Published February 1, 2003. Distribution restricted to Sponsors until May 1, 2003.

AUTO-ID CENTRE

WHITE PAPER

Development of a Prototype PML Server for an Auto-ID Enabled Robotic Manufacturing Environment

Mark Harrison, Duncan McFarlane

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

ABSTRACT

The Physical Markup Language (PML) is an XML Schema for representing information about products. This paper describes some of the structure and functionality of a PML server in general terms and reports the ongoing development of a prototype PML Server at the Cambridge (UK) lab of the Auto-ID Centre, in order to store information about objects, batch orders and manufacturing recipes and to make this information available to an Auto-ID enabled robotic manufacturing environment. We report firstly on our initial PML server developments for the Phase 1 robot manufacturing demonstration which used a conventional control system. For the Phase 2 demonstration with holonic control and batch orders, we completely redesigned our prototype PML server to handle much more generic query and update methods aligned more closely with XML query standards.

Published February 1, 2003. Distribution restricted to Sponsors until May 1, 2003.

WHITE PAPER

Development of a Prototype PML Server for an Auto-ID Enabled Robotic Manufacturing Environment

Biographies

Mark Harrison
Senior Research Associate

Mark Harrison is a Senior Research Associate at the Auto-ID Centre lab in Cambridge working on the development of a PML server, web-based graphical control interfaces and manufacturing recipe transformation ideas. In 1995, after completing his PhD research at the Cavendish Laboratory, University of Cambridge on the spectroscopy of semiconducting polymers, Mark continued to study these materials further while a Research Fellow at St. John's College, Cambridge and during 18 months at the Philipps University, Marburg, Germany. In April 1999, he returned to Cambridge, where he has worked for three years as a software engineer for Cambridge Advanced Electronics/Internet-Extra, developing internet applications for collaborative working, infrastructure for a data synchronisation service and various automated web navigation/capture tools. He has also developed intranet applications for his former research group in the Physics department and for an EU R&D network on flat panel displays.

Duncan McFarlane
Research Director Europe

Duncan McFarlane is a Senior Lecturer in Manufacturing Engineering in the Cambridge University Engineering Department. He has been involved in the design and operation of manufacturing and control systems for over fifteen years. He completed a Bachelor of Engineering degree at Melbourne University in 1984, a PhD in the control system design at Cambridge in 1988, and worked industrially with BHP Australia in engineering and research positions between 1980 and 1994. Dr McFarlane joined the Department of Engineering at Cambridge in 1995 where his work is focused in the areas of response and agility strategies for manufacturing businesses, distributed (holonic) factory automation and control, and integration of manufacturing information systems. He is particularly interested in the interface between production automation systems and manufacturing business processes.

Published February 1, 2003. Distribution restricted to Sponsors until May 1, 2003.

WHITE PAPER

Development of a Prototype PML Server for an Auto-ID Enabled Robotic Manufacturing Environment

Contents

1. INTRODUCTION

The Physical Markup Language (PML) is an XML Schema for representing information about products. It is being developed as a freely available open standard to serve as a global, uniform method of conveying information about tagged objects, which can be used by all sectors of industry. In addition to the development of the language specifications in the form of an XML Schema for PML, the Auto-ID Centre is also testing the practical capabilities of the language and building prototype database architectures to test and support it, usually referred to as PML Servers.

This paper reports the ongoing development of a prototype PML Server at the Cambridge (UK) lab of the Auto-ID Centre, in order to store information about objects, batch orders and manufacturing recipes and to make this information available to an Auto-ID enabled robotic manufacturing environment. Two phases of development are reported in this paper:

1. Auto-ID enhanced manufacturing with a conventional control system, assisted by a prototype PML server which used dedicated methods for querying and updating manufacturing recipes specified at a per-box level (mass-customisation).

2. Auto-ID enhanced manufacturing with a holonic control system, assisted by a redesigned prototype PML server with more generic query and update methods and representation of batch orders in the Physical Markup Language.

We begin with a brief review of the various components of the Auto-ID infrastructure [1], which includes:

1. **Hardware** – such as RFID tags and readers but also including optical readers and other types of sensors.
2. **Savant™ [2]** – a software platform for interfacing to various types of readers and filtering the data generated when the tags are within range of the readers.
3. **The Electronic Product Code™ (EPC™)** [3] – a unique identification number for each object – the minimum information which must be stored on its ID tag.
4. **The object name service (ONS) [4]** – an extension of the domain name service (DNS) [5], which provides a method of converting an EPC™ into the network address of the corresponding networked database, which in turn holds the product data for that object.
5. **The physical markup language (PML) [6, 7]** – an open standard XML Schema for representing information about products and communicating this information globally in a unified manner, between parties who otherwise have disparate databases, data storage formats and who use diverse application programs to act upon the product data.
6. **Control systems [8]** – to provide feedback into the supply chain or manufacturing process, using the data generated by EPC™ readings.

