) Set state Running (2)Notify line (3)Monitor device events (4)Reply data request (5)Monitor device events (6)Monitor line command (7)Monitor container state (8)RFID event handling
Running	e4[g4]a4	Stop button click event Line command event	(1) For Stop button event, / (2) For Line command event, Line state = Stop	(1) Set state Ready/Waiting (2)Notify line (3)Monitor device events (4)Monitor setting events (5)Monitor line command (6)Reply data request
	e5[g5]a5	Line command event Container event	Line command ==PAUSE OR Container state==Empty	(1) Set state Pause (2)Notify line (3)Monitor device events (4)Reply data request (5)Monitor device events (6)Monitor line command
	e7[g7]a7	Container event	Container state==Removed	(1) Set state Interrupt (2)Notify line (3)Monitor device events (4)Reply data request (5)Monitor device events (6)Monitor line command
Pause	e6[g6]a6	Restart button click event Container event	(1) For Restart button clicked / (2) For Container event Container empty==false	(1) Set state Running (2)Notify line (3)Monitor device events (4)Reply data request (5)Monitor device events (6)Monitor line command (7)Monitor container state (8)RFID event handling
	e9[g9]a9	Stop button click	(1) For Stop button clicked	(1) Set state

		event Line command	/ (2) For Line command Line state==Stop	Ready/Waiting (2)Notify line (3)Monitor device events (4)Monitor setting events (5)Monitor line command (6)Reply data request
Interrupt	e8[g8]a8	Container event	Container found ==true	(1) Set state Running (2)Notify line (3)Monitor device events (4)Reply data request (5)Monitor device events (6)Monitor line command (7)Monitor container state (8)RFID event handling
	E10[g10] a10	Stop button click event Line command event	(1) For Stop button clicked / (2) For Line command Line state==Stop	(1) Set state Ready/Waiting (2)Notify line (3)Monitor device events (4)Monitor setting events (5)Monitor line command (6)Reply data request

The events listed in the Table 1 are monitored by Window UI, and TCP socket from RFID reader, and ZMQ message from the line. In the Microsoft Visual C# environment, since the application has already been complicated, the handling of multi-thread communication is important for the performance of the traceability application.

3.2 State Machine Implementation

The major concerns in implementation of the state machine in Microsoft Visual C# .NET platform are:

(1) Event observing
Events can be triggered by ZMQ message, RFID reader tag event, and UI operation. The events need to be observed in real-time and passed to the main thread for state transfer.

(2) Multi-thread handling
There are multiple threads in the traceability application such as main UI thread, RFID reader monitoring thread, tag messaging processing thread, ZMQ message receiving thread,etc. Cross-thread communication isneeded for some use cases.

The state machine designed in 3.1 is implemented in the traceability system.

4. Traceability Module Data Sharing

Since the traceability system collects the source information, product information, logistic information of the products in the Database, application interfaces is expected for other modules to access the Database information. Anapplication interface is provided for Database information sharing based on the ZMQ messages.

4.1 ZMQ Interfaces for Data Request Operations

The ZMQ based message interface for data request operation is as shown in Figure 3.

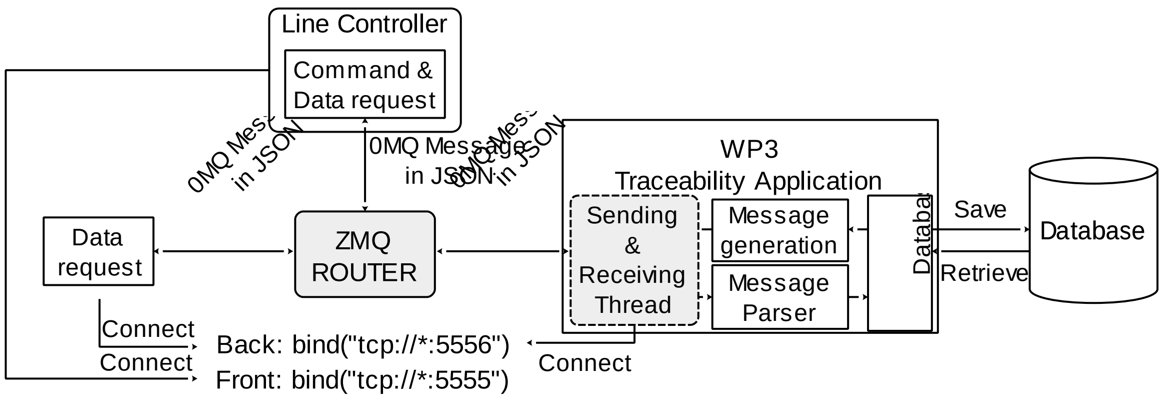


Figure 3 ZMQ Interface for Data Request

As shown in Figure 3, the traceability application connects to a TCP port as a ZMQ DEALER, and modules requesting data also connected to a ZMQ ROUTER. The modules send a ZMQ message in JSON to the traceability application. The traceability application receives the message, parses the JSON message, executes the request, wraps the data in JSON, and then replies the message to the requester. On the other hand, the traceability system can also request data from other modules in the line. The line controller works as a ZMQ Client.

