Published September 1, 2001. Distribution restricted to sponsors until December 1, 2001.

AUTO-ID CENTRE

WHITE PAPER

Integration of Auto-ID Tagging System with Holonic Manufacturing Systems

Chien Yaw, WONG

CAMBRIDGE UNIVERSITY AUTO-ID CENTRE INSTITUTE FOR MANUFACTURING, UNIVERSITY OF CAMBRIDGE, MILL LANE, CAMBRIDGE, CB2 1RX, UK

ABSTRACT

This report is the result of seven weeks of work by Chien Yaw WONG from Institute for Manufacturing (IfM), University of Cambridge. Out of the seven weeks of project, a week was spent at Massachusetts Institute of Technology (MIT) Auto-ID Centre, Boston to engage the scientists about the issues around this project. This report contains the project information regarding the integration of Auto-ID Tagging System with an emerging approach to distributed control referred to as Holonic Manufacturing System (HMS).

This project involves integrating an HMS developed at IFM, University of Cambridge and Auto-ID Tagging System initially developed at MIT. Both the system were successfully integrated but full integration was not possible due to on-going development work at MIT Auto-ID Centre and lack of development time.

The main deliverable of this project is a software interface referred to as XCHANGE, which enables communication between the HMS and the Auto-ID Tagging System. The XCHANGE is capable of on-the-fly extraction of product information and converting them into the format used by HMS. A CRMInterface (CustomerRelationshipManagementInterface) was also developed to provide a graphical user interface for customers to customize their production order, check and alter production requirements, and automatically communicate with the XCHANGE for product synchronization.

The benefit of this integration was analysed and a video demonstration was developed to illustrate the key benefits.

WHITE PAPER

Integration of Auto-ID Tagging System with Holonic Manufacturing Systems

Biography

by Chien Yaw, WONG
PhD Candidate & Research Assistant

Chien Yaw is reading his Master of Manufacturing Engineering at Cambridge University. He has been involved with several automation control project including Holonic Manufacturing System (HMS) and PLC programming. With the close collaboration between MIT and Cambridge University, he aims to integrate Auto-ID technology with HMS to enable a paradigm shift in the way things are manufactured in the future. He is also heavily involved in evaluating and analysing the benefit of adopting e-Manufacturing. In his free time, Chien Yaw enjoys playing the piano and listening to funky jazz.

WHITE PAPER

Integration of Auto-ID Tagging System with Holonic Manufacturing Systems

Contents

WHITE PAPER

Integration of Auto-ID Tagging System with Holonic Manufacturing Systems

Contents

This report is the result of seven weeks of work by Chien Yaw WONG from Institute for Manufacturing (IfM), University of Cambridge. Out of the seven weeks of project, a week was spent at Massachusetts Institute of Technology (MIT) Auto-ID Centre, Boston to engage the scientists about the issues around this project. This report contains the project information regarding the integration of Auto-ID Tagging System with an emerging approach to distributed control referred to as Holonic Manufacturing System (HMS).

This project involves integrating an HMS developed at IFM, University of Cambridge and Auto-ID Tagging System initially developed at MIT. Both the system were successfully integrated but full integration was not possible due to on-going development work at MIT Auto-ID Centre and lack of development time.

The main deliverable of this project is a software interface referred to as XCHANGE, which enables communication between the HMS and the Auto-ID Tagging System. The XCHANGE is capable of on-the-fly extraction of product information and converting them into the format used by HMS. A CRMInterface (CustomerRelationshipManagementInterface) was also developed to provide a graphical user interface for customers to customize their order and send the requirements to the XCHANGE for synchronization.

The benefit of this integration was analysed and a video demonstration was developed to illustrate the key benefits.

1. AIMS

This project aims to integrate Auto-ID Tagging System with Holonic Manufacturing System.

The basic deliverables for this project are to provide:
1. A fully integrated system of architecture consisting of MIT Auto-ID Tagging System and Holonic Manufacturing System.
2. A full report itemising every aspect of the integration and the rationale behind it. This report will largely be oriented as a manual for future development.

It was hoped that during the duration of this project, it would be possible to develop a fully presentable demonstration video of the production cell. The demonstration video is now available from Institute for Manufacturing.

The project timeline is as follow:
First Week:
– Scoping Week

Second Week:
– Review Auto-ID Documentation
– Review Holonic Manufacturing Documentation
– Identification of integration issues for discussion

Third Week:
– MIT Auto-ID Center Visit
– Identification of full Auto-ID functionality
– Discussion of integration issues

Fourth Week:
– Identification of Critical Success Factors (CSFs) for integration
– TimeBoxing of functionality and reiterations
– Generate possible demonstration scenarios

Fifth, Sixth & Seventh Week:
– Integration of both systems using DSDM approach
– Develop Cell Demonstration
– Cell Demonstration

2. OVERVIEW OF AUTO-ID TAGGING SYSTEM

The Auto-ID Center intiated at the Massachusetts Institute of Technology, and now also with a sister centre of University of Cambridge, is a new industry sponsored lab charged with researching and developing automated identification technologies and applications. The Auto-ID Center envisions a world in which all electronic devices are networked and every object, whether it is physical or electronic, is electronically tagged with information pertinent to that object.

This section will outline the infrastructure needed to enable the vision. At the time this report was written, much of the standards and infrastructure are still in development. Hence, care must be taken when using information from this section.

2.1. The Electromagnetic Identification Tag

The Electromagnetic Identification (EMID) tag is a memory device with circuitry for wireless contactless communication with an external tag reader. It is capable of storing and transmitting information such as the EPC code (Refer Section 2.2.1).

[1] The Network Physical World, MIT Auto-ID WH-001.

The Radio Frequency Identification (RFID) is a subset of EMID as EMID includes classes of device that use sub-RF frequencies to transmit information. The tag cost must be significantly low for it to be commercially viable (the preliminary cost target for MIT Auto-ID Center is 10 cents[1]).

2.2. Standards

[2] http://auto-id.mit.edu

2.2.1. Electronic Product Code

The Electronic Product Code (EPC) is a numbering scheme[2] that can provide unique identification for physical or virtual entities. The code serves as a reference for networked information; no information about the entity is stored within the code. The code will direct any queries about the entity to where it should go to find information on the Internet.

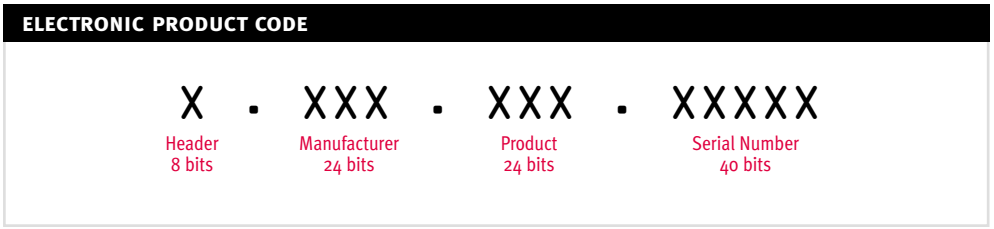
[3] (Hyperlink "http://auto-id.mit.edu") – Fact Sheets

Scientists at MIT Auto-ID Center have proposed a 96-bit scheme, including an 8-bit header and three data partitions[3], as shown in Exhibit 1.

2.2.2. Product Markup Language

The Product Markup Language (PML) is a standard language for describing objects. It is based on eXtensible Markup Language (XML).

Exhibit 1: Electronic Product Code



PML will describe physical objects, their configuration and state. This is an extension from XML, which possesses information about the type of data, and HTML, which possesses information about **how** information should be displayed.

PML will ultimately contain:

- Static Data – such as product history information
- Instructions – such as production processes recipe
- Dynamic Data – such as most up-to-date information about location of the product
- Software – such as intelligent agents that dictates behaviour of the product

2.2.3. Object Naming Service

Object Naming Service (ONS)[4] tells computer systems where to locate information on the Internet about any object that carries an EPC. ONS is similar to the existing Internet's Domain Name System (DNS), which allows Internet routing computers to identify where the pages associated with a particular Web site are stored.

[4] Developed at Massachusetts Institute of Technology by Dr. David Brock, Professor Sanjay Sarma and Joseph Foley

ONS will be need to carry out much more tasks more quickly than the present DNS due to the sheer number of objects that could potentially carry an EPC code in the future (the number is to the order of trillions).

The EPC, PML and ONS completes the fundamental infrastructures that are needed to **link information with physical objects**. The EPC identifies the product, the PML describes the product and the ONS links them together.

2.3. Architecture

2.3.1. PML Servers

Special servers, called PML Server will store the PML files. At the time this report was written, the exact specifications of PML Servers are not finalised.

It is likely that XQL will be the standard query language for the PML files that are stored in PML Server because it is developed especially for XML, a standard that the PML is based on. XQL will only extract the relevant information needed from a possibly huge PML file for data transfer. This reduces the size of data transfer and increases information processing efficiency at the client-side.

2.3.2. The Savant

The **Savant**[5] is the central "active" component that sends synthesized data to the outside world without being polled for that data. The **Idiot-Savant** is a passive device that can make limited inferences on data input via XQL queries/requests.

[5] J.L.Waldrop, 2001 (MIT)

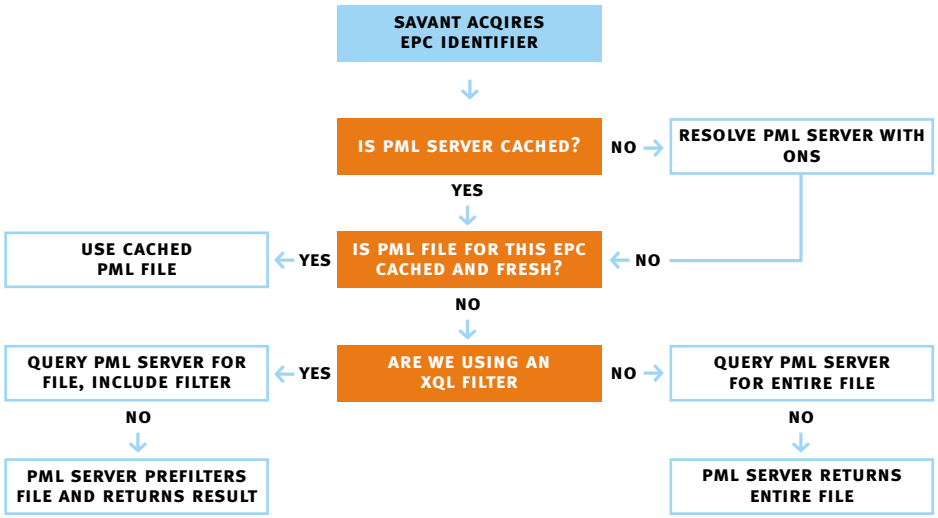
The Savant is an intelligent agent that manages the information flow internally, and to the outside world. The Savant that retrieves an EPC identifier will convert EPC ID to PML server (either local or global) using

ONS. The Savant queries the PML server for PML file and will listen to incoming requested information from the server. Some Savants will have the intelligence to update the PML files of certain EPCs using XQL-enabled PML Server.

The path of a full ONS/PML query is as shown in Exhibit 2.

Exhibit 2: The Savant queries the PML server file.

THE SAVANT EPC IDENTIFIER



2.3.3. The Architecture Design

The architecture design that utilizes the standards of Auto-ID tagging system is as shown in Exhibit 3. The network system is divided into **Global** and **Local** network. The local network resides within a confined area, such as a manufacturing plant. The **Global** network consists of the PML Server that can be accessed globally through the ONS server using TCP/IP [6].

The respective functions for Savant and DBMS [7] can be found in Exhibit 3.

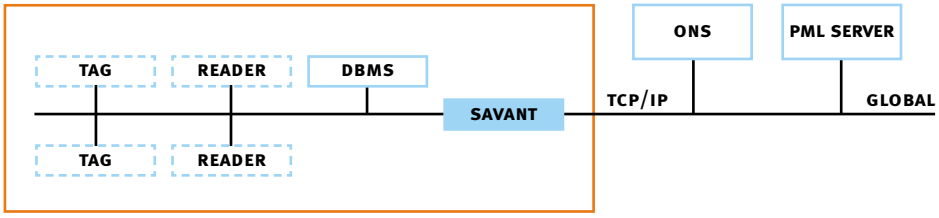
[6] TCP/IP, as a set of communications protocols, is based on layers. Unlike SNA or OSI, which distinguish seven layers of communication, there are only four layers in the TCP/IP model. They enable heterogeneous systems to communicate by performing network-related processing such as message routing, network control, error detection, and correction.

Exhibit 3: The architecture design that utilizes the standards of Auto-ID tagging system.
Savant:
– DBMS Management
– PML Server(XQL-enabled)
– Control routines
– Drivers for local system

DBMS:
– Caches PML files locally
– Caches tag reader location
– Stores timestamp

[7] Database Management Systems.

THE ARCHITECTURE DESIGN



3. OVERVIEW OF HOLONIC MANUFACTURING SYSTEM

[8] White Papers from Institute for Manufacturing, University of Cambridge.

The concept of holonic manufacturing involves the operation of a manufacturing function based on the cooperation of autonomous, functionally complete entities with diverse, often conflicting goals. This section will outline the definition and philosophy of the system, and evaluate its characteristics with the conventional control system. More details are included in a forthcoming Auto-ID white paper on this subject. [8]

3.1. Definition of Holonic Manufacturing System

[9] http://hms.ifw.uni-hannover.de/public/Feasibil/holo2.htm

Holonic Manufacturing Systems (HMS) is defined as **a holarchy which integrates the entire range of manufacturing activities from order booking through design, production and marketing to realise the agile manufacturing enterprise.** [9]

The holarchy is defined as **a system of holons which can co-operate to achieve a goal or objective. The holarchy defines the basic rules for co-operation of the holons and thereby limits their autonomy.**

In essence, Holonic Manufacturing System consists of a modular architecture to support reconfigurability of a process as well as distributed algorithms for cooperative behaviour and decision-making execution. In this report, more emphasis will be placed on the control architecture. Algorithms for planning, scheduling and shop floor control will not be covered.