Much of the momentum for the Auto-ID Centre came from industrial sponsors who were looking for a new technology to replace the optical barcode, to provide a higher level of granularity of product data and item-level tracking, and a greater degree of automation in inventory control than could be achieved with the existing UCC/EAN barcodes, which have now been in use for over 25 years. Founder sponsors included manufacturers such as Gillette, Procter & Gamble and global retailers such as Wal-Mart. Many such companies deal with the manufacture or retail of consumer product goods, whose volumes run into several billions each year. The industrial sponsors identified the need for the development of global open standards for uniquely identifying objects and communicating the associated data in a unified manner, while minimising the costs of deployment [9]. For this reason, a major focus of the Auto-ID Centre has been the development of RFID tags which are cheap enough to be disposable and readers which are agile enough to be deployed globally, since they can operate at mutliple frequencies [10] and thereby satisfy diverse legislation on radiation power levels and allowed operating frequency bands in different parts of the world.

In order to develop cheap disposable tags for low value goods produced in large volumes, it is usually necessary to minimise the capacity of the computational memory of each tag [9]. This led to the introduction of a number-plate approach in which at the bare minimum, only an Electronic Product Code™ (EPC™) (typically 96 bits) need be stored on the tag, while the remaining product data could be stored elsewhere on networked databases, with the EPC™ acting as a key to retrieve the data about a particular object at each database on the computer network.

In this paper, we attempt to identify some of the requirements of networked PML Servers which hold the product data, both in general terms and by reference to prototype PML servers we developed for our robot packing cell, which demonstrates various uses of Auto-ID technology in a manufacturing environment. We begin with a brief review of the Electronic Product Code™ (EPC™).

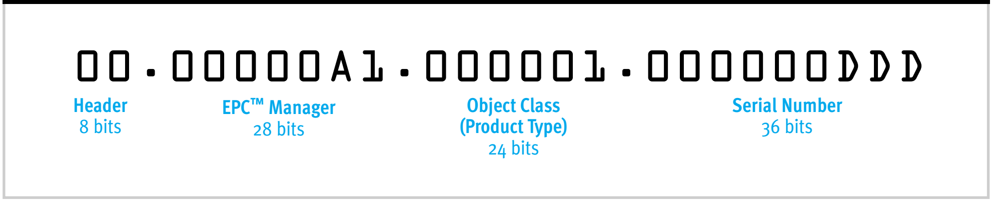
1.1 Electronic Product Code™ (EPC™) as a Key to Access the PML Data

Within the Auto-ID framework, the identity of the product is the Electronic Product Code™ (EPC™) [3], usually a 96-bit number, which consists of four ranges of bits, as depicted in Figure 1:

Header	8 bits	Identifies the partitioning scheme for the EPC™
EPC™ Manager	28 bits	The party responsible for the product (e.g. manufacturer)
Object Class	24 bits	The product type
Serial Number	36 bits	The serial number within that product type

Figure 1

TYPICAL STRUCTURE OF THE ELECTRONIC PRODUCT CODE™ (EPC™)



The 96-bit Type I EPC™ has sufficient capacity to uniquely identify over 10^{28} unique items, with the flexibility through different partitioning schemes to cater for both large volumes and a large number of different types of products per manufacturer.

The EPC™ as such does not contain any detailed information about the object, although it may be possible to deduce the manufacturer and product type. The hierarchical partitioning is used so that the object name service (ONS) can operate in a scalable manner, just as the domain name service (DNS) does for web addresses.

The EPC™ is the database 'key' to lookup the object's PML data, just as a bank account and sort-code is a unique 'key' to lookup an individual bank customer's financial transactions. An object's PML data is in essence very similar to a bank statement – it is also a timestamped record of the history of the physical object corresponding to that EPC™, along with additional information about the product, its specifications, calibrations and sensor measurements and its transactions etc..