In the data interface, the ZMQ ROUTER plays a very important role. It forwards module X's request to the traceability application and the forwards traceability application's reply back to module X. It also forwards traceability applications data request to module X and forward back module X's reply. In addition, it forwards traceability application's broadcasting information to all connected modules. The following code snippet illustrates how the ZMQ ROUTER works.

```
switch(incoming[1].ReadString())
{
        case"DataRequest": //For data request
            using (var outgoing = newZMessage())
            {
            outgoing.Add(newZFrame(incoming[2].ReadString()));//Object module
            outgoing.Add(newZFrame(incoming[1].ReadString()));//Data request mark
            outgoing.Add(newZFrame(incoming[4].ReadString()));//Request message content

            backend.Send(outgoing);
            Console.WriteLine("I:      sending  data  request  to:{0}:{1}",  incoming[2].ReadString(),
            incoming[4].ReadString());
            }
            break;
        case"BROADCAST": //For ID broadcasting
            foreach (Workerworkerin workers) //Broadcast to all connected modules
            {
                using (var outgoing = newZMessage())
                {
                    outgoing.Add(ZFrame.CopyFrom(worker.Identity)); //Object module ID
                    outgoing.Add(newZFrame(incoming[1].ReadString())); //Broadcasting mark
                    outgoing.Add(newZFrame(incoming[4].ReadString())); //Broadcast message

                    backend.Send(outgoing);
                    Console.WriteLine("I:       sending    broadcasting    to:{0}:{1}",    worker.Identity,
                    incoming[4].ReadString());
                }
            }
        break;
        ...
        default: break;
}
```

List 2 ZMQ ROUTERHandling Message Forward in C#

4.2 Data Formats of ZMQ Messages

Both the requester and the replier needs to follow the same format which is clearly defined, so that they can parse the message from the other and the message they sent can be correctly parsed by other modules.

4.2.1 Share Data withother Modules

The traceability system shares data with other modules with a ZMQ interface. When a data request message is received, it will reply the data accessed from Database to the requester. The format of request message from other modules is designed as follows in List 3.

```
{
        "MessageType":"Data_Request",
        "msgFrom":"ModuleX",
        "msgTime":"27/03/2015 10:30:45",
        "MessageInfo":{
                "RequestType":"RFID",
                "RequestValue":"E2001063100801270700D0D0"
        }
}
```

List 3Data Request Message Format

The reply message format is designed as follows in List 4.

```
{
        "messageType":"Data_Request_Reply",
        "requestType":"RFID",
        "moduleName":"PnP_RFID",
        "returnTime":"27/03/2015 10:30:45",
```

```
        "objectType":"SmallPackage",
        "sourceInformation":{
                "BatchName":"Tomato26Sep(ID:
                19)",
                "SupplierName":"Supplier03333",
		"supplierGLN":"3405353",
		"receivedTime":"26/09/2014
		00:18:01",
		"productName":"Tomato_Raw",
		"inputGTIN":"500695898590",
		"categoryName":"Tomato",
		"InSCC":"104689981846061452",
		"IncontainerType":"In_Container"
		,
		"InternalContainerID":"6"
	},
	"productInfomation":{
		"outputTypename":"Tomato_Ra
		w(Lot No.:0100100003)",
		"inFillTime":"26/09/2014
		00:30:10",
		"lotNumber":"0100100003",
		"operatorName":"Zhaozong",
		"outputGTIN":"500695898590",
		"productionLineID":"1",
		"plantName":"Plant01962",
		"plantGLN":"9099393",
		"fixPrice":"True",
		"priceRate":"0.5000",
		"netWeight":"500",
		"price":"0.5000",
		"extendBarcode":"00050",
		"registerTime":"26/09/2014
		00:38:03",
		"externalContainerID":"16"
	},
	"deliveryInformation":{
		"outSSCC":"37694922609231149
		8(OutUnitID:4)",
		"plantName":"Plant01962",
		"dispatchGLN":"9099393",
		"departureTime":"26/09/2014
		01:00:01",
		"destination":"Customer9910",
		"destinationGLN":"8736134"
	}
}
```

List4 Reply Message Format for Data Request

4.2.2 Request Data from other Modules

The traceability system also provides interface to request data from other modules such as the WP4 DAQ module. When information from other modules is useful, a request message is sent to the object module with ZMQ. The JSON message format for data request is designed as shown in List 5.

```
{
	"messageType":"DataRequest",
	"mdouleName":"PnP_RFID",
	"objectModule":"PnP_DAQ",
	"sendTime":"27/03/2015 10:49:54",
	"msgInfo":{
		"requestType":"ProductQuality",
		"productID":"20150327104954823"
	}
}
```

List 5 Data Request Message Format

When the reply message is returned to the traceability application, the message is parsed with JSON parser and the information is presented to the user interface.

4.2.3 Unique Product ID Broadcasting

The Unique product ID (birth time) can be broadcast to the line when the RFID reader observes the new product, this function is implemented in the traceability application.