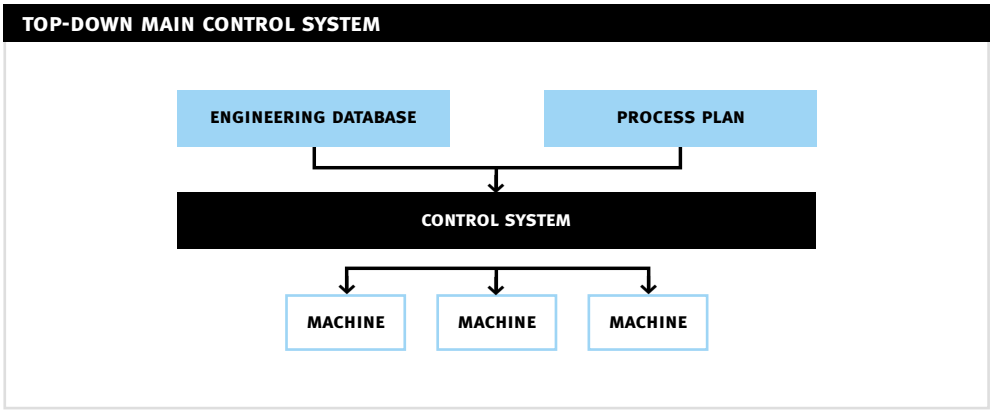
3.2. The Holonic Control Architecture

[10] Jin-Lung CHIRN, 2001, IfM

The particular holonic control architecture developed at IfM [10] consists of **resource holons** and **product holons** that interact giving distributed control. **Resource holons** have both a physical element (equipment, devices etc) and a control element (machine control, decision making, communication). **Product holons** also have a physical element (raw material, pallet/fixture) and a control element (routing/process control, decision making, communication). The product holon contains a "recipe" of the resources and operations that are required to create the finished product.

The Holonic Control Architecture is different from the conventional architecture because each product entity (such as products to be manufactured-) and resource entity (such as machine cells) has the capabilities for autonomous decision-making. As shown in Exhibit 4, the conventional architecture "resources" consist of a top-down main control system that manages the engineering database and process plan, and passes instructions down to the machines level. No decisions are made at the machine level.

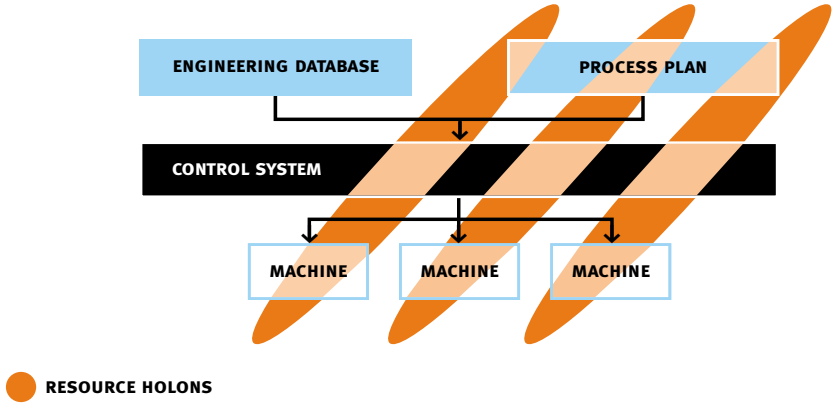
Exhibit 4



However, as shown in Exhibit 5, the **resource holons** comprises of the process plan, control systems as well as the machine. The inclusion of the control system and process plan into the machine level as a single autonomous entity enables decision-making in the **resource holons**. These decision making capabilities will enable "cooperation" between other **resource holons** and **product holons**.

Exhibit 5

RESOURCE HOLONS

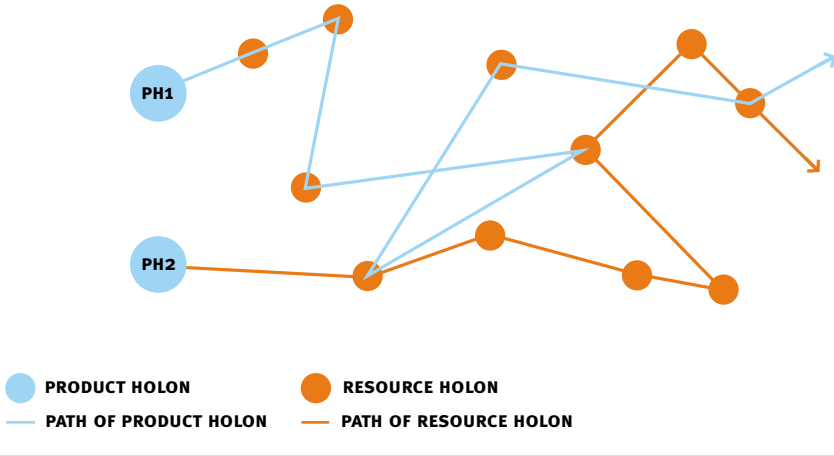


The **cooperation** between different resource holons and product holons is often quoted in the work of Koestler in social organisations, living organisms and human beings. Koestler defines **cooperation** as **a process whereby a set of entities develops mutually acceptable plans and executes these plans**.

This cooperation can be shown in Exhibit 6. In the diagram, there are numerous resource holons that are in constant negotiations with the product holons as needed. The product holons have a set of recipes, which dictates their path through **only certain or all** the resource holons, and make their way through the resource holons by cooperation with other holons. These recipes are analogous to human DNAs, as they dictate the physical and mental being of humans **individually**.

Exhibit 6

RESOURCE HOLONS THAT ARE IN CONSTANT NEGOTIATION WITH PRODUCT HOLONS



3.3. Conventional Architecture versus Holonic Architecture

[11] Adapted from C.Y.WONG, C.TUOHY 2001 (MET)

The **Evaluation Matrix**, as shown in Exhibit 7 shows the difference between the conventional architecture and holonic architecture.[11]

Exhibit 7

EVALUATION MATRIX	CONVENTIONAL ARCHITECTURE	HOLONIC ARCHITECTURE
STRUCTURE	Top-down, Detrministic, Centralised Hierarchical	Distributed Control with decision-making capability
FLEXIBILITY	Rigid and static architecture and capabilities assigned to specific layers	Flexible, programmable and dynamic architecture
PRODUCTION TIME	Most efficient for highly complex algorithm	Most efficient for highly complex algorithm
MACHINE UTILISATION	Potential for high utilisation and machine balancing	Random utilisation depending on product mix
ERROR RECOVERY	Low resilience to failures due to complex coding	High resilience to failure through dynamic reconfiguration ration and task re-negotiation
INITIAL SET-UP TIME	Long and tedious due to low level code design	Short due to “Plug-n-Play”
CUSTOMIZATION	Difficult for individual product	Potential for 100% customization

As a summary, the holonic architecture is potentially more robust and resilient compared to the conventional architecture. As it is designed to support autonomous, cooperative decision making, each product can be treated individually, and hence a potential for a mass customisation.

HMS can execute numerous scenarios, even scenarios that were never predicted before. The conventional architecture could support these scenarios, but only if these scenarios are pre-determined and potentially complex algorithms are put in place.

4. THE INTEGRATION BENEFIT

The feasibility of integrating Auto-ID tagging system with the HMS will be the main deliverable in the video demonstration. Not only will the demonstration show that the integration is possible, it will take a step further to actually illustrate the benefit of such integration. Exhibit 8 shows the advantages of both systems and the integrated system.

Exhibit 8

DISTRIBUTED MANUFACTURING	AUTO-ID TAGGING
– Distributed (Holonic) Control – Autonomous – Individual Product Customisation – 'No Schedule' Possible – Plug & Play Modular System – Easily Reconfigurable – Various Optimization Capabilities – Highly Resilient To Disruptions	– Product Information Availability – Unique Product Information – Individual Product Identification – 'All Data' Possible – Dynamic information of product – Customizable information – Connectivity to other platforms – Possible hidden tags

↓ ↓

INTEGRATED SYSTEM

– Uniquely identify products for customization in assembly
– dynamic product product mix based on customer requirements
– Further WIP customisation during assembly
– Intelligent Production System
– Accurate Diagnostic and Prognostic Capability

5. PHYSICAL SYSTEMS INTEGRATION

The physical integration, to be demonstrated here, of Holonic Manufacturing System with Auto-ID Tagging system involves one of the current production cells within Institute for Manufacturing.

This section aims to provide as much information as possible about the product and the production cell used in this integration. At the end of this section, the focus will be on the video demonstration and what it intends to illustrate.

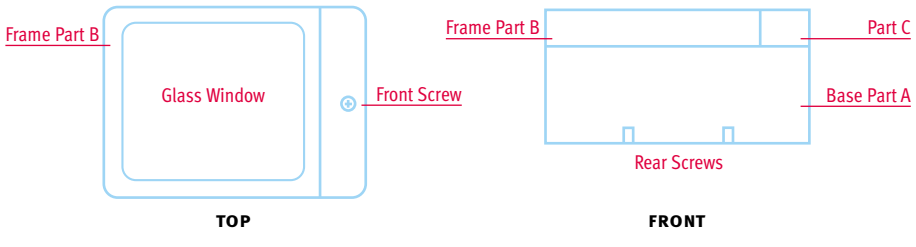
5.1. The Product

5.1.1. The Product: Meter Box

The Meter Box consists of three parts: Part A, Part B and Part C. Part A forms the base for Part B and C to reside on and it contains a tag for identification. All the parts comes in three different colour: red, yellow and black. For this demonstration, only black parts will be used.

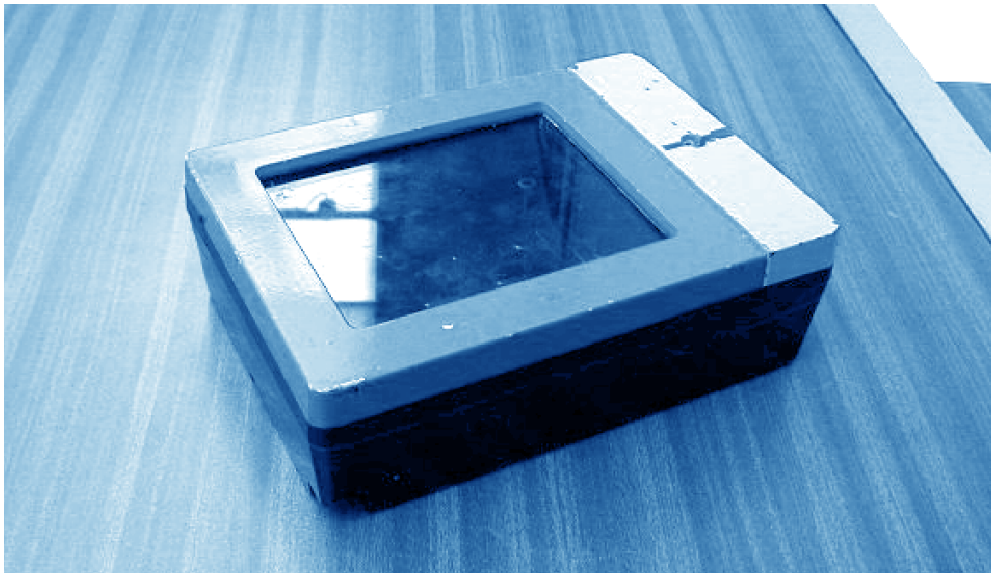
Exhibit 9

THE METER BOX



The sequence of assembly starts with Part C fastened to Part A using a single screw to form Part AC. The combination of Part B into Part AC using two screws fastened at the rear will form Part ABC, as shown in Exhibit 9 and 10. For this reason, it is necessary to flip Product AC during assembly using the flipper unit.

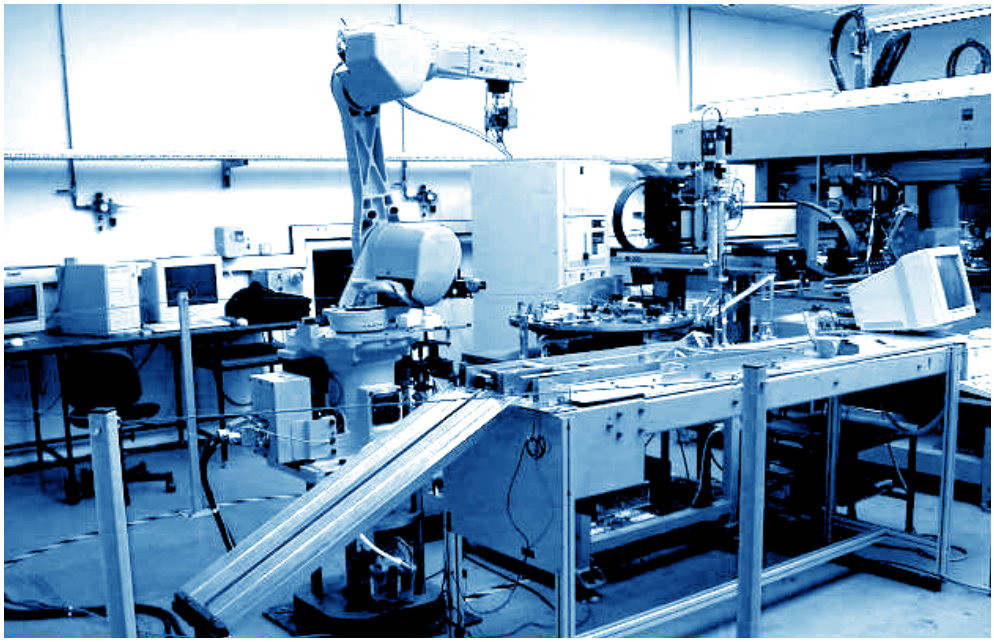
Exhibit 10



5.2. The Meter Box Assembly Cell

This assembly cell includes five physical machines and two main controllers. The following section describes the operation of each of these machines. More detailed information can be found in the Appendix.