1.2. PML as an Interchange Language for Exchanging Data about Products

The physical markup language (PML) uses an XML Schema [11] which is being developed by the Auto-ID Centre as a global open standard for communicating data about products. XML [12] is a flexible markup language which at first glance resembles the HTML (hypertext markup language) [13] currently widely used for writing web pages. The reason for the apparent similarity is that both XML and HTML share roots in SGML – the Standard Generalised Markup Language, although XML is intended to be more rigorous and more flexible. HTML defines a number of markup elements and attributes for the display of a document, such as `<BODY></BODY>` elements which are usually nested within the `<HTML></HTML>` elements. In HTML, the `<BODY>` element may contain attributes such as `BGCOLOR` to define the background colour for a page. Both HTML and XML use nested elements – where one type of element is found within the opening and closing tags of another element – and this hierarchical nesting of elements is often referred to as the document tree or document object model. Figure 2 shows an example of nested elements and attributes. The reader should note that potential for confusion arises in that HTML or XML elements such as `<BODY>` are frequently also referred to as 'tags' in computing literature – and these data tags should not be confused with the physical RFID tags which transmit EPCs™.

Figure 2: Nested elements and attributes of elements

```
    An attribute - BGCOLOR
<HTML>          /
  <BODY BGCOLOR="00FFFF">
    <P>Hello Everyone!</P>    <-- The <P> </P> paragraph element
  </BODY>
</HTML>
```

In marked contrast with the HTML used for layout of web pages, XML does not define the names of such elements and attributes – everyone can define their own choice of names of elements and attributes. However, this has already resulted in several hundred XML markup languages for different sectors of industry [14], of which the majority are conveying data about similar properties, such as mass, geometry, composition, etc. but each using their own particular nomenclature.

Physical markup language [6, 7] is an XML Schema being developed as an open standard by the Auto-ID Centre so that all parties on the supply chain anywhere in the world can communicate information about objects in a unified efficient manner, which everyone could understand, in order to avoid an n x n translation problem between n competing languages, each dedicated to a particular sector of industry.

It is important to note that PML is a communication language and does not prescibe that the underlying data must be stored natively as PML, nor does it dictate which databases should be used, nor the names of tables and fields in which the data is ultimately stored. It is anticipated that many companies will continue to store data about products in their existing relational databases, since these are proven to be robust and can be interrogated using structured query language (SQL) [15] to perform moderately complex queries involving joins between tables and filtering of the results against multiple criteria. However, they would use a translation layer [16] to markup the extracted data values in the standard PML format on-demand when communicating with others outside their own company. This has a number of security advantages over storing the data on the tag directly, since each company retains complete control over which subset of the data it exposes to outsiders and which part remains private. Furthermore, since the EPC™ stored on the tag is immutable, there is much less scope for accidental or malicious over-writing of one company's data by another company as a tagged object passes between them, since each company stores its data in-house rather than on the tag. Another advantage is that there is no need to divulge the database location or structure, nor the connection string or file pathname used to access it. The markup used for communication uses the open standard PML schema and reveals nothing about how or where the underlying data is stored at each company.

1.3. Data at Product-type vs Instance-level Data – Objects as Individuals within a Product Type

Auto-ID technology will allow much finer granularity in the identity of a product and the data which is stored about it. Many current inventory procedures merely count the number of products of a given product type. The Auto-ID infrastructure will speed up object detection by using methods such as radio-frequency identification (RFID) to enable simultaneous detection of several tagged objects at a distance, without requiring an unobstructed line of sight to the tag and thereby reducing the need for human intervention and the accompanying errors. Furthermore, since each unique object will have a unique identifier, in the form of the Electronic Product Code™ (EPC™), each object will become an individual, with its own individual history about how it passed through the supply chain. We refer to uniquely identifiable individual objects of the same product type as **instances** of that product.

The individual history consists not only of location and timestamp information for tracking purposes, but also measurements made directly on the object and its environment at all times. An example in the food industry may be to record the temperature of frozen goods throughout the supply chain and use the fine object-level granularity for audit purposes, to verify for each individual item that the temperature remained below a certain level at all times. Or in the case of a product recall, to be able to identify for an individual item, where and when the problem occurred (such as too high a temperature) – whether in the freezer of the supermarket, the distribution centre or during transportation.

In manufacturing, the higher granularity also facilitates mass-customisation of products, so that instance-level customisation overrides the default or product-level specifications for that product. An example may be a laptop computer, built to order with a larger amount of RAM memory than usual. Furthermore, very late-stage customisation is also possible, since the manufacturing recipe and parameters for a customised product can be stored in a networked database and updated in real time, just before manufacturing begins.

From this discussion, we see that there are two distinct types of data or properties, which can be held about an object:

– **Product-level data**, such as product specifications, capabilities and geometry, which are common to all objects of the same product type.