The broadcast function is triggered by the RFID tag event by TCP port message from the RFID reader. When the RFID reader detects a new product, the 17 digits time string (in format: yyyyMMddHHmmssfff) is considered the unique ID for this new product and is broadcast to the line. Broadcasting message format is given in List 6.

```
{
        "messageType":"Broadcasting",
        "mdouleName":"PnP_RFID",
        "sendTime":"27/03/2015 10:49:54",
        "msgInfo":{
                "tagID":"000011112222333344445555",
                "uniqueID":"20150327104954823"
        }
}
```

List 6Broadcasting Message Format

5. Test

The integration functions of the traceability system implemented are tested with the ZMQ ROUTER, Client, and DEALER. This section gives the tested of thefunctions with screenshots.

5.1 Functions Implemented

Functions implemented in the latest version are listed as follows:

(1) ZMQ receive command (in JSON) from/reply message (in JSON) to line controller
(2) State machine implementation
(3) Receive data request and reply Database information (in JSON) with ZMQ
(4) Request data from other modules (in JSON) with ZMQ
(5) Unique ID broadcasting (in JSON) with ZMQ

For cross-module interoperability, all interaction operations are in JSON format with ZMQ messaging mechanism.

5.2 Test Method

The functions described in the above sections are implemented in the traceability system. The system is integrated to the line with ZMQ, JSON, and LCSM technologies. It is tested with the PPP pattern in the following way:

(1) Traceability system works as a ZMQ DEALER

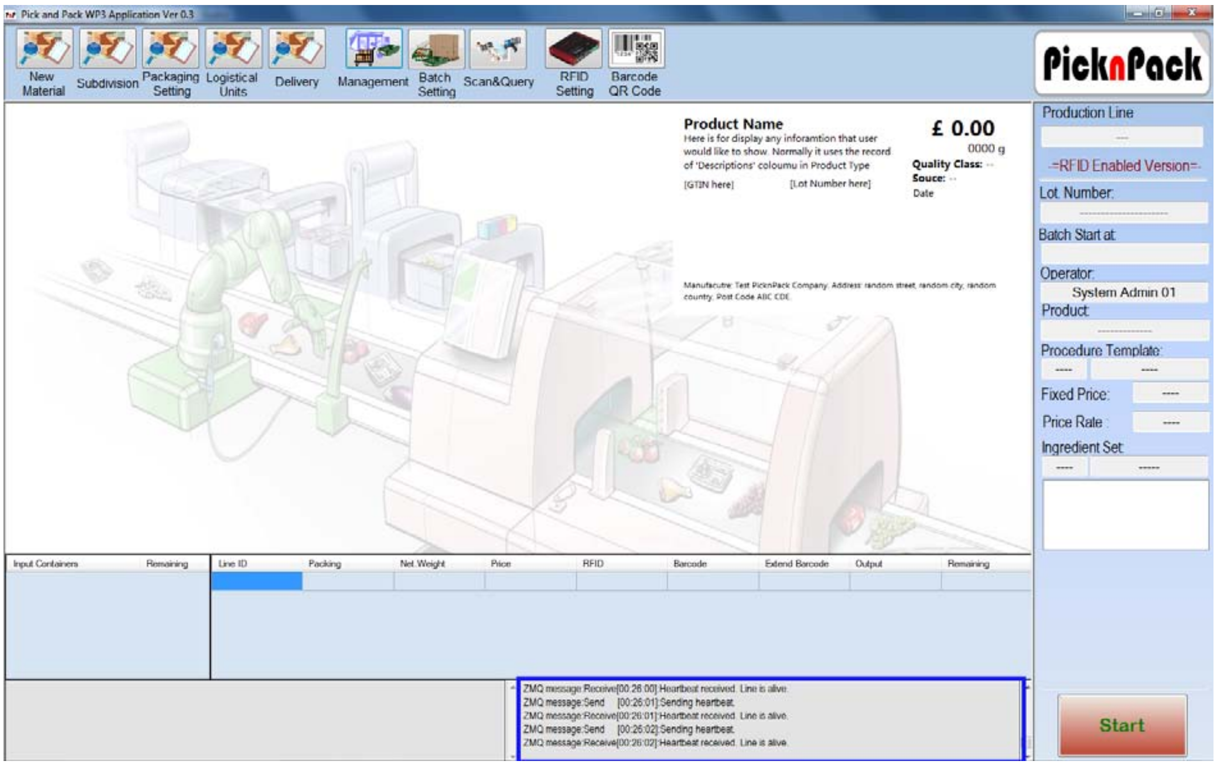
(2) A ZMQ ROUTER is implemented to forward messages
(3) A ZMQ Client is implemented as the line controller to send request and receive reply
(4) ZMQ DEALERs works as other modules
(5) The traceability system main interface

The applications designed and used for the test are: traceability application, ZMQ ROUTER as a router to forward messages, ZMQ Client as line controller, and ZMQ worker as other modules.

(1) Traceability Application Interface

In the main interface of the system, a message box as highlighted in the blue square (bottom right) is added to display the message received from the ZMQ interface.

This message box in the main interface is used to notify the users all sending and receiving messages related to the ZMQ communication.