Exhibit 11



5.2.1. Tagging System

The tagging system is used to identify Part A at the end of the conveyor. The tag reader is the Omron V700-HMD11 and has read and write capabilities. It is connected physically to the PLC.

Exhibit 12

THE TAGGING SYSTEM

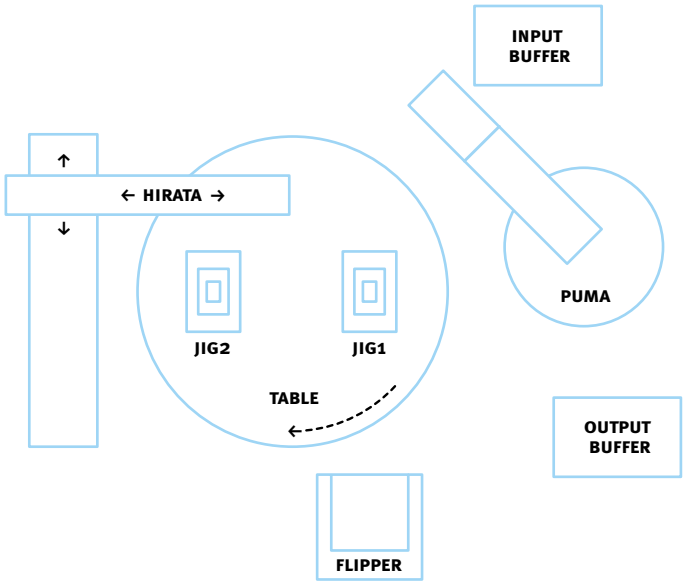
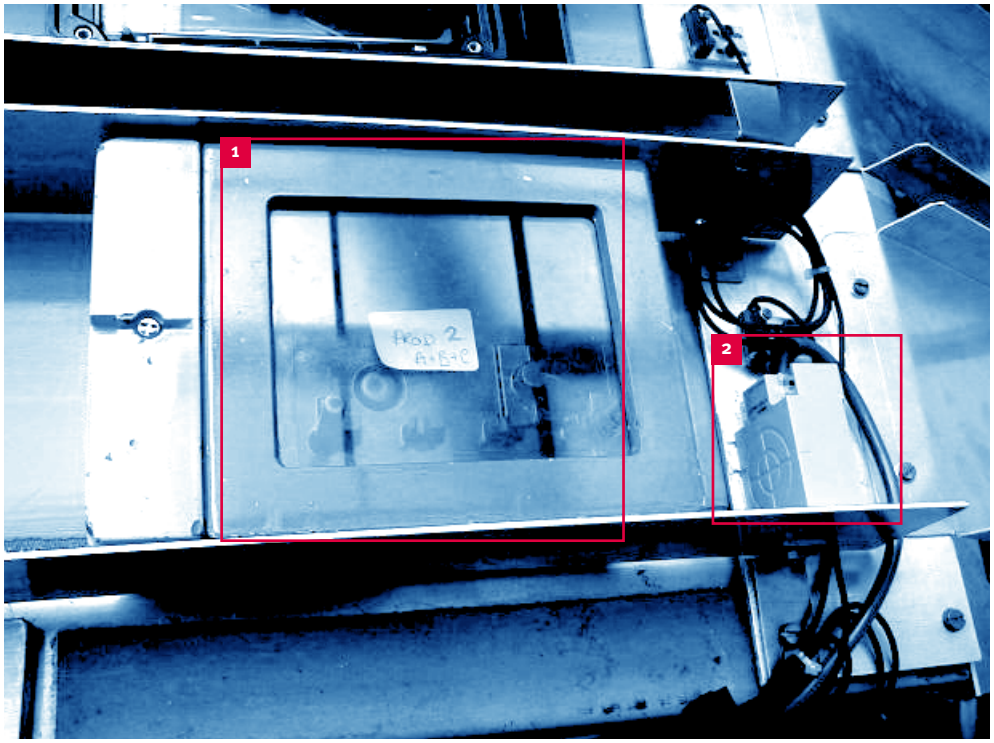


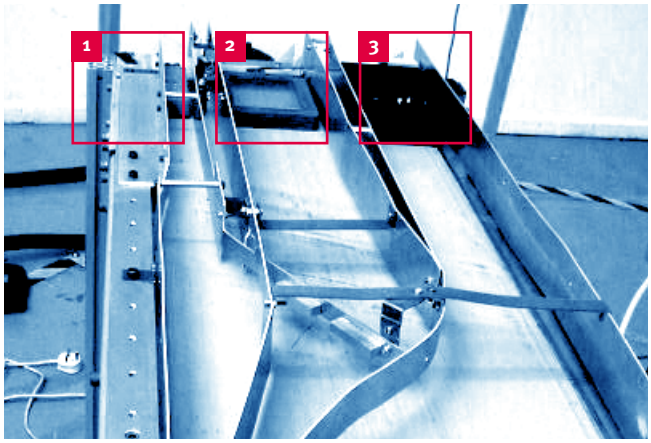
Exhibit 13:
1 Tag within Product ABC
2 Tag Reader



5.2.2. Conveyor

The purpose of this conveyor is to feed the Fanuc with Part A, B and C located precisely at the end of the conveyor using a physical separator (Exhibit 14). The conveyor belt can run on forward and reverse mode as needed. At the end of the conveyor, a tag reader is attached to read the tag on Part A. This tag reader is controlled by the computer through the PLC.

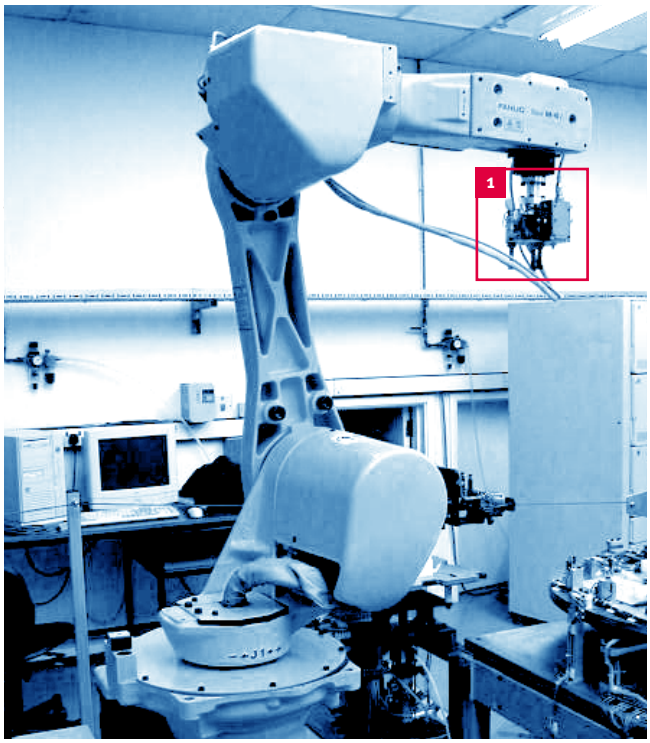
Exhibit 14:
1 Part C
2 Part B
3 Part A



5.2.3. Fanuc Robot

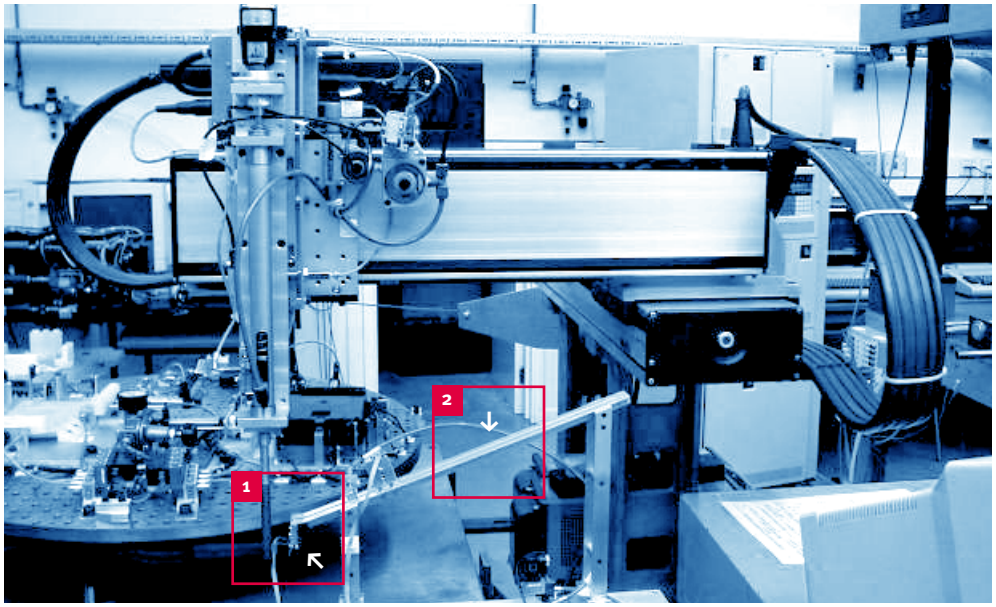
The Fanuc system consists of a R-J3 main controller system with a Robot M-6i. The robot is a six-axis, electric servo-driven robot designed for a variety of manufacturing and system processes. The Fanuc is responsible to move all the parts in the assembly cell. R-J3 has a teaching programme which is accessible by the PC.

Exhibit 15:
1 Pneumatic Picker



The Hirata MB-240 fastens screws onto the back plane of Part AB to attach Part C. Screws are fed via a feeding system (Exhibit 16). This fastening operation is executed on the jig table.

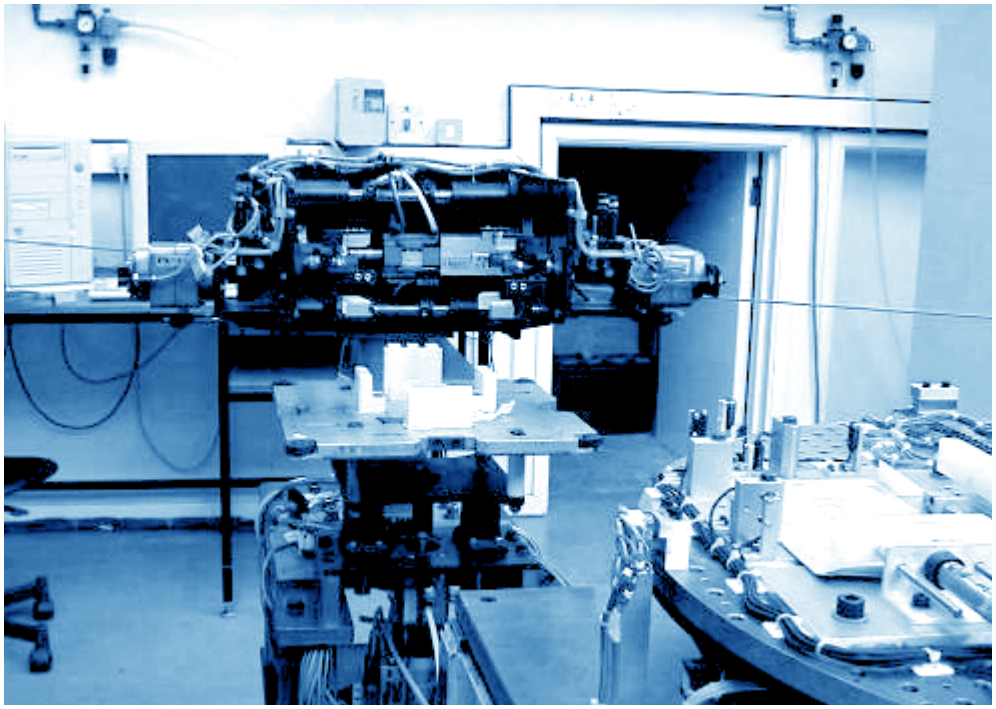
Exhibit 16:
1 Pneumatic Picker
2 Screw Feeder



5.2.5. Flipper Unit

If Part C needs to be assembled to Part AB, four screws need to be fastened by the Hirata at the back plane of Part AB (Exhibit 17). Hence, Part AB needs to be flipped in order for the Fanuc to position the flipped Part AB on Part C, previously positioned by the Fanuc on the Table.

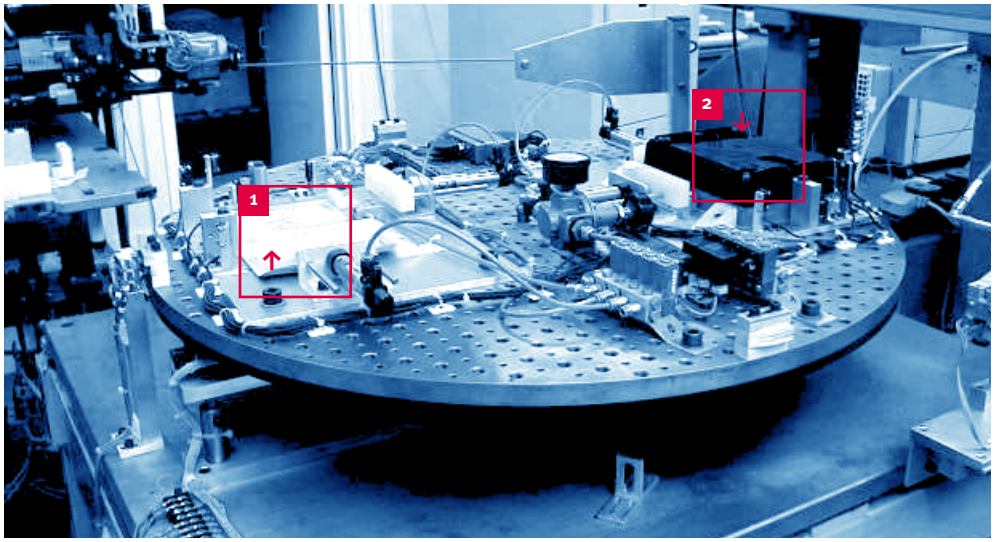
Exhibit 17



5.2.6. Jig Table

This jig table is used to position the parts precisely for further assembly, either by the addition of another part by the Fanuc or by screws fastened by the Hirata. The table has two jigs, with pneumatic pistons, to realign the parts into location. The sensors are there to indicate whether parts are in the jigs and it will flag the PLC appropriately.