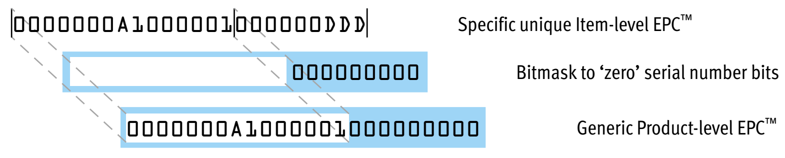
– **Instance-level data**, such as the unique history of an individual object (location, timestamp, sensor measurements) as it passed through the supply chain, as well as individual customisation parameters for that particular object, which over-ride the default product-level values.

1.4. EPCs™ at Instance Level and at Product-type-level

We have already introduced the concept of the Electronic Product Code™ (EPC™) to identify each object uniquely. However, there are occasions when we want to either access standard product-level data directly or perhaps in a manufacturing recipe, to specify the product type of a component which is required, without specifying the exact serial number of the component, which should be used. For such purposes, we propose the use of a product-level EPC™, in which the range of bits which represent the serial number (the last 36 bits for a Type I 96-bit EPC™) are set to zero. An example of such a product-level EPC™ in hexadecimal format is `00000000100000A000000000`, whereas a unique EPC™ of a particular instance of that product type may look like `00000000100000A0000039D7`. Of course, in the case where our manufacturing recipes only specify a product-level EPC™ for the component to be used in the future manufacture, it is important to record the actual unique EPC™ (including the serial number bits) which was actually used when the component was incorporated into the product, in order that we preserve the high granularity of the object tracking. Figure 3 illustrates the idea of bitmasking to convert a unique instance-specific EPC™ into a product-level EPC™ for the corresponding product type.

Figure 3

BITMASKING A UNIQUE INSTANCE-LEVEL EPC™ TO OBTAIN THE PRODUCT-LEVEL EPC™


0000000A1000001|000000DDD
Specific unique Item-level EPC™
000000000
Bitmask to 'zero' serial number bits
0000000A1000001000000000
Generic Product-level EPC™

1.5. Accessing the PML Data

The key to accessing the data about a product is the identity of the product (in the form of the EPC™) and the specification of the property of interest, such as its mass, length, width, height etc., which will be stored at a particular hierarchical level of the PML data, using a meaningful XML element or attribute name which identifies the property.

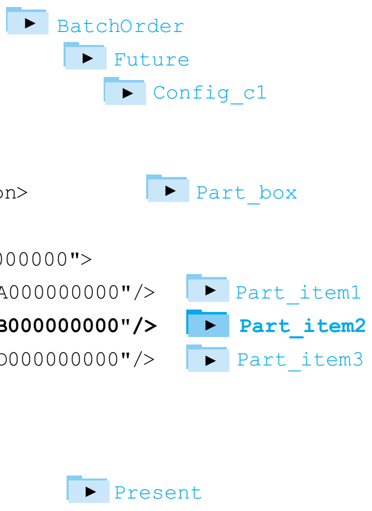
Since the underlying data storage structure of each company is not revealed to outsiders, queries for particular subsets of the PML data are done by reference to a particular path, node or sub-tree within the PML/XML hierarchy and the company's translation layer validates the incoming query and converts it into a corresponding query of the underlying database (e.g. an SQL SELECT statement), then marks up the returned values within the PML Schema.

One of the simplest methods for referring to a particular part of an XML hierarchy is the XPath [17] or XQL [18], which in its abbreviated form resembles a UNIX-style pathname, except that each successive word between the forward slash delimiters refers to a child element nested more deeply within the XML hierarchy than its parent element, rather than a subdirectory located within the parent directory of a UNIX filesystem. Figure 4 illustrates this analogy.

Figure 4: An XPath or XQL expression is used to refer to a particular node of an XML hierarchy, just as a filepath can refer to a particular subdirectory.

```
/BatchOrder/Future/Config[ @label='c1'] /Part[ @label='box'] /Part[ @label='item2']

<?xml version="1.0" encoding="UTF-8"?>
<BatchOrder>
  <Future>
    <Config label="c1">
      <Msr q="1">2</Msr>
      <Price>2</Price>
      <Description>Gillette gift box</Description>
      <Date label="edited">1043060411000</Date>
      <Part label="box" epc="0000000A1000002000000000">
          <Part label="item1" epc="000000000100000A000000000"/>
          <Part label="item2" epc="000000000100000B000000000"/>
          <Part label="item3" epc="000000000100000D000000000"/>
      </Part>
    </Config>
  </Future>
  <Present/>
</BatchOrder>
```



To summarise, when querying a property of an object via its PML data, we can use a combination of:

– A unique object identifier (EPC™).
– An unambigous path to the part of the PML hierarchy of interest (e.g. XPath).