Product Name
Here is for display any inforamtion that user would like to show. Normally it uses the record of 'Descriptions' coloumu in Product Type
[GTIN here] [Lot Number here]
£ 0.00
0000 g
Quality Class: --
Souce: --
Date
Manufacutre: Test PicknPack Company. Address: random street, random city, random country. Post Code ABC CDE

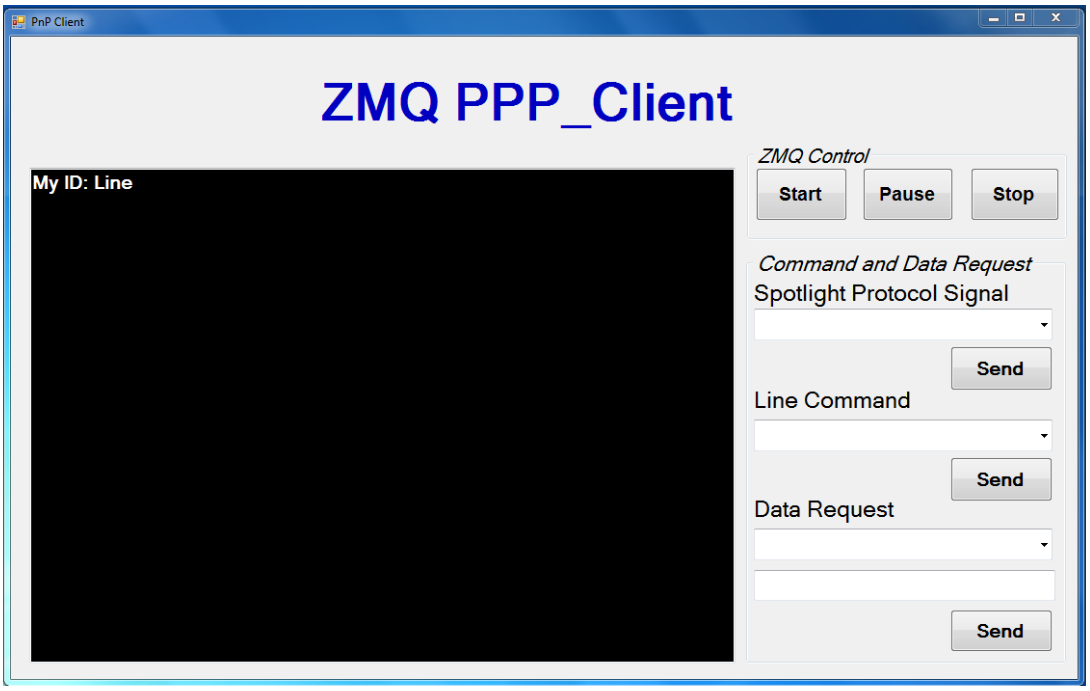
Production Line: ---
-=RFID Enabled Version=-
Lot Number:
Batch Start at:
Operator: System Admin 01
Product:
Procedure Template: ---- ----
Fixed Price: ----
Price Rate: ----
Ingredient Set: ---- -----

```
ZMQ message:Receive[00:26:00]:Heartbeat received. Line is alive.
ZMQ message:Send     [00:26:01]:Sending heartbeat.
ZMQ message:Receive[00:26:01]:Heartbeat received. Line is alive.
ZMQ message:Send     [00:26:02]:Sending heartbeat.
ZMQ message:Receive[00:26:02]:Heartbeat received. Line is alive.
```

Figure 4 Traceability System Main Interface

(2) ZMQ PPP client as the line controller

The ZMQ application which is used to test the traceability system is developed and the interface is as shown in Figure 5. Spotlight protocol signal, line command, and data request command can be sent with the application.