Exhibit 18:
1 Jig 1
2 Jig 2 with Part ABC



5.2.7. Controller

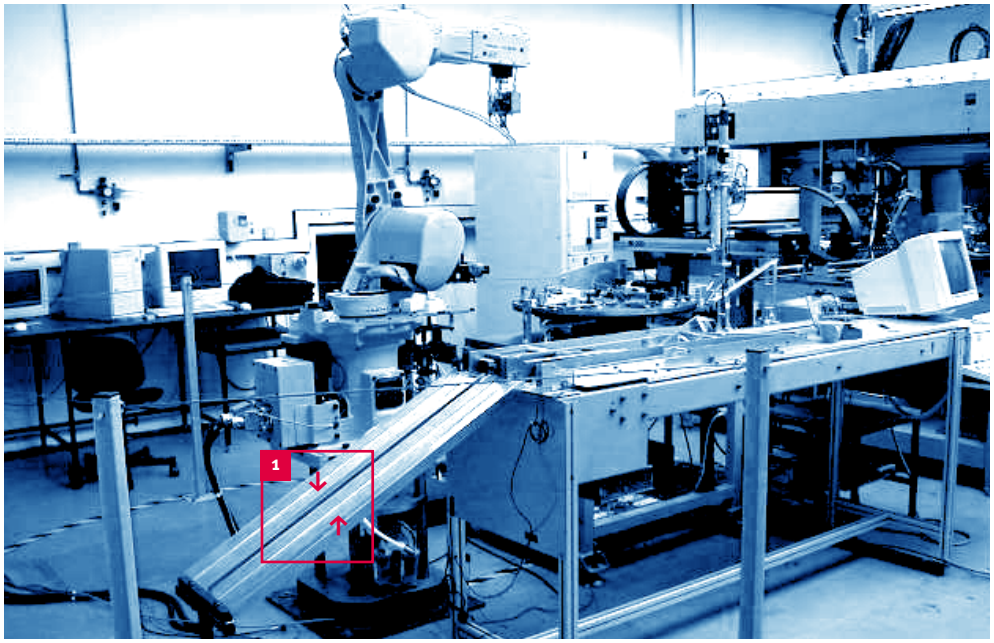
There are two main control infrastructures in this cell: Omron PLC and PC controller. Omron PLC is physically connected with all the machine cells except for the conveyor belt. It allocates addresses to each of these machines and does low-level machine operation. The PC controller consists of two computers connected via the Ethernet network and with the PLC via RS-232 link. The PC controller uses high-level programming language to control the behaviour of the machine as an integrated cell.

Exhibit 19

5.2.8. Output Buffer

There are two output buffers: **Output A** and **Output B**. Finished products will be dropped into the output buffer according to specified requirements.

Exhibit 20:
1 Output A (right) & Output B (left)



5.3. The Video Demonstration

The integration of both systems will not just enable extrapolated linear benefits but will also provide a big leap towards fully customizable customer-oriented products in a highly reconfigurable, resilient modular production cells. Hence, ideally, the video demonstration[12] ideally needs to prove a capability that could not have existed in the conventional production system.

[12] Video Demonstration available from Manufacturing Engineering Department, Cambridge University

The benefit demonstration scenarios were selected as follow:
1. Similar Products, Unique Identification
2. Intelligent Product Status Update
3. Further Customer Customization Execution

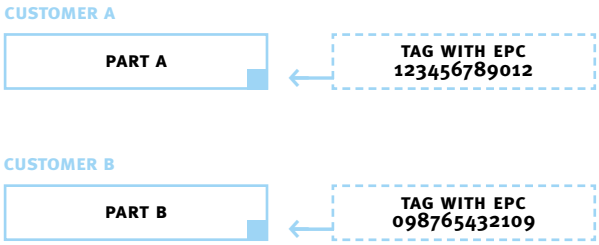
Similar Products, Unique Identification

This first demonstration scenario showed the ability to treat **physically identical** products uniquely. Physically identical means that both of the products demonstrated are visually indistinguishable. The only difference is with the information stored within the products: **both products have different EPC numbers stored within the tag**. Hence, any physical sensors (colour, weights, features etc), bar code readers or image recognition will not be able to differentiate these products.

Although they are physically identical, these products could be **different internally** with different embedded chips or parts, or with different information within the same embedded chip. These product could have different customer specifications and hence possesses different production requirements. In the demonstration, Customer A and Customer B placed their order for Product AC. Both of the customers are assigned with unique EPC numbers that are embedded in the tags of Part A respectively, as shown in Exhibit 21.

Exhibit 21

UNIQUE EPC NUMBER THAT ARE EMBEDDED IN THE TAGS



Both of the parts went through the production cell with Part C added on to it to form Product AC. At that point, both Product AC looked identical but they were inherently because Product AC of Customer A (EPC = 123456789012) was dropped into Output A once production completed and Product AC of Customer B (EPC = 098765432109) was dropped into Output B.

Note that in the video demonstration, both products (Customer A and Customer B) are manufactured at the same time in the machine cell. However, having identified the identity of the product at the beginning of the production, the product holons will find their way through the production system until the correct output buffer.

Intelligent Product Status Update.
This dictates the ability to keep track of products in manufacturing processes. The product status updates will provide information about the current characteristics of the product and will use this dynamic information for intelligent purposes.

In the demonstration, once Product AC was manufactured, its production status was updated in the PML files to indicate that the product was manufactured. Hence, the attempts in bringing Product AC back to the conveyor was harmless, as the product "recognized its state" and was taken back to the output buffer (instead of trying to manufacture Product AC **out of Product AC**). This scenario might, for example, replicate repeat testing or processing in a flexible routing environment.

Further Customer Customization Execution.
As more and more emphasis is placed upon Customer Relationship Management (CRM), the ability to customize the product according to the wish of customers is likely to be vital for long-term survival of the firm. To take that step further, customers will be have the production information transparency and customize each and every product that they own individually, while it is still in the production process. This is a tremendous shift from the traditional product information response of "Not yet available" to "Your product is in Pallet 23, South Wing, Level 4. Do you wish to change the requirements of this product?"

In the demonstration, both Product AC from Customer A and B are in Output A and B respectively. As shown in Scenario 2, both of the products were taken back from the conveyor straight into the correct Outputs. However, when Customer A changes the requirement from "Product AC" to "Product ABC", the product with EPC=123456789012 (Customer A) was brought back into the production cell for further manufacturing process (Part B is fastened to Product AC). However, as the requirement for Product B was not changed, it was moved back into the Output B, similar to Scenario 2.

6. SOFTWARE SYSTEMS INTEGRATION

The software integration involves new software developments and integration with the existing Holonic Manufacturing System. This section will outline the approach and the software architecture design.

6.1. The Implementation Approach

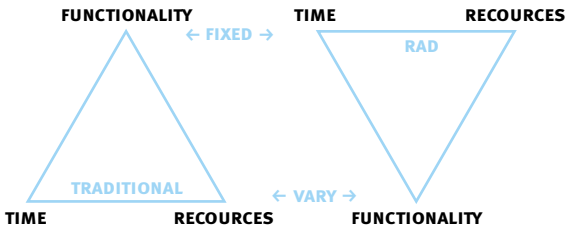
6.1.1. Dynamic Systems Development Method

[13] http://www.dsdm.org

The Dynamic Systems Development Method (DSDM)[13] is a framework of control for the development of IT systems to tight timescales. The lifecycle that DSDM uses is iterative and incremental. As shown in Exhibit 22, traditional approaches fix requirements (and deliver software which satisfies all of them) while allowing time and resources to vary during development. In DSDM, the exact opposite is true, time is fixed for the life of a project, resources are fixed as far as possible. This means that the requirements that will be satisfied are allowed to change. Hence an important product of the business study, is a clear prioritisation of the high-level functional and non-functional requirements. These functional requirements are prioritised in TimeBoxes using MOSCOW (Must, Ought, Should, Could and Wouldn't). DSDM projects guarantee to satisfy at least the minimum usable subset of requirements because it identifies the critical success factors (functional requirements of "Must" from MOSCOW prioritisation) very early in the project and act accordingly.

Exhibit 22

DYNAMIC SYSTEM DEVELOPMENT METHOD



This implementation approach allows this integration to be categorised to three different level of sophistication: **Basic, Advanced, Sophisticated**. In order to achieve higher level of sophistication, certain functionalities that are normally developed in a proper development environment, will be abandoned. For example, due to lack of time, functionalities that are not a **MUST** prioritisation in **Basic** could be discarded in order for the next **Advanced** development to continue. This is analogous to a three-course meals whereby the starter is not fully dined in order for the person to enjoy the main course, so on and so forth.

6.1.2. Programming Language

The choice of programming language is dependent on a few criteria. The first and foremost is time con straint. The proposed integration period is three weeks. This includes modification to the cell (adding the output buffers), troubleshooting existing production cell problems and so on. Therefore, the maximum time spent for coding should be less than one and a half week. The language must be easy to learn and easy to code. The language must support and enable XML programming, Client/Server communications and other necessary functions.

As such, Delphi has been selected as the preferred choice. Delphi has set a new standard in high-performance rapid application development. Delphi is open and extensible and does not suffer from a limited feature set. Delphi ships with a very robust set of components, each of which can be customized in appearance and in the way it handles events. The Pascal language is also robust, giving access to a stable run-time library and complete access to the Windows API. The look and feel is quite similar to Visual Basic, except that Delphi uses Pascal as the core language. Although Visual Basic and Visual C++ are the norms, they are difficult to learn in such a short period of time to provide the required functionalities.

Microsoft XML 2.0 component could be inserted into Visual Delphi and used to parse XML documents. Since Microsoft XML 2.0 is not language dependent, the Delphi code could be adapted with minimal changes to suit other programming languages, rendering the code considerably scalable.

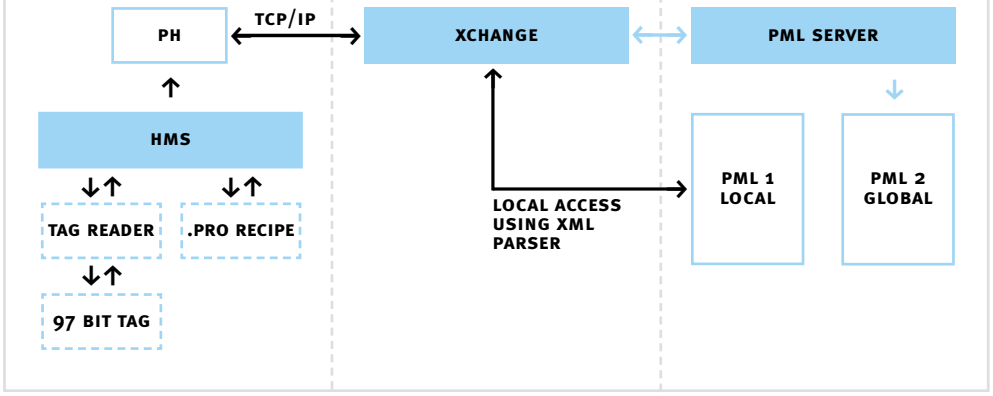
6.2. The Product Holon Integration with the XCHANGE

The aim of this integration is to provide a quick, yet extensible interconnectivity platform between the tagging system and Holonic Manufacturing System (HMS). This integration will allow the HMS to exploit the benefits of Auto-ID Tagging system in the existing manufacturing cell. Due to the complex legacy architecture of the HMS and short integration period, it is not practical to change the HMS architectural database according to the standard. The proposed solution, as shown in Exhibit 23, is a middleware [14] that manages communication between both systems.

[14] Software that mediates between an application program and a network. It manages the interaction between disparate applications across the heterogeneous computing platforms. The Object Request Broker (ORB), software that manages communication between objects, is an example of a middleware program.

Exhibit 23

THE PRODUCT HOLON INTEGRATION WITH XCHANGE



The aims of the XCHANGE are to:

- Establish communication and data transfer with HMS via Product Holon (PH)
- Establish communication and data transfer with PML Server, or directly access PML files locally
- Read and write information into PML files as requested by HMS
- Convert PML file into .PRO recipe files
- Extract information from PML file

Establishing communication and data transfer with HMS via Product Holon (PH).
HMS and XCHANGE will communicate using TCP/IP as the basic four layers of communication[15].

[15] http://www1.ibm.com/servers/eserver/iseries/beyondtech/tcp_ip_services.htm

Other layers of communication protocols are as defined in Section 6.4. Once all seven layers of communications are defined, HMS will be able to send and retrieve data from XCHANGE.

Establishing communication and data transfer with PML Server, or directly access PML files locally.
Once communication with HMS is established, XCHANGE needs to have a way of accessing the PML files. There are two ways of achieving this. The first and easiest way is to store the PML files locally and local is defined as where the XCHANGE programme is located. Alternatively, XCHANGE could communicate with PML server via ONS, but at the time this report was written, such an infrastructure was yet to available.