1.6. Access Controls

Of course, not all data about the product will be available to every member of the public, so trusted partners will need to authenticate themselves with the PML server as a particular user of the data. Access to particular sub-trees within the PML data will be restricted to certain users or groups of users, as specified in the security layer, just as particular subdirectories of a filesystem may have restricted read/write permissions for certain users and groups. The authentication process will open up (or close off) access to particular sub-trees of the PML data, depending on which data the user is entitled to view. Much of the history data on a product and its manufacture will be commercially sensitive and restricted to certain groups of users within the company or in exceptional circumstances and certain countries, authorised government agencies, such as those responsible for the public safety of food products and pharmaceuticals. Other parts of the PML data (such as that containing the public-domain product specifications or usage/safety instructions) may be globally readable for most products.

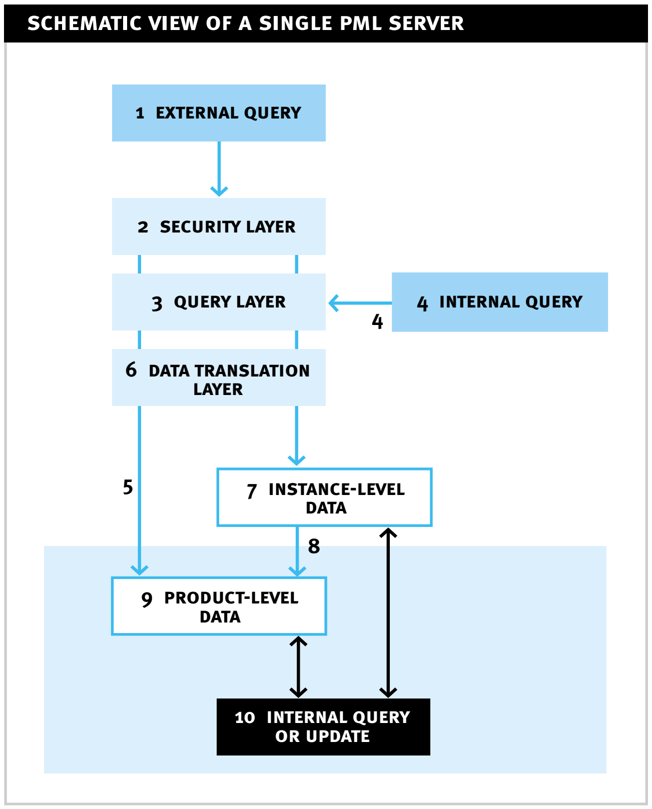


Figure 5: Schematic view of a single PML server showing both product-level and instance-level data and the query, security and data transformation layers involved in responding to an external PML query.

1. From outside the company
2. Exposes subset of PML data tree to user
3. Returns PML sub-tree given (EPC™, XPath))
4. From inside the company alternative PML-based query
5. Product-level data can be accessed directly if a product-level EPC™ is specified
6. Maps RDBMS to PML Schema – both the query and the returned data
7. Relational Database Tables
8. 'Fall-through' to product-level data where instance-level data is not defined
9. Existing Relational Database Tables
10. From within the company including data from Savant™ using existing SQL methods

Figure 5 represents a structure for a single PML server. We now discuss our Auto-ID robotic manufacturing environment and the prototype PML servers we are developing for this purpose.

2. PROTOTYPE PML SERVERS IN AN AUTO-ID ROBOTIC MANUFACTURING ENVIRONMENT

The Robot Packing Cell demonstration [19] developed at the Cambridge (UK) lab of the Auto-ID centre involves packing of Gillette gift boxes, where the empty gift box is available in two alternative configurations and each has three slots for items. In the demonstration, we specify three items from a range of four – razor, foam, gel and deodorant – more than one of each type is also acceptable. We therefore have the objects and EPC™ ranges shown in Figure 6, where the last 9 hexadecimal digits (= 36 bits) represent the range of serial numbers of unique instances of a particular product type and xxxxxxxxx denotes those nine hexadecimal digits 0–9 or A–F.