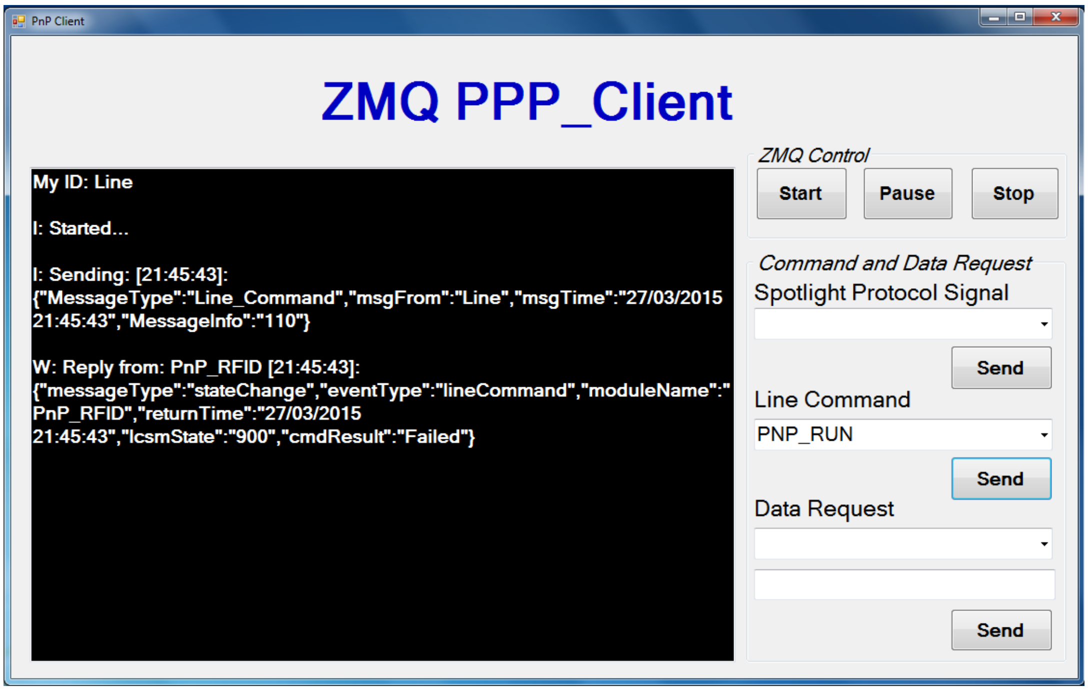
```
My ID: Line
```

Figure 5 Interface of ZMQ PPP Client Test Application

5.3 Test Results

Some screenshots of test results are given as follows.

(1) Linecommand and reply



Line Command: PNP_RUN

```
My ID: Line

I: Started...

I: Sending: [21:45:43]:
{"MessageType":"Line_Command","msgFrom":"Line","msgTime":"27/03/2015
21:45:43","MessageInfo":"110"}

W: Reply from: PnP_RFID [21:45:43]:
{"messageType":"stateChange","eventType":"lineCommand","moduleName":"
PnP_RFID","returnTime":"27/03/2015
21:45:43","lcsmState":"900","cmdResult":"Failed"}
```

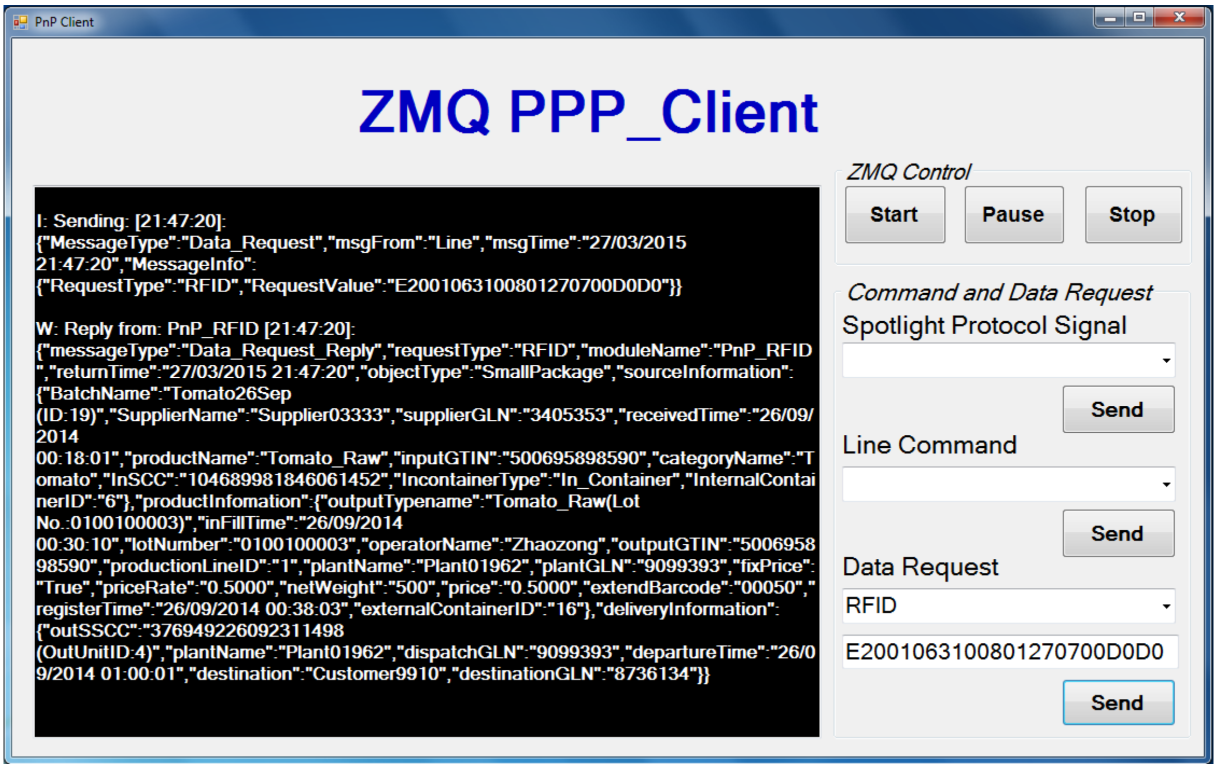
Figure 6 Line Command Request and Reply

The traceability system receives line command, execute the command, and then reply the results to the line. Figure 6 gives an example, the line sends “PNP_RUN” to the traceability system, and the traceability system replies “lcsmState”=”900” means the module is at a configure state and command is not successful.

(2) Data request and reply

The traceability system can receive data request from line controller and other modules in the line and reply the required information from the Database.