Reading and Writing information into PML files as requested by HMS.
The ability to do this should be the ultimate objective of XCHANGE. Using pre-defined communication protocols, HMS can request to write and read information from PML files through XCHANGE. The ability to **read** will enable production data of a particular product to be stored in the PML. The ability to **write** will, however, allow a big leap towards much more dynamic PML files, capable of storing near real-time information about the product.

In this implementation, HMS will be able to read production data from PML, and update the production status accordingly, whether it is not in production, completed Part AB or Part ABC. Once HMS has completed Part AB or Part ABC, the production data will be re-written to enable that particular product to be picked up from the start of the production (conveyor) straight to the end of the production (drop-zone) unless the customer requires Part AB to be further manufactured into Part ABC.

Converting PML file into .PRO recipe files.
This requirement is dedicated for this implementation only. HMS can be considered as a legacy system, and it uses .PRO recipe files to store production data (Refer Appendix for the difference between .PRO file and PML file format). Hence, for every production data request, XCHANGE needs to convert the PML file on the fly to .PRO format (or a manipulated .PRO format for file transfer protocol).

Extracting information from PML file.
The actual size of PML file could be very big. The whole file contains irrelevant information for certain requests. Hence, it has to be **filtered** before responding to the request. This filtering will enable only the relevant information to be sent and hence a smaller file size (as compared to the full PML file).

Exhibit 24: Extract and Convert to desired format

PML FILE

```
<ProductionStatus>
          <Info> Not in Production> </Info>
          <Operation/>
</ProductionStatus>
.
.
.
<ProductionInfo>
            <Machineoperations>14</Machineoperations>
            <Machine>Robot1</Machine>
            <ProgramNumber>1</ProgramNumber>
       .
       .
</ProductionInfo>
```

6.2.1. Implementation Approach

The DSDM approach segments different functionalities into Timeboxes as indicated in Exhibit 25.

In Timebox 1, the implementation focus is on the basic communication between PH and XCHANGE. This includes TCP/IP Client-Server architecture as well as other defined protocols (Refer Section 6.4) and standards.

Exhibit 25: MOSCOW Prioritisation for PH & XCHANGE Integration

TIMEBOX 1: BASIC	AIM
Communication Protocols	MUST
Read Capability	MUST
Read Recipe Info	MUST
Local PML Access	MUST
Single File Access	MUST
Single Client Access	MUST
EPC and PML standard	MUST

TIMEBOX 2: ADVANCED	AIM
Write Capability	MUST
PML to .pro conversion	Ought
Pseudo-XQL Dynamic	Ought
PML information	Should
Exception Handling	Should
Timestamp Information	Could
PML ServerCould	Could

TIMEBOX 3: SOPHISTICATED	AIM
Multiple File Access	Must
Multiple Client Access	Ought
Single Database for Clients	Ought
Interactivity btwn Client Global	Ought
PML using ONS	Should
XQL	Should

In Timebox 2, a more advanced feature will enable a large PML file to be extracted according to the information needed by the client and converted into the client's legacy format. This will enable minimal disruption to any legacy system that XCHANGE will be "plugged" into in the future.

In Timebox 3, XCHANGE will be developed to support multi-client access, not just PH but other applications as well. To do so, it must have a single database that is flexible and scalable for other applications to use. It will have the capability to "plug n play" different converters, to enable rapid and efficient future integration.

6.2.2. Development Results

XCHANGE has been developed into a **pseudo** idiot-savant. It has most of the requirements that constitute an idiot-savant, but lacks the ability to contact the PML server to retrieve information from the PML files. All the PML files are stored locally within XCHANGE. The full development result are shown in Exhibit 26.

Exhibit 26: PH & XCHANGE IntegrationResult as compared with intended MOSCOW.

TIMEBOX 1: BASIC	AIM	RESULT
Communication Protocols	MUST	Yes
Read Capability	MUST	Yes
Read Recipe Info	MUST	Yes
Local PML Access	MUST	Yes
Single File Access	MUST	Yes
Single Client Access	MUST	Partial
EPC and PML standard	MUST	Yes

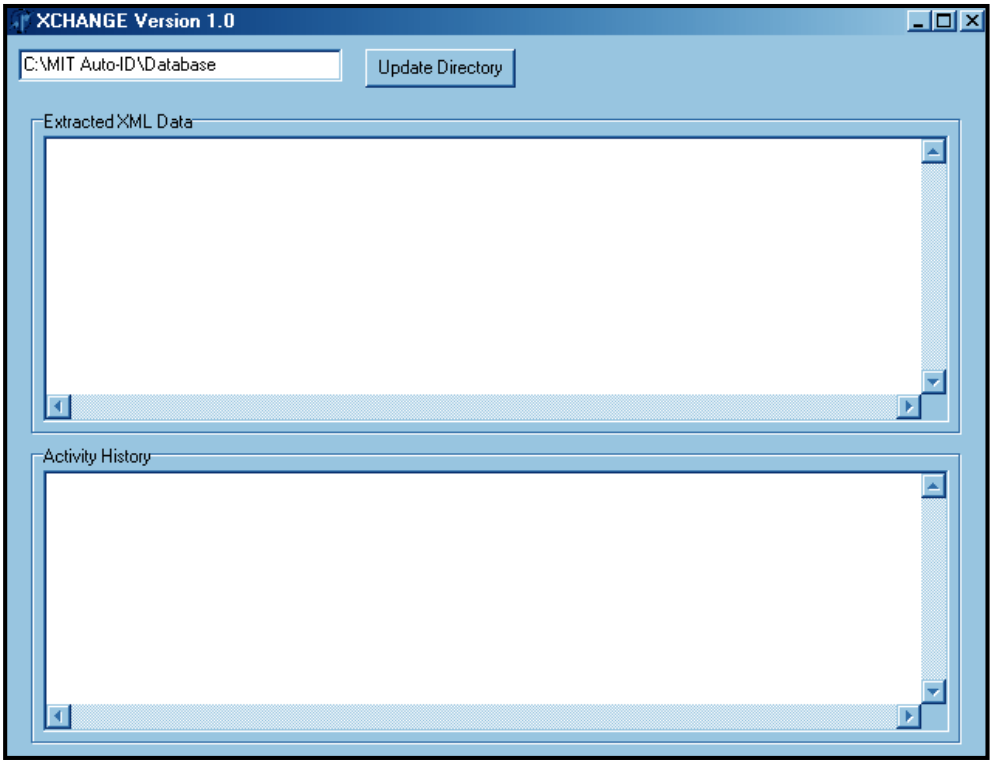
TIMEBOX 2: ADVANCED	AIM	RESULT
Write Capability	MUST	Yes
PML to .pro conversion	Ought	Yes
Pseudo-XQL Dynamic	Ought	Yes
PML information	Should	Partial
Exception Handling	Should	Partial
Timestamp Information	Could	No
PML ServerCould	Could	No

TIMEBOX 3: SOPHISTICATED	AIM	RESULT
Multiple File Access	Must	Yes
Multiple Client Access	Ought	Yes
Single Database for Clients	Ought	Yes
Interactivity btwn Client Global	Ought	Partial
PML using ONS	Should	No
XQL	Should	No

6.2.3. Look and Feel

The user interface of XCHANGE is self-explanatory. When XCHANGE is started, it automatically goes into the server mode, and listens for incoming commands. The “Update Directory” button as shown in Exhibit 27 updates the local PML files directory. The “Extracted XML Data” and “Activity History” are static information regarding the processing of information.

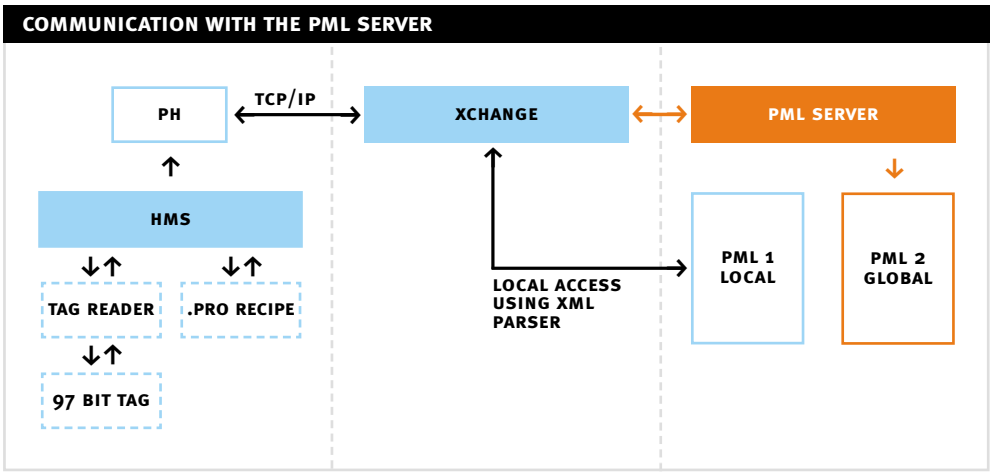
Exhibit 27: Auto-ID Center Database.



6.2.4. Future Development for XCHANGE

There is a difference between a **Global** and **Local** PML file. A **Global** PML server could be accessed anywhere in the world using TCP/IP or other communication protocols. A **Local** PML file is stored locally within the XCHANGE server. A **Local** PML file might contain sensitive information such as production recipe, and hence for security and redundancy reason will not be appropriate to store in the **Global** PML server. The integration so far only involves the **Local** PML file. Hence, the ability to communicate with the PML Server should be the next most important development, as shown in Exhibit 28.

Exhibit 28



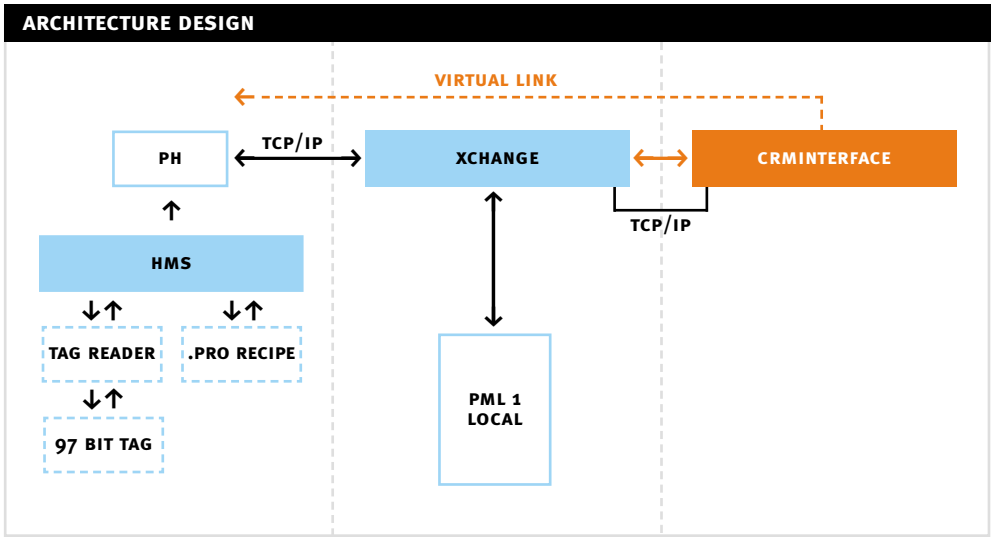
Future development should also include the ability to integrate the XCHANGE straight into the HMS. Hence, the PH will be able to communicate with the PML Server directly instead of via the XCHANGE.

6.3. The CRMInterface Integration with the XCHANGE

The aim of this integration is to provide a mean for customers to interact with the HMS to obtain production transparency and control over their products to be manufactured.

In this architecture design (Exhibit 29), the CRMInterface will **not** communicate directly with the HMS. However, communication will be **indirect** via the XCHANGE. This is possible using a common database of PML files that the CRMInterface and the HMS will use via the XCHANGE.

Exhibit 29



The aims of the CRMInterface are to:

- Establish communication and data transfer with XCHANGE
- Enable customers to manage individual product according to manufacturing requirements
- Provide HMS production transparency to customers

Establishing communication and data transfer with XCHANGE.
CRMInterface will use the existing TCP/IP communication layers to establish basic communication with XCHANGE. Once that is established, communication protocols (Refer Section 6.4) will enable data transfer to and from XCHANGE.

Enabling customers to manage individual product according to manufacturing requirements.
Using CRMInterface, customers will have the ability to choose the exact unique product (as identified by EPC) and customize according to requirements. In this implementation, customers could choose to produce Part AB or Part ABC.

16 Date and time a product is at certain location.

Providing the HMS production transparency to customers.
With this feature, customers could keep track of the product as it is produced in the factory. Status of product could include location and time it is at the location (timestamp[16]), physical make of product and other relevant information. In this implementation, customers will be able to get the status production of “Not in production”, “Part AB Completed” and “Part ABC completed” as updated by HMS. With this transparency, customer could “add on” Part C when Part AB is completed or if product is not in production.

6.3.1. Implementation Approach

As in Section 5.2.1, the TimeBoxes for this software delpoment are as shown in Exhibit 30.

Exhibit 30: MOSCOW Prioritisation for CRMInterface & XCHANGE Integration

TIMEBOX 1: BASIC	AIM
Communication Protocols	MUST
Read Capability	MUST
Read Production Status	MUST
Single File Access	MUST
Single Client Access	MUST
EPC and PML standard	Ought
Activity History	Ought

TIMEBOX 2: ADVANCED	AIM
Write Capability	MUST
Pseudo-XQL	Ought
Same database as HMS	Ought
Dynamic PML information	Should
Exception Handling	Should

TIMEBOX 3: SOPHISTICATED	AIM
Multiple File Access	Must
Make AB	Ought
Make ABC	Ought
Cancel production	Should
Multiple client access	Ought
Interactivity with HMS	Ought

The aims of these timeboxes are somewhat similar to PH & XCHANGE integration. The main difference is in TimeBox 3, where CRMInterface will have the capability to enable customers to choose whether they would like to manufacture Part AC or Part ABC. However, once Part AC and Part ABC are made, the manufacturing process cannot be reversed, ie. Part B cannot be removed from Part ABC to produce part AC.