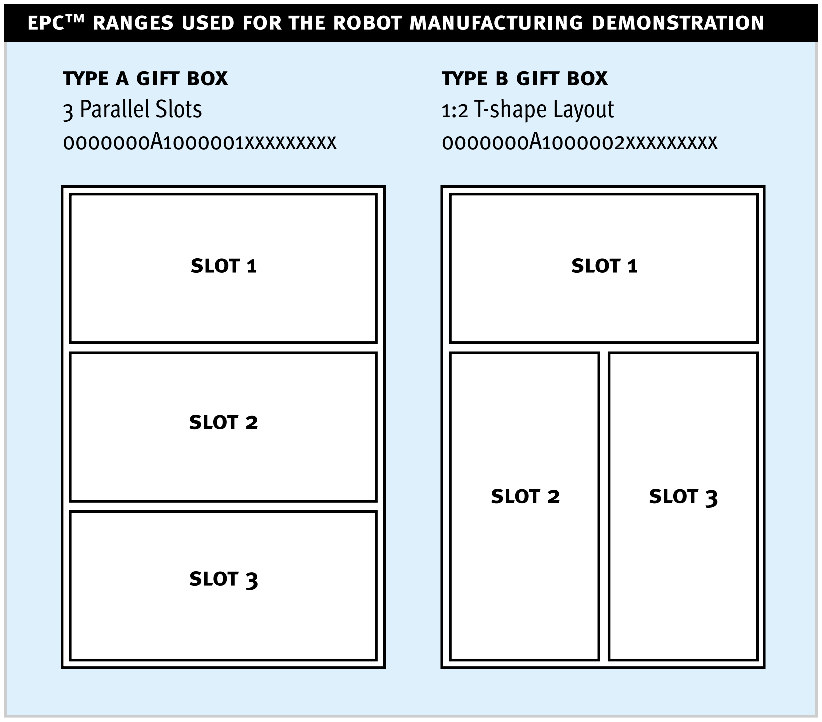
Our Robot Packing Cell demonstration is intended to show the use of Auto-ID technologies such as:

- **RFID tags and readers** – to uniquely identify incoming items and boxes down to the serial number level and without the need for line-of-sight scanning.
- **The Savant™ system** – to filter and aggregate readings from various RFID readers, remove spurious reads and noise and make available timestamped queues of EPC™ readings marked up in the physical markup language (PML).
- **A prototype PML server** – a networked database used to obtain information about a particular item, which might include a customised manufacturing recipe specific to its unique serial number – or negotiation with batch orders.

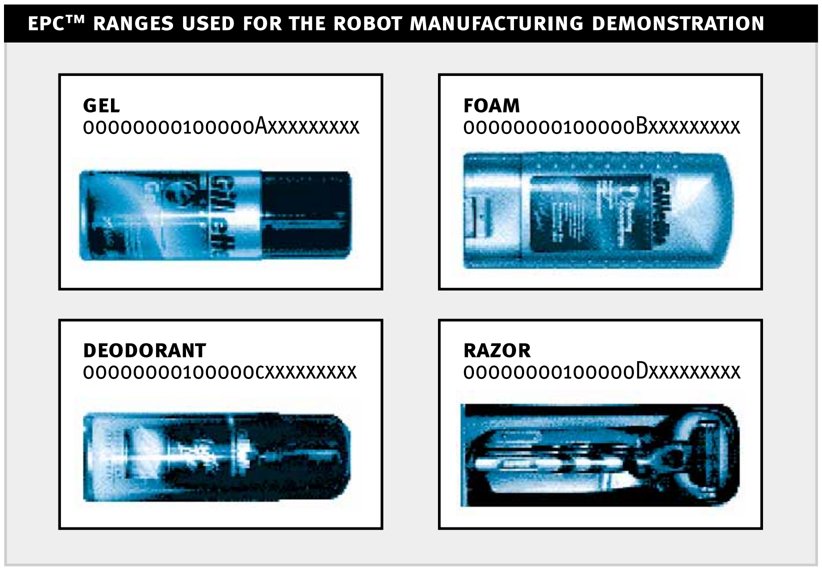
We have also developed a drag-and-drop graphical user interface for easy placement of orders or batch orders, which are then formatted in the Physical Markup Language for use by the robotic manufacturing control systems.

Figure 6: EPC™ ranges used for the robot manufacturing demonstration

Note that xxxxxxxxx denotes the 36 serial number bits – 9 hexadecimal digits [0–9, A–F] which vary for different instances of the same product type



Continuation of Figure 6



2.1. Phase 1 – Mass late-stage Customisation of Pick-and-Place

In Phase 1, the demonstration aims to show how Auto-ID technologies can be used in various manufacturing operations:

- to log and identify incoming goods, storing the unique EPCs™ – and either using the goods immediately for the current packing operation or placing in a storage area.
- to correct disturbances introduced by human error within the storage chutes.
- to identify an empty box and retrieve a PML manufacturing recipe for assembling the custom order for that box.
- packing of the box in accordance with the retrieved recipe.
- recording aggregation events – updating of the PML data for the packed box to record which items (specific EPCs™ including serial number) were actually packed into the box.
- quality control – using RFID readers to simultaneously read EPCs™ of the box and packed items, without line-of-sight access to the packed items – and checking for compliance with the PML recipe associated with that specific box.