Figure 7 gives an example of data request with “RequestType”=”RFID”, “RequestValue”=”E2001063100801270700D0D0” and the reply of data in Database from the traceability system.



```
I: Sending: [21:47:20]:
{"MessageType":"Data_Request","msgFrom":"Line","msgTime":"27/03/2015
21:47:20","MessageInfo":
{"RequestType":"RFID","RequestValue":"E2001063100801270700D0D0"}}

W: Reply from: PnP_RFID [21:47:20]:
{"messageType":"Data_Request_Reply","requestType":"RFID","moduleName":"PnP_RFID
","returnTime":"27/03/2015 21:47:20","objectType":"SmallPackage","sourceInformation":
{"BatchName":"Tomato26Sep
(ID:19)","SupplierName":"Supplier03333","supplierGLN":"3405353","receivedTime":"26/09/
2014
00:18:01","productName":"Tomato_Raw","inputGTIN":"500695898590","categoryName":"T
omato","InSCC":"104689981846061452","IncontainerType":"In_Container","InternalContai
nerID":"6"},"productInfomation":{"outputTypename":"Tomato_Raw(Lot
No.:0100100003)","inFillTime":"26/09/2014
00:30:10","lotNumber":"0100100003","operatorName":"Zhaozong","outputGTIN":"5006958
98590","productionLineID":"1","plantName":"Plant01962","plantGLN":"9099393","fixPrice":
"True","priceRate":"0.5000","netWeight":"500","price":"0.5000","extendBarcode":"00050","
registerTime":"26/09/2014 00:38:03","externalContainerID":"16"},"deliveryInformation":
{"outSSCC":"376949226092311498
(OutUnitID:4)","plantName":"Plant01962","dispatchGLN":"9099393","departureTime":"26/0
9/2014 01:00:01","destination":"Customer9910","destinationGLN":"8736134"}}
```

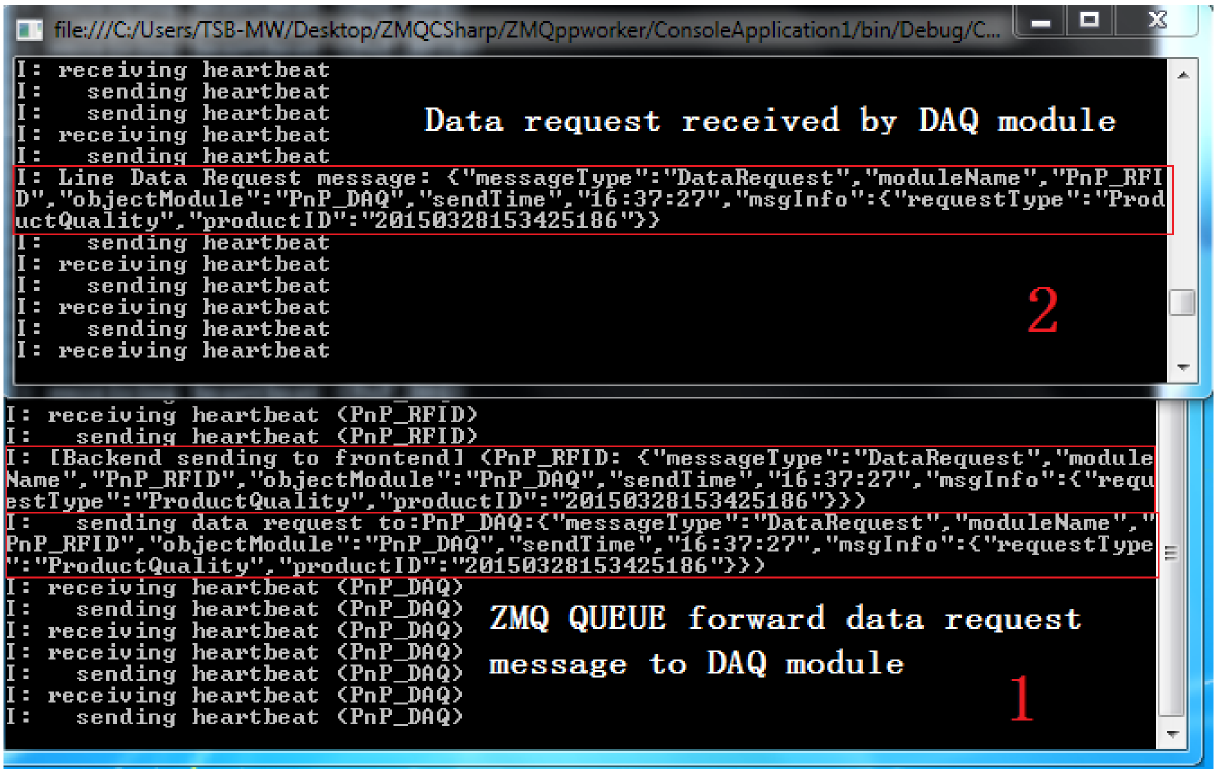
Data Request: RFID
E2001063100801270700D0D0

Figure 7 Data Request and Reply

(3) Request data from other modules

When data from other modules are useful to the traceability system, it is expected to request data from other modules with ZMQ message. The traceability system send data request message to the ROUTER and the ROUTER then forward the message to the object modules. When the result data is returned to the traceability system, it parses the message and presents the data to the user interface.