6.3.2. Development Results

The development results are as shown in Exhibit 31.

The exception handlings are partially handled. Unambiguous commands will not be executed, but there is no fault correction logic built into it. For example, once Part AC and Part ABC are made, the manufacturing process cannot be reversed, ie. Part C cannot be removed from Part ABC to produce part AC. If for instance, Part AC is requested after Part ABC is made, HMS **will still execute** the production for Part AC if the product is detected on the conveyor belt. Therefore the CRMInterface should be used with great care at this stage, and user **should not** blindly request for production without using the Production Status as a guide.

Exhibit 31: CRMInterface & XCHANGE IntegrationResult as compared with intended MOSCOW.

TIMEBOX 1: BASIC	AIM	RESULT
Communication Protocols	MUST	Yes
Read Capability	MUST	Yes
Read Production Status	MUST	Yes
Single File Access	MUST	Yes
Single Client Access	MUST	Yes
EPC and PML standard	Ought	Partial
Activity History	Ought	Yes

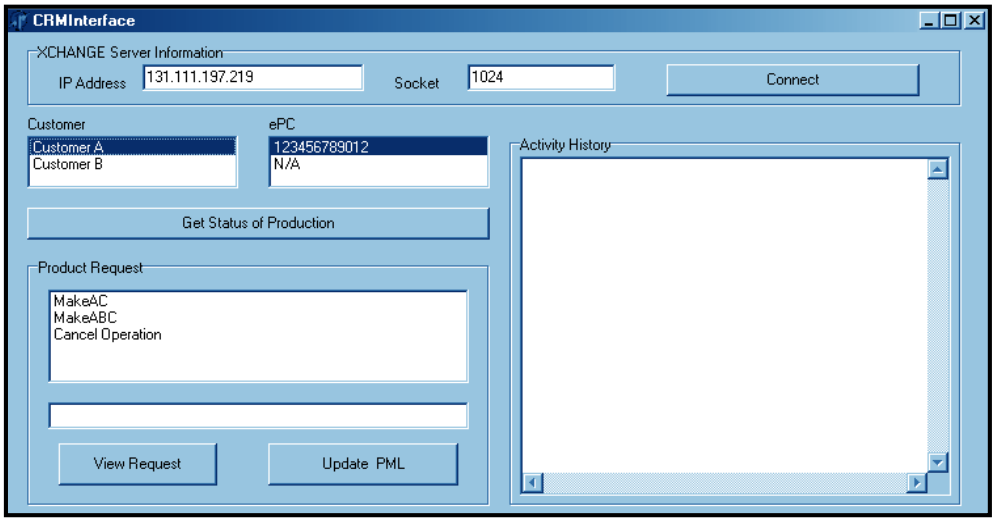
TIMEBOX 2: ADVANCED	AIM	RESULT
Write Capability	MUST	Yes
Pseudo-XQL	Ought	Yes
Same database as HMS	Ought	Yes
Dynamic PML information	Should	Partial
Exception Handling	Should	Partial

TIMEBOX 3: SOPHISTICATED	AIM	RESULT
Multiple File Access	Must	Yes
Make AB	Ought	Yes
Make ABC	Ought	Yes
Cancel production	Should	No
Multiple client access	Ought	Yes
Interactivity with HMS	Ought	Partial

6.3.3. Look and Feel

The user interface for CRMInterface was developed to be as user-friendly as possible. As shown in Exhibit 32, the user has to key in the IP address and socket number of XCHANGE and press the "Connect" button to establish TCP/IP communication with XCHANGE.

Exhibit 32



Once the communication is established, Customer A and Customer B can "Get Status of Production" near real-time. The status will be displayed in the "Activity History". Customers can then choose to make the appropriate product and "Update PML" to execute production.

6.3.4. Future Development for CRMInterface

One of the most important future developments for CRMInterface is a "fool-proof" user interface. This means that customers are restricted to making certain blatantly impossible requests such as making Product AC although that particular product is already made into Product ABC. This will increase safety and security of the programme.

6.4. Summary of Communication Protocols

The flow diagram below shows the communication protocols and procedure after the basic TCP/IP communication has been established. Any client that needs to initiate communication with XCHANGE will need to send a string of commands containing EPC identification number and the service number in the following format: **EPC,Service_Number,Mode**

Exhibit 33

COMMUNICATION PROTOCOLS AND PROCEDURE		
EPC, Service_Number, Mode	Requests by Client	Response by XCHANGE Server
EPC,1	PH requests for Production Info	Sends recipe[17] from EPC.xml
EPC,2,0	PH updates status: **Not in Production**	Updates ‹ProductionStatus› in 'EPC.xml' to **Not in Production**. Sends 'ACK' for acknowledgement.
EPC,2,1 **and** Status of product is "Not Manufactured"	PH updates status: **Part AB Manufactured**	Updates ‹ProductionStatus› in 'EPC.xml' to **Part AB completed.** Also replaces ‹ProductionInfo› node with Manufactured.xml. Sends 'ACK' for acknowledgement.
EPC,2,1 **and** Status of product is "Part AB Manufactured"	PH updates status: **Part ABC Manufactured**	Updates ‹ProductionStatus› in 'EPC.xml' to **Part ABC completed**. Also replaces ‹ProductionInfo› node with Manufactured.xml. Sends 'ACK' for acknowledgement.
EPC,3,MakeAB	CRMInterface[18] requests for Part AB to be produced.	Updates ‹ProductionInfo› node in 'EPC.xml' with MakeAB.xml. Sends 'PML Updated'.
EPC,3,MakeABC	CRMInterface requests for Part ABC to be produced.	Updates ‹ProductionInfo› node in 'EPC.xml' with MakeABC.xml. Sends 'PML Updated'.
EPC,4	CRMInterface requests for Production Status.	Sends ‹ProductionStatus› value from EPC.xml.

[17] Example of format of recipe: 14,Table1,0,Robot1,1,Table1,1, Flipper1,0

[18] Each customer has a CRMInterface. XCHANGE supports multi-customers at the same time.

The **EPC** number is the product identification number and this number is used by XCHANGE to search the database for '**EPC**.xml' PML file in the local drive. The EPC number in this case is 123456789012 for Customer A and 098765432109 for Customer B. Once the file is loaded, XCHANGE will use the **Service_Number** to decide what to do with the data and how to respond to the client. Exhibit 33 shows the summary of request and response.

7. CONCLUSION AND RECOMMENDATIONS

The integration of Auto-ID tagging system with the HMS is feasible. Both systems has similar and consistent philosophy, and together, they have the potential to revolutionize the manufacturing industry, as well as push their boundaries for the realisation of e-Manufacturing[19].

[19] As defined by C.Y.Wong, M.H.Kuok, M.Dunne in E-Manufacturing: The Benefits of its Adoption, Manufacturing Engineering Department.

It is recommended that more advanced manufacturing scenarios to be scoped in terms of business feasibility and developed technically into a full-fledged working software.

The possible scenarios are as follow:

- Customer rush order prioritisation, delaying current WIP execution
- Automated renegotiation with clients and suppliers sourcing based on inventory constraint
- WIP product customisation redefinition by customers enabling direct production execution changes during assembly process
- WIP reidentification to retrieve information about the product assembly stage

8. APPENDIX

8.1. Machine Cell Start-Up Procedure

– Switch on Pneumatic Air
– Power up PLC
– Switch on Extension Lead Socket
– Power up Conveyor Belt
– Power up Cell (Key in lock position, Press green switch for more than 5 seconds)
– Switch Hirata to Auto Mode. Load Syswin 3.3 and download Hirata programme.
– Power up Fanuc (three power switches, starting with the higher levels).
– Power up PC One: Load BBS, MB, Flipper, Table, Hirata.
– Power up PC Two: Load Robot Console and other product holons. For each product holons,
– Change Blackboard, Change Broker and Register.
– Test each product holon independently. (In this case, run Table programme number 1 only).

8.2. Machine Cells Programming Number

Flipper
0 – Reverse flip-over unit for use by agent
1 – Release control of flip-over unit
2 – Flip parts

Table
0 – Use jig at position 0 (near Fanuc)
2 – Use jig at position 2 (near Hirata)
4 – Unclamp, leave & relinquish control of jig
5 – Release hold on table movement
6 – Clamp jig
7 – Unclamp jig

Hirata
0 – Screw Part B to Part A (one screw)
1 – Screw Part AB to Part C (two screws)

Fanuc

PROGRAMMING NUMBERS	FILENAME	DESCRIPTION
0	A_CJ	Pick Part A from Conveyor to Jig
1	C_CJ	Pick Part C from Conveyor to Jig
2	B_CJ	Pick Part B from Conveyor to Jig
3	AC_JOa	Pick Part AC from Jig to Output A
4	AC_JOb	Pick Part AC from Jig to Output B
5	AC_JF	Pick Part AC from Jig to Flipper
6	AC_FJ	Pick Part AC from Flipper to Jig
7	ABC_JOa	Pick Part ABC from Jig to Output A
8	ABC_JOb	Pick Part ABC from Jig to Output B
9	AC_JOa	Pick Part AC from Jig to Output A
10	AC_JOb	Pick Part AC from Jig to Output B
11	AC_CF	Pick Part AC from Conveyor to Flipper
12	ABC_JF	Pick Part ABC from Jig to Flipper
13	ABC_FOa	Pick Part ABC from Flipper to Output A
14	ABC_FOb	Pick Part ABC from Flipper to Output B
15	AC_COa	Pick Part AC from Conveyor to Output A
16	AC_COb	Pick Part AC from Conveyor to Output B

8.3. .PRO file for MakeAC for Customer A (Output A)

[20] Note that this is just a statement explaining the operations. It **can** be in the .PRO file and it **can** also be left out.

```
10          'Number of Operations = 10 [20]
Table 1     'Machine Name with associated programme number as follow (Reserve Table)
0
Robot 1     'Pick Part A from conveyor to jig
0
Table 1     'Clamp Part A
6
Robot1      'Pick Part C from conveyor to jig
1
Table1      'Rotate Table 180 degrees (Absolute Position Near Hirata)
2
Hirata1     'Fasten one screw on Part AC
0
Table1      'Rotate Table 180 degrees (Absolute Position Near Fanuc)
0
Table1      'Unclamp both Part AC
7
Robot1      'Pick both part to Output A
3
Table1      'Release table resource
4
```

8.4. .PRO file to produce ABC for Customer A (Output A) with the condition that Part AC is on conveyor

```
14          'Number of Operations = 14
Flipper 1   'Reserve Flipper
0
Robot 1     'Pick Part AC from conveyor to jig (Need to clamp?)
11
Flipper 1   'Flip Part AC
2
Table 1     'Reserve Table
0
Robot 1     'Pick Part B from conveyor to jig
2
Robot 1     'Pick Part AC from flipper to jig
6
Flipper 1   'Release Flipper Resource
1
Table 1     'Clamp Part ABC
6
Table 1     'Rotate Table 180 degrees
2
Hirata1     'Fasten two screws on Part ABC
1
Table 1     'Rotate Table 180 degrees
0
Table 1     'Unclamp Part ABC
7
Robot 1     'Pick Part A from jig to Output A
7
Table 1     'Release table resource
4
```

8.5. .PRO files for to move Part AC back to Output A if Part ABC is not required to be manufactured

```
1           'Number of Operations = 1
Robot1      'Pick up Part AC from Conveyor to Output A
15
```

8.6. .PRO files for to move Part AC back to Output B if Part ABC is not required to be manufactured

```
1           'Number of Operations = 1
Robot1      'Pick up Part AC from Conveyor to Output B
16
```

8.7. PML FILE: 0123456789012.xml

```
<?xml version="1.0"?>
<NODE LABEL="Meter Box" EPC="123456789012">
 <!-- Most up-to-date product location at particular time
Dates measured from January 1, 2000 00:00:00 Greenwich Mean Time
-->
   <TIMESTAMP>
       <DATE>1213898989</DATE>
       <LOCATION LABEL="Reader A">
       </LOCATION>
       <DESC>Factory B, East Wing, Cell 20, Conveyor 2</DESC>
   </TIMESTAMP>
   <!-- Current Production Status of the product. Elements for INFO
could be Not Manufactured or Part AB Manufactured or Part ABC
Manufactured. -->
   <PRODUCTIONSTATUS>
       <INFO>Not manufactured</INFO>
   </PRODUCTIONSTATUS>
   <!-- Known information about the future of the product -->
   <FUTUREINFO>
       <NEXTOWNER>Customer A</NEXTOWNER>
       <DUEDATE>123456799999</DUEDATE>
   </FUTUREINFO>
   <!-- Production information about the product -->
   <PRODUCTIONINFO>
       <MACHINEOPERATIONS>10</MACHINEOPERATIONS>
       <OPERATION>
             <OPERATIONNUMBER>1</OPERATIONNUMBER>
             <MACHINE>Table1</MACHINE>
             <PROGRAMNUMBER>0</PROGRAMNUMBER>
       </OPERATION>
       <OPERATION>
             <OPERATIONNUMBER>2</OPERATIONNUMBER>
             <MACHINE>Robot1</MACHINE>
             <PROGRAMNUMBER>0</PROGRAMNUMBER>
       </OPERATION>
             <OPERATION>
             <OPERATIONNUMBER>3</OPERATIONNUMBER>
             <MACHINE>Robot1</MACHINE>
             <PROGRAMNUMBER>1</PROGRAMNUMBER>
       </OPERATION>
       <OPERATION>
             <OPERATIONNUMBER>4</OPERATIONNUMBER>
             <MACHINE>Table1</MACHINE>
             <PROGRAMNUMBER>6</PROGRAMNUMBER>
       </OPERATION>
       <OPERATION>
             <OPERATIONNUMBER>5</OPERATIONNUMBER>
             <MACHINE>Table1</MACHINE>
             <PROGRAMNUMBER>2</PROGRAMNUMBER>
```

```
        </OPERATION>
        <OPERATION>
              <OPERATIONNUMBER>6</OPERATIONNUMBER>
              <MACHINE>Hirata1</MACHINE>
              <PROGRAMNUMBER>0</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
              <OPERATIONNUMBER>7</OPERATIONNUMBER>
              <MACHINE>Table1</MACHINE>
              <PROGRAMNUMBER>0</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
              <OPERATIONNUMBER>8</OPERATIONNUMBER>
              <MACHINE>Table1</MACHINE>
              <PROGRAMNUMBER>7</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
              <OPERATIONNUMBER>9</OPERATIONNUMBER>
              <MACHINE>Robot1</MACHINE>
              <PROGRAMNUMBER>3</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
              <OPERATIONNUMBER>10</OPERATIONNUMBER>
              <MACHINE>Table1</MACHINE>
              <PROGRAMNUMBER>4</PROGRAMNUMBER>
        </OPERATION>
    </PRODUCTIONINFO>
    <!-- Product History -->
    <HISTORY>
        <OWNER NAME="Moulding Inc">
              <DESC>Supplier</DESC>
              <ACQUIRE_DATE>1235687999</ACQUIRE_DATE>
              <RELEASE_DATE>1235688023</RELEASE_DATE>
        </OWNER>
    </HISTORY>
</NODE>
```