2.1.1 Prototype PML representation of a customised order for a single gift box

For the Phase 1 demo, the prototype PML markup shown in Figure 7 was used for an empty box with EPC™ `0000000A1000001000000DDD` (of type A – 3 parallel slots) with a pre-specified order: Razor in slot 1, Gel in slot 2 and Foam in slot 3.

Figure 7: Phase 1 Prototype PML markup for an order placed for a unique box (box EPC™ `0000000A1000001000000DDD`), as the PML appears before the specified items are packed in the box – note that the item EPCs™ specified in the `<future>` block have their serial number bits set to zero, since we only want to specify the item types rather than unique items to pack in the box.

```
<?xml version="1.0" ?>
<node epc="0000000A1000001000000DDD">
<future>
<owner name="retailer">tesco</owner>
<desc>gillete gift pack for tesco summer sale</desc>
    <node epc="00000000100000D000000000" />
    <node epc="00000000100000A000000000" />
    <node epc="00000000100000B000000000" />
</future>

<present>
<owner name="manufacturer">gillette</owner>
<desc>gillette giftbox configuration type A</desc>
</present>

<history>
....
</history>
</node>
```

The root-level `<node>` element has an EPC™ attribute which identifies the unique serial number of the empty gift box. The particular box EPC™ is also used as the PML database key for the PML data shown above. In the Phase 1 demonstration, for simplicity, we associate the order for the finished product directly with the empty gift box.

The `<future>...</future>` block contains information about the order specified for the box – a minimal manufacturing recipe containing a bill of materials ordered in sequence. The three `<node>` elements within the future block represent the three items ordered for that particular gift box. The order in which these are specified is used to correlate with the giftbox slot positions, as indicated in Figure 6.

Within this recipe, the EPC™ attributes of the `<node>` elements in the `<future>` block specify only the product type (i.e. razor/foam/gel/deo) to be inserted into slot 1/2/3 – without specifying down to the particular serial number of the actual item to be used, so that the most conveniently accessed item of the correct type can be used – rather than searching the storage units for the one with the matching unique serial number.

To do this, we have used the previously introduced concept of the 'product-level EPC™', in which the range of bits which normally specify the serial number are all set to zero – so that under this bitmask operation, all Gillette razors of the same product type (but different serial numbers) result in the same 'product-level EPC™'. Setting the bit or digits to zero to represent a more generic class of items is a standard practice, which is used in the Domain Name Service [5] (to specify subnets or clusters of IP addresses) as well as in the United Nations Standard Products and Services Code (UNSPSC) [20] – to indicate broader levels of hierarchy. The zeroed bits or digits are needed as placeholders to maintain the addressing structure, so that the remaining digits appear in the expected positions, relative to the start of the number – and can be thought of as equivalent to wildcard characters which match any digits in that range of bits.

After the robot has packed the three required items into the gift box, the robot cell controller sends the `'SETITEMS'` command to our prototype PML server to specify that for the particular box EPC™, the following specific item EPCs™ have been placed in slots 1-3. In this case, the item EPCs™ include the serial number bits and are recorded as new `<node>` elements within the `<present>` block. The final PML data for the packed box in shown in Figure 8.

Figure 8: Phase 1 Prototype PML markup after packing a unique box – note that after packing we record the unique EPCs™ of the items used in the `<present>` block, to enable instance-level tracing and quality control. Note that the unique EPCs™ match with the corresponding product-level EPCs™ specified in the `<future>` section.

```
<?xml version="1.0" ?>
<node epc="0000000A1000001000000DDD">
<future>
<owner name="retailer">tesco</owner>
<desc>gillete gift pack for tesco summer sale</desc>
    <node epc="00000000100000D000000000" />
    <node epc="00000000100000A000000000" />
    <node epc="00000000100000B000000000" />
</future>

<present>
<owner name="manufacturer">gillette</owner>
<desc>gillette giftbox configuration type A</desc>
    <node epc="00000000100000D000000017" />
    <node epc="00000000100000A000000031" />
    <node epc="00000000100000B000000009" />
</present>
</node>
```

Finally, as the packed box passes through the quality control station, the pair of tag readers simultaneously read the EPCs™ of the box and the three items and check against the PML data whether:

1. **they are of the correct type** – i.e. match at the product-type level, as specified in the `<future>` block of the order/recipe.
2. **they are still the same unique items which were packed in the box** – i.e. match down to the serial number level , as recorded in the `<present>` block at the time of packing.