As shown in Figure 8, window 1 is the ROUTER forwardingthe message, and window 2 is the object module receiving the message.



```
file:///C:/Users/TSB-MW/Desktop/ZMQCSharp/ZMQppworker/ConsoleApplication1/bin/Debug/C...
I: receiving heartbeat
I:   sending heartbeat
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I: Line Data Request message: {"messageType":"DataRequest","moduleName","PnP_RFI
D","objectModule":"PnP_DAQ","sendTime","16:37:27","msgInfo":{"requestType":"Prod
uctQuality","productID":"20150328153425186"}}
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I: receiving heartbeat
```

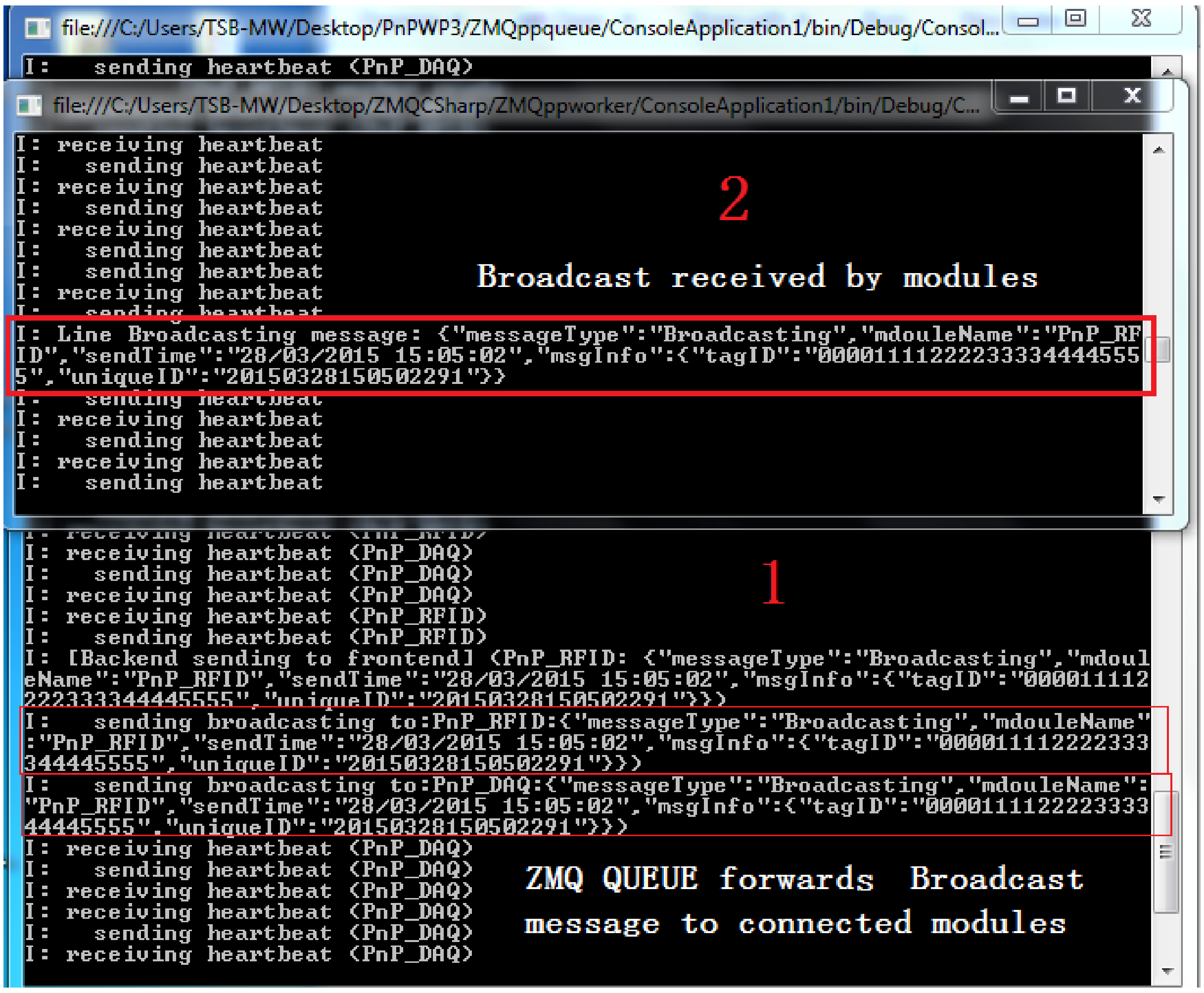
```
I: receiving heartbeat (PnP_RFID)
I:   sending heartbeat (PnP_RFID)
I: [Backend sending to frontend] (PnP_RFID: {"messageType":"DataRequest","module
Name","PnP_RFID","objectModule":"PnP_DAQ","sendTime","16:37:27","msgInfo":{"requ
estType":"ProductQuality","productID":"20150328153425186"}})
I:   sending data request to:PnP_DAQ:{"messageType":"DataRequest","moduleName","
PnP_RFID","objectModule":"PnP_DAQ","sendTime","16:37:27","msgInfo":{"requestType
":"ProductQuality","productID":"20150328153425186"}}
I: receiving heartbeat (PnP_DAQ)
I:   sending heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
I:   sending heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
I:   sending heartbeat (PnP_DAQ)
```

Figure 8 Request Data from other Modules in the Line

(4) Unique ID broadcasting

When new product is found by the RFID reader, the new object event is triggered and the unique ID can be generated or registered if it has been generated by WP2 module and it can be broadcast to the line.This may be useful for the synchronisation of the modules in the line.