8.8. PML FILE: 0123456789012.xml

```
<?xml version="1.0"?>
<NODE LABEL="Meter Box" EPC="09876543210">
        <!-- Most up-to-date product location at particular time
Dates measured from January 1, 2000 00:00:00 Greenwich Mean Time
-->
        <TIMESTAMP>
              <DATE>1213898989</DATE>
              <LOCATION LABEL="Reader A">
              </LOCATION>
              <DESC>Factory B, East Wing, Cell 20, Conveyor 2</DESC>
        </TIMESTAMP>
        <!-- Current Production Status of the product. Elements for
INFO could be Not Manufactured or Part AB Manufactured or Part ABC
Manufactured. -->
        <PRODUCTIONSTATUS>
              <INFO>Not manufactured</INFO>
        </PRODUCTIONSTATUS>
        <!-- Known information about the future of the product -->
        <FUTUREINFO>
              <NEXTOWNER>Customer B</NEXTOWNER>
              <DUEDATE>123456799999</DUEDATE>
        </FUTUREINFO>
        <!-- Production information about the product -->
        <PRODUCTIONINFO>
              <MACHINEOPERATIONS>10</MACHINEOPERATIONS>
              <OPERATION>
                    <OPERATIONNUMBER>1</OPERATIONNUMBER>
                    <MACHINE>Table1</MACHINE>
                    <PROGRAMNUMBER>0</PROGRAMNUMBER>
              </OPERATION>
              <OPERATION>
                    <OPERATIONNUMBER>2</OPERATIONNUMBER>
                    <MACHINE>Robot1</MACHINE>
                    <PROGRAMNUMBER>0</PROGRAMNUMBER>
              </OPERATION>
              <OPERATION>
                    <OPERATIONNUMBER>3</OPERATIONNUMBER>
                    <MACHINE>Robot1</MACHINE>
                    <PROGRAMNUMBER>1</PROGRAMNUMBER>
              </OPERATION>
              <OPERATION>
                    <OPERATIONNUMBER>4</OPERATIONNUMBER>
                    <MACHINE>Table1</MACHINE>
                    <PROGRAMNUMBER>6</PROGRAMNUMBER>
              </OPERATION>
              <OPERATION>
                    <OPERATIONNUMBER>5</OPERATIONNUMBER>
                    <MACHINE>Table1</MACHINE>
                    <PROGRAMNUMBER>2</PROGRAMNUMBER>
```

```
            </OPERATION>
            <OPERATION>
                  <OPERATIONNUMBER>6</OPERATIONNUMBER>
                  <MACHINE>Hirata1</MACHINE>
                  <PROGRAMNUMBER>0</PROGRAMNUMBER>
            </OPERATION>
            <OPERATION>
                  <OPERATIONNUMBER>7</OPERATIONNUMBER>
                  <MACHINE>Table1</MACHINE>
                  <PROGRAMNUMBER>0</PROGRAMNUMBER>
            </OPERATION>
            <OPERATION>
                  <OPERATIONNUMBER>8</OPERATIONNUMBER>
                  <MACHINE>Table1</MACHINE>
                  <PROGRAMNUMBER>7</PROGRAMNUMBER>
            </OPERATION>
            <OPERATION>
                  <OPERATIONNUMBER>9</OPERATIONNUMBER>
                  <MACHINE>Robot1</MACHINE>
                  <PROGRAMNUMBER>4</PROGRAMNUMBER>
            </OPERATION>
            <OPERATION>
                  <OPERATIONNUMBER>10</OPERATIONNUMBER>
                  <MACHINE>Table1</MACHINE>
                  <PROGRAMNUMBER>4</PROGRAMNUMBER>
            </OPERATION>
      </PRODUCTIONINFO>

   <!-- Product History -->
   <HISTORY>
            <OWNER NAME="Moulding Inc">
                  <DESC>Supplier</DESC>
                  <ACQUIRE_DATE>1235687999</ACQUIRE_DATE>
                  <RELEASE_DATE>1235688023</RELEASE_DATE>
            </OWNER>
   </HISTORY>
</NODE>
```

8.9. PML FILE: MakeAC.xml

```
<PRODUCTIONINFO>
      <MACHINEOPERATIONS>10</MACHINEOPERATIONS>
      <OPERATION>
           <OPERATIONNUMBER>1</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>0</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>2</OPERATIONNUMBER>
           <MACHINE>Robot1</MACHINE>
           <PROGRAMNUMBER>0</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>3</OPERATIONNUMBER>
           <MACHINE>Robot1</MACHINE>
           <PROGRAMNUMBER>1</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>4</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>6</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>5</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>2</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>6</OPERATIONNUMBER>
           <MACHINE>Hirata1</MACHINE>
           <PROGRAMNUMBER>0</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>7</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>0</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>8</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>7</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>9</OPERATIONNUMBER>
           <MACHINE>Robot1</MACHINE>
           <PROGRAMNUMBER>3</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>10</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>4</PROGRAMNUMBER>
      </OPERATION>
</PRODUCTIONINFO>
```

8.10. PML FILE: MakeABC.xml (With Part AC on Conveyor)

```
<PRODUCTIONINFO>
      <MACHINEOPERATIONS>14</MACHINEOPERATIONS>
      <OPERATION>
           <OPERATIONNUMBER>1</OPERATIONNUMBER>
           <MACHINE>Flipper1</MACHINE>
           <PROGRAMNUMBER>0</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>2</OPERATIONNUMBER>
           <MACHINE>Robot1</MACHINE>
           <PROGRAMNUMBER>11</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>3</OPERATIONNUMBER>
           <MACHINE>Flipper1</MACHINE>
           <PROGRAMNUMBER>2</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>4</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>0</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>5</OPERATIONNUMBER>
           <MACHINE>Robot1</MACHINE>
           <PROGRAMNUMBER>2</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>6</OPERATIONNUMBER>
           <MACHINE>Robot1</MACHINE>
           <PROGRAMNUMBER>6</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>7</OPERATIONNUMBER>
           <MACHINE>Flipper1</MACHINE>
           <PROGRAMNUMBER>1</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>8</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>6</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>9</OPERATIONNUMBER>
           <MACHINE>Table1</MACHINE>
           <PROGRAMNUMBER>2</PROGRAMNUMBER>
      </OPERATION>
      <OPERATION>
           <OPERATIONNUMBER>10</OPERATIONNUMBER>
```

```
            <MACHINE>Hirata1</MACHINE>
            <PROGRAMNUMBER>1</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
            <OPERATIONNUMBER>11</OPERATIONNUMBER>
            <MACHINE>Table1</MACHINE>
            <PROGRAMNUMBER>0</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
            <OPERATIONNUMBER>12</OPERATIONNUMBER>
            <MACHINE>Table1</MACHINE>
            <PROGRAMNUMBER>7</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
            <OPERATIONNUMBER>13</OPERATIONNUMBER>
            <MACHINE>Robot1</MACHINE>
            <PROGRAMNUMBER>7</PROGRAMNUMBER>
        </OPERATION>
        <OPERATION>
            <OPERATIONNUMBER>14</OPERATIONNUMBER>
            <MACHINE>Table1</MACHINE>
            <PROGRAMNUMBER>4</PROGRAMNUMBER>
        </OPERATION>
    </PRODUCTIONINFO>
```

8.11. PML File: PartManufacturedOutputA.xml

```
    <PRODUCTIONINFO>
        <MACHINEOPERATIONS>1</MACHINEOPERATIONS>
        <OPERATION>
            <OPERATIONNUMBER>1</OPERATIONNUMBER>
            <MACHINE>Robot1</MACHINE>
            <PROGRAMNUMBER>15</PROGRAMNUMBER>
        </OPERATION>
    </PRODUCTIONINFO>
```

8.12. PML File: PartManufacturedOutputB.xml

```
    <PRODUCTIONINFO>
        <MACHINEOPERATIONS>1</MACHINEOPERATIONS>
        <OPERATION>
            <OPERATIONNUMBER>1</OPERATIONNUMBER>
            <MACHINE>Robot1</MACHINE>
            <PROGRAMNUMBER>16</PROGRAMNUMBER>
        </OPERATION>
    </PRODUCTIONINFO>
```

8.13. Source Code: XCHANGE

```
unit XCHANGE;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, ExtCtrls, ScktComp, OleServer,
    MSXML_TLB, ComCtrls, ComObj, SelectDir;

type
    TForm1 = class(TForm)
    Memo1: TMemo;
    GroupBox1: TGroupBox;
    ServerSocket1: TServerSocket;
    Memo2: TMemo;
    GroupBox2: TGroupBox;
    TreeView1: TTreeView;
    XMLDocument1: TXMLDocument;
    Edit1: TEdit;
    Button1: TButton;
    procedure GetFileAndRespond (Str:String; sn: integer;
    info: String; Socket: TCustomWinSocket);
    procedure ServerSocket1ClientRead(Sender: TObject;
    Socket: TCustomWinSocket);
    procedure FormCreate(Sender: TObject);
    procedure ServerSocket1Accept(Sender: TObject;
    Socket: TCustomWinSocket);
    procedure Button1Click(Sender: TObject);
    procedure WriteToPML(filename: string; mlist, list:
    IXMLDOMNodelist; Part : String);
    procedure WriteToPML2(filename: string; mlist, list:
    IXMLDOMNodelist; Part : String);
    procedure UpdatePML(filename: string; mlist, list:
    IXMLDOMNodeList; Part : String; NewText:String; Tag1:String;
    Tag2:String);
    procedure ProductionStatus(list: IXMLDOMNodelist; Part : String);

private
    { Private declarations }
public
    XMLDir: String;
    DataList: TStringlist;
    doc: IXMLDOMDocument;
    sdoc: IXMLDOMDocument;
    root, child, child1: IXMLDomElement;
    text1, text2: IXMLDOMText;
    nlist: IXMLDOMNodelist;
    procedure GetChildrenNodes(list:IXMLDOMNodelist; node:TTreeNode);
end;
```

```
var
    Form1: TForm1;

implementation

{$R *.DFM}

// Main Programme
procedure TForm1.FormCreate(Sender: TObject);
begin
    doc := CreateOleObject('Microsoft.XMLDOM') as IXMLDomDocument;
    sdoc := CreateOleObject('Microsoft.XMLDOM') as IXMLDomDocument;
    //Start Server
    ServerSocket1.Active := TRUE;
    //Clear Memos
    Memo1.Lines.Clear;
    Memo2.Lines.Clear;
    XMLDir := 'C:\MIT Auto-ID\Database';
end;

// Upon Client's Request, get EPC string and service number
procedure TForm1.ServerSocket1ClientRead(Sender: TObject; Socket:
TCustomWinSocket);
var
    GivenString : String;
    EPC, mode2 : String;
    index, mode: Integer;
begin
GivenString := Socket.ReceiveText;

// Look for , position within the GivenString
index := Pos(',',GivenString);

if index <> 0 then
begin
    // Get EPC string
    EPC := Copy(GivenString, 1, index-1);
    // Get Service Number
    mode := StrToInt(Copy(GivenString,index+1, 1));
    mode2 := Copy(GivenString,index+3,Length(GivenString));
    // Activate History
    Memo2.Lines.Append(Format('Client %s [Port : %d]:
    Getting %s.xml',
       [Socket.RemoteHost, Socket.RemotePort, EPC]));
    // Use EPC string and Service number to extract from PML files
    GetFileAndRespond(EPC, mode, mode2, Socket);
    end;
end;
```

```
// Load XML Document, respond to Service_no and get Children Nodes
procedure TForm1.GetFileAndRespond(Str:String; sn:integer; info:
String; Socket: TCustomWinSocket);
var
    //Temp : TStrings;
    Node: TTreeNode;
    slist: IXMLDOMNodelist;
    NewText1, Tag1, Tag2 : string;
begin
    //Temp := TStrings.Create;
try
    //Load File into XML Document
    //Temp.LoadFromFile(XMLDir + '\' + Str + '.xml');
    Memo1.Lines.LoadFromFile(XMLDir + '\' + Str + '.xml');
    doc.loadXML(Memo1.Text);
    Memo1.Clear;

    // Displaying XML Root in Tree
    nlist := doc.Get_childNodes;
    TreeView1.Items.Clear;
    Node := TreeView1.Items.Add(NIL, 'XML ROOT');

    // Responding to different service_no
    If (sn = 1) then                    // PH Ask for recipe
    begin
    Memo1.Clear;
    GetChildrenNodes(nlist, Node);
    Socket.SendText(Memo1.Text);   // Send Recipe
    end
    else if (sn = 2) then               // PH Update Status
    begin
    Memo1.Clear;
    // Update PML!!
    // Memo1.Lines.LoadFromFile(XMLDir + '\' + str + '.xml');
    // sdoc.loadXML(Memo1.Text);

    ProductionStatus(nlist,''); // Get Production Status
    // Socket.SendText(Memo1.Text);


If ( info = '0') then               // Not in production
    begin
    Memo1.Clear;
    Memo1.Lines.LoadFromFile(XMLDir + '\' + str + '.xml');
    sdoc.loadXML(Memo1.Text);

    NewText1 := 'Not manufactured';
    Tag1 := 'PRODUCTIONSTATUS';
    Tag2 := 'INFO';
    UpdatePML(str, nlist, slist, info, NewText1,Tag1, Tag2);
```

```
    end
    else if (memo1.text = 'Not manufactured') then    // not
    manufactured to Part AC
    begin
    Memo1.Lines.LoadFromFile(XMLDir + '\' + str + '.xml');
    sdoc.loadXML(Memo1.Text);

    NewText1 := 'Part AC Manufactured';
    Tag1 := 'PRODUCTIONSTATUS';
    Tag2 := 'INFO';

    //Update <ProductionStatus>
    UpdatePML(str, nlist, slist, info, NewText1,Tag1, Tag2);
    //Update Recipe to PartManufactured.xml
    Memo1.Clear;

    If (str='123456789012') then
    Memo1.Lines.LoadFromFile(XMLDir + '\' +
    'PartManufacturedOutputA.xml')
    else
    Memo1.Lines.LoadFromFile(XMLDir + '\' +
    'PartManufacturedOutputB.xml');

    sdoc.loadXML(Memo1.Text);
    slist := sdoc.getElementsByTagName('PRODUCTIONINFO');
    WriteToPML2(str, nlist, slist, info);
    end
    else if (memo1.text = 'Part AC Manufactured') then         //
    Part ABC Produced
    begin
    Memo1.Lines.LoadFromFile(XMLDir + '\' + str + '.xml');
    sdoc.loadXML(Memo1.Text);

    NewText1 := 'Part ABC Manufactured';
    Tag1 := 'PRODUCTIONSTATUS';
    Tag2 := 'INFO';

    //Update <ProductionStatus>
    UpdatePML(str, nlist, slist, info, NewText1, Tag1, Tag2);
    //Update Recipe to PartManufactured.xml
    Memo1.Clear;

    If (str='123456789012') then
    Memo1.Lines.LoadFromFile(XMLDir + '\' +
    'PartManufacturedOutputA.xml')
    else
    Memo1.Lines.LoadFromFile(XMLDir + '\' +
    'PartManufacturedOutputB.xml');

    sdoc.loadXML(Memo1.Text);
    slist := sdoc.getElementsByTagName('PRODUCTIONINFO');
    WriteToPML2(str, nlist, slist, info);
end;
```

```
    Socket.SendText('ACK');       // Acknowledgement
    Memo2.Lines.Append(Format('%s [Port : %d]: %s',
    [Socket.RemoteHost, Socket.RemotePort, 'Acknowledged']));
    end
else if (sn = 3) then           // CRMInterface wants to update PML
    begin
    Memo1.Clear;
    Memo1.Lines.LoadFromFile(XMLDir + '\' + info + '.xml');
    sdoc.loadXML(Memo1.Text);
    slist := sdoc.getElementsByTagName('PRODUCTIONINFO');
    WriteToPML(str, nlist, slist, info);

    Socket.SendText(Format('PML Updated : %s', [info]));
    Memo2.Lines.Append(Format('%s [Port : %d]: %s for request: %s',
    [Socket.RemoteHost, Socket.RemotePort, 'PML Updated', info]));
    end
    else if (sn = 4) then             // CRMInterface needs
    production status
    begin
    ProductionStatus(nlist,'');
    Socket.SendText(Memo1.Text);
    end
    except
    on e:Exception do
    //No advanced exception handlings
    Showmessage(e.message);
end;
end;

// Parse XML and extract required information for use with TreeView
procedure TForm1.GetChildrenNodes(list: IXMLDOMNodelist; node:
TTreeNode);
var
    i : integer;
    Name : String;
    ChildList : IXMLDOMNodeList;
    PNode, XMLNode : IXMLDOMNode;
    ChildNode : TTreeNode;
    value : String;
begin
    for i:=0 to list.Get_length-1 do
    begin
    XMLNode := list.Get_item(i);
    Name := XMLNode.nodename;
    if (XMLNode.Get_nodeType = 3) then
    begin
    value := XMLNode.Get_nodeValue;
    ChildNode := TreeView1.Items.AddChild(node, Value);
    PNode := XMLNode.Get_parentNode;
    if (Uppercase(PNode.Get_nodeName) = 'MACHINEOPERATIONS') then
    Memo1.text := Memo1.text + Value + ','
```

```
    else if (Uppercase(PNode.Get_nodeName) = 'MACHINE') then
    Memo1.text := Memo1.text + Value + ',';
    else if (Uppercase(PNode.Get_nodeName) = 'PROGRAMNUMBER') then
    Memo1.text := Memo1.text + Value + ',';
    end
    else
    ChildNode := TreeView1.Items.AddChild(node, Name);
    ChildList := XMLNode.Get_childNodes;
    GetChildrenNodes(childlist, childnode);
    end;
end;

    // Write into PML for PH (Service_no = 2)
    procedure TForm1.WriteToPML(filename: string; mlist, list:
    IXMLDOMNodeList; Part : String);
    var
    XNodeList: IXMLDOMNodeList;
    NewMNode, MNode, PNode, SNode : IXMLDOMNode;
    begin
    XNodeList := doc.getElementsByTagName('NODE');
    MNode := XNodeList.item[0];
    SNode := list.Get_item(0);
    XNodeList := doc.getElementsByTagName('PRODUCTIONINFO');
    PNode := XNodeList.item[0];
    NewMNode := MNode;
    NewMNode.replaceChild(SNode, PNode);
    doc.save(XMLDir + '\' + filename + '.xml');
end;

    // Write into PML for CRMInterface (Service_no = 3)
    procedure TForm1.WriteToPML2(filename: string; mlist, list:
    IXMLDOMNodeList; Part : String);
    var
    XNodeList: IXMLDOMNodeList;
    NewMNode, MNode, PNode, SNode : IXMLDOMNode;
    begin
    XNodeList := doc.getElementsByTagName('NODE');
    MNode := XNodeList.item[0];
    SNode := list.Get_item(0);
    XNodeList := doc.getElementsByTagName('PRODUCTIONINFO');
    PNode := XNodeList.item[0];
    NewMNode := MNode;
    NewMNode.replaceChild(SNode, PNode);
    doc.save(XMLDir + '\' + filename + '.xml');
    end;

    // Update PML
    procedure TForm1.UpdatePML(filename: string; mlist, list:
    IXMLDOMNodeList; Part : String; NewText:String; Tag1:String;
    Tag2:String);
    var
```

```
    slist, XNodeList: IXMLDOMNodeList;
    NewMNode, MNode, PNode, SNode : IXMLDOMNode;
    sChildNode, mChildNode, pChildNode :IXMLDOMNode;

begin


    XNodeList := doc.getElementsByTagName(Tag1);
    MNode := XN0deList.item[0];
    mChildNode := MNode.Get_firstChild;

    slist := sdoc.getElementsByTagName(Tag2);
    sNode := slist.Get_item(0);
    sChildNode := sNode.Get_firstChild;
    sChildnode.Set_text(NewText);

    XNodeList := doc.getElementsByTagName(Tag2);
    PNode := XNodeList.item[0];
    pChildNode := PNode.Get_firstChild;

    NewMNode := MNode;
    NewMNode.replaceChild(sNode, pNode);
    doc.save(XMLDir + '\' + filename + '.xml');
end;

// Extract Production Status
procedure TForm1.ProductionStatus(list: IXMLDOMNodelist; Part :
String);
var
i : integer;
Name : String;
ChildList : IXMLDOMNodeList;
PNode, XMLNode : IXMLDOMNode;
value : String;
begin
for i:=0 to list.Get_length-1 do
begin
XMLNode := list.Get_item(i);
Name := XMLNode.nodename;
if (XMLNode.Get_nodeType = 3) then
begin
value := XMLNode.Get_nodeValue;
PNode := XMLNode.Get_parentNode;
if (Uppercase(PNode.Get_nodeName) = 'INF0') then
Memo1.Text := Memo1.Text + Value;
end;
ChildList := XMLNode.Get_childNodes;
ProductionStatus(childlist,'');
end;
end;
```

```
// Display connection in Activity History
procedure TForm1.ServerSocket1Accept(Sender: TObject;
begin
Memo2.Lines.Append(Format('Client Accepted from %s [Port : %d]',
[Socket.RemoteHost, Socket.RemotePort]));
end;

// Get Directory for XML Files
procedure TForm1.Button1Click(Sender: TObject);
var
Frm : TDirForm;
begin
Frm := TDirForm.Create(Self);
Frm.ShowModal;
edit1.Text := Frm.DirectoryListBox1.Directory;
XMLdir := edit1.Text;
Frm.Destroy;
end;


end.
```

8.13. Source Code: CRMInterface

```
unit CRM;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
    StdCtrls, ScktComp;

type
    TCRMInterface = class(TForm)
    ListBox1: TListBox;
    EPC: TLabel;
    Customer: TLabel;
    GroupBox1: TGroupBox;
    Button3: TButton;
    ListBox2: TListBox;
    Button4: TButton;
    Edit1: TEdit;
    ClientSocket1: TClientSocket;
    GroupBox2: TGroupBox;
    GroupBox3: TGroupBox;
    Edit3: TEdit;
    Edit4: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    Button6: TButton;
    memo1: TMemo;
    Button7: TButton;
    ListBox3: TListBox;
    procedure ListBox1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure ClientSocket1read(Sender: TObject; Socket:
    TCustomWin Socket);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);

private
    { Private declarations }
public
    { Public declarations }
    EPCs : array [0..1] of array [0..1] of String;
    Customers : array [0..1] of String;
    Requests : array [0..2] of String;
end;
```

```
var
    CRMInterface: TCRMInterface;

implementation

{$R *.DFM}

procedure TCRMInterface.FormCreate(Sender: TObject);
begin
// Clear Activity History
memo1.clear;

// Data
Customers[0] := 'Customer A';
Customers[1] := 'Customer B';
EPCs[0][0] := '123456789012';
EPCs[0][1] := 'N/A';
EPCs[1][0] := '098765432109';
EPCs[1][1] := 'N/A';
Requests[0] := 'MakeAC';
Requests[1] := 'MakeABC';
Requests[2] := 'Cancel Operation';
ListBox1.Items.Add(Customers[0]);
ListBox1.Items.Add(Customers[1]);
ListBox1.ItemIndex := 0;
ListBox1Click(Self);
ListBox3.Items.Add(Requests[0]);
ListBox3.Items.Add(Requests[1]);
ListBox3.Items.Add(Requests[2]);
end;

procedure TCRMInterface.ListBox1Click(Sender: TObject);
var
    i : Integer;
begin
    i := ListBox1.ItemIndex;
    ListBox2.Items.Clear;
    ListBox2.Items.Add(EPCs[i][0]);
    ListBox2.Items.Add(EPCs[i][1]);
    ListBox2.ItemIndex := 0;
end;

procedure TCRMInterface.Button3Click(Sender: TObject);
var
    i,j, k : Integer;
begin
    i := ListBox1.ItemIndex;
    j := ListBox2.ItemIndex;
    k := ListBox3.ItemIndex;
    if (i < 0) or (j < 0) or (k < 0) then Exit;
    Edit1.Text := Format('%s : %s : %s',[Customers[i], EPCs[i][j],
Requests[k]]);

end;
```

```
procedure TCRMInterface.Button5Click(Sender: TObject);
begin
edit1.Clear;
end;

procedure TCRMInterface.ClientSocket1read(Sender: TObject;
    Socket: TCustomWinSocket);
var
    str : String;
begin
    str := Socket.ReceiveText;
    memo1.Lines.Append(str);
end;

procedure TCRMInterface.Button6Click(Sender: TObject);
var
    Sock : Integer;
begin
    // Connect to Server
    ClientSocket1.Host := Edit3.Text;
    Sock := StrtoInt(Edit4.Text);
    ClientSocket1.Port := Sock;
    ClientSocket1.Active := TRUE;
    // ClientSocket1.Socket.SendText(Memo1.Text);
    // memo1.clear;
    Memo1.Lines.Append(Format('Server Connected at %s [Port : %d]',
        [ClientSocket1.Host, ClientSocket1.Port]));
end;

procedure TCRMInterface.Button4Click(Sender: TObject);
Var
    i,j, k : Integer;
    Result : Word;
begin
    i := ListBox1.ItemIndex;
    j := ListBox2.ItemIndex;
    k := ListBox3.ItemIndex;

If (k = 2) or (i = 1) then
    begin
    Edit1.Clear;
    Edit1.Text := 'Feature not yet available';
    end
    Else
   Result := messagedlg('ARE YOU SURE?', mtConfirmation, [mbYes,
mbNo], 0);
If (Result = mrYes) then
   ClientSocket1.Socket.SendText(EPCs[i][j] + ',3,'+ Requests[k])
Else
   exit;
end;
```

```
procedure TCRMInterface.Button7Click(Sender: TObject);
Var
   i,j : Integer;
begin
   i := ListBox1.ItemIndex;
   j := ListBox2.ItemIndex;
   ClientSocket1.Socket.SendText(EPCs[i][j] + ',4');
end;
end.
```