Figure 9 gives an example of unique ID broadcasting to the ZMQ ROUTER. The messages can be forwarded to other modules in the line. In Figure 9, the bottom window is the ROUTER which forwards broadcasting message to all connected DEALERs(modules), and the top window is a DEALER window which has received the broadcasting message.


file:///C:/Users/TSB-MW/Desktop/PnPWP3/ZMQppqueue/ConsoleApplication1/bin/Debug/Consol...
file:///C:/Users/TSB-MW/Desktop/ZMQCSharp/ZMQppworker/ConsoleApplication1/bin/Debug/C...
2
Broadcast received by modules
1
ZMQ QUEUE forwards Broadcast message to connected modules

```
I:   sending heartbeat (PnP_DAQ)
```

```
I: receiving heartbeat
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I: Line Broadcasting message: {"messageType":"Broadcasting","mdouleName":"PnP_RF
ID","sendTime":"28/03/2015 15:05:02","msgInfo":{"tagID":"00001111222233334444555
5","uniqueID":"20150328150502291"}}
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
I: receiving heartbeat
I:   sending heartbeat
```

```
I: receiving heartbeat (PnP_RFID)
I: receiving heartbeat (PnP_DAQ)
I:   sending heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_RFID)
I:   sending heartbeat (PnP_RFID)
I: [Backend sending to frontend] (PnP_RFID: {"messageType":"Broadcasting","mdoul
eName":"PnP_RFID","sendTime":"28/03/2015 15:05:02","msgInfo":{"tagID":"000011112
2223333444455555","uniqueID":"20150328150502291"}})
I:   sending broadcasting to:PnP_RFID:{"messageType":"Broadcasting","mdouleName"
:"PnP_RFID","sendTime":"28/03/2015 15:05:02","msgInfo":{"tagID":"000011112222333
344445555","uniqueID":"20150328150502291"}})
I:   sending broadcasting to:PnP_DAQ:{"messageType":"Broadcasting","mdouleName":
"PnP_RFID","sendTime":"28/03/2015 15:05:02","msgInfo":{"tagID":"0000111122223333
44445555","uniqueID":"20150328150502291"}})
I: receiving heartbeat (PnP_DAQ)
I:   sending heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
I:   sending heartbeat (PnP_DAQ)
I: receiving heartbeat (PnP_DAQ)
```

Figure 9 Unique ID Broadcasting

6 Summary

This report summarises the integration work of the traceability system. It mainly focuses on two tasks: (1) Integration to the line, and (2) Data sharing with other modules. The tasks are based on the technologies of messaging mechanism ZMQ, data format JSON, and event and activity modelling method LCSM.

The functions for integration of the traceability system are implemented. However, further test and optimisation is dependent on progress of the other modules. The future work for the traceability system is suggested as follows:

(1) Fit the traceability system to the line with thefinalised line controller LCSM
(2) Adapt the data sharing application interfaces according to the interfaces other modules provide
(3) Optimise the integrated traceability system
(4) In line test and evaluationof the functions, speed, etc.

The future integration work is more dependent on the collaboration with other individual modules. The system needs to integrate to theintegralsoftware framework of the line. To complete the work listed above, the following issues need to be finalised:

(1) Software architecture
A software architecture incorporating each separate module, clarifying the interface, variables, and relationship between the modules is expected for the integration of the separate modules.
(2) Events and activities modelling
Events and activities of the line and each separate module need to be modelled, andapplication specific data interfaces and formatsneedto be explicitly defined.

The work mentioned in the above topics is very important for the integration of the traceability system and other modules as well.

Reference

[1] Hintjens, P. (2013). *ZeroMQ: Messaging for Many Applications*. Sebastopol, USA: O'Reilly Media, Inc.

[2] Hintjens, P. (2013). *ØMQ – The Guide,*http://zguide.zeromq.org/page:all, Accessed: 20 March 2015

[3] Ezzadeen, M. (2012). ZeroMQ via C#: Introduction, http://www.codeproject.com/Articles/488207/ZeroMQ-via-Csharp-Introduction, Accessed: 20 March 2015

[4] Ecma International (2013). Standard ECMA-404 – The JSON Data Interchange Format (1st Edition), http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, Accessed: 20 March 2015

[5]Newtonsoft (2015). JSON .NET – Popular high-performance JSON framework for .NET, http://www.newtonsoft.com/json, Accessed: 20 March 2015

[6] Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P. (2006) Modeling Software with Finite State Machine: A Practical Approach, USA: Auerbach Publications

[7] Philips, J., Bruyninckx, H. (2015). Pick-n-Pack WP2 Software Workshop Handouts, WP2 Workshop Pick-n-Pack meeting, Leuven, 3 March 2015

[8]Rob-Pick-n-Pack Group (2015), Pick-n-Pack Line Implementation, https://gitlab.mech.kuleuven.be/rob-picknpack/pnp-line/tree/master, Accessed: 20 March 2015