2.1.2 Phase 1 PML Server Implementation

For the Phase 1 demonstration, we decided to implement a prototype PML server using the Apache [21] webserver, since it would be relatively easy for the existing Visual Basic cell controller to communicate with it via a web-browser module, effectively by requesting web addresses (URLs) in which the URL encoded the command and arguments to be sent to the PML server for queries and updates.

Various URL handlers were written, specific to the operations of our robot packing cell. We identified the minimum command set for the queries and update functions from our basic Phase 1 PML server as listed in Table 1.

The URL handlers were written in ModPerl [22] as modules stored in the directory/etc/httpd/lib/perl/Apache. The Apache configuration file (/etc/httpd/conf/httpd.conf) was modified to include `<LocationMatch>` directives to send all requests containing particular command keywords to the appropriate ModPerl handlers.

Table 1: A minimal PML command set required for the Phase 1 packing cell Note that in Phase 2, we use a more generic set of PML queries together with XQL/Xpath expressions to refer to a specific node of the PML

QUERIES	
`IFEMPTY(boxepc)`	Given a box EPC™, return `'EMPTY'` or `'ITEMS'` depending on `PRESENT` state of box
`QUERYORDER(boxepc)`	Given a box EPC™, return a list of the three item types in the `FUTURE` order for the box
`QUERYITEMS(boxepc)`	Given a box EPC™, return a list of the three item IDs in the `PRESENT` state of the box
UPDATE COMMANDS	
`SETORDER(boxepc,item1epc,item2epc,item3epc)`	Given a box EPC™, set the three items for the `FUTURE` order of the box
`SETITEMS(boxepc,item1epc,item2epc,item3epc)`	Given a box EPC™, set the three items for the `PRESENT` state of the box

The URL handlers also used standard XML tools (XML::DOM – document object model and XML::XQL – extensible query language) to query and update information from the PML data.

The PML data was stored as a simple DB_File database consisting of a key/value lookup table (hash table), in which the key was the EPC™ and the value was the corresponding PML data. We used this approach for simplicity and because we wanted to show changes to the PML data alongside a graphical representation of the order in our user interface. Also, at the time of development, the PML schema was not well-defined – and a well-defined schema will assist with mapping [16] hierarchical XML data to fields within tables of a relational database such as SQL.

We also implemented a number of additional commands for communication with the graphical user interface and with the Savant™. However, from Phase 2, communications with both the Savant™ and the prototype PML server are now being implemented using the simple object access protocol (SOAP) [23].

2.2. Phase 2 – Towards a More Generic PML Server and Fulfilment of Batch Orders

In the Phase 2 demonstration, instead of placing customised orders for individual boxes, the new user interface allows various users (e.g. Wal-Mart, Tesco etc.) to place batch orders for gift boxes, where each order can consist of multiple sub-orders and each sub-order consists of a specified quantity of boxes of a specified type and configuration (items and their arrangement within the box). The user interface also allows the customer to indicate a priority for each sub-order, so that the holonic cell controller can give priority to rush orders and high priority orders.

The original Apache webserver which runs the user interface continues to operate on port 80, but all PML server functions are now handled via a SOAP interface, using port 81 as the proxy address, which redirects the SOAP request to the appropriate handler. The conduit to Savant™ provided by the old webserver-based PML server is now redundant, since the holonic cell controller, written in Java and JACK [24], communicates directly with Savant™ and the new prototype PML server via SOAP.

By analysing the functionality provided by the Phase 1 PML server, we could identify a restricted set of generic methods for the new PML server, making use of XPath/XQL expressions as parameters passed to these few generic methods to indicate which part of the PML data to query or update. (See Figure 4 for a reminder) The minimum set of proposed methods of the Phase 2 PML server are listed in Table 2:

Table 2: A more generic command set for querying our prototype PML server in the Phase 2 demonstration, consisting of a function to return the entire PML stored on the server for a given EPC, a function to read data from a specified PML node, and a function to write data values into a specified PML node, creating the necessary parent elements, where required. The 'read()' function also supports counting when the XQL/Xpath expression ends in /count().

QUERIES	
`getpml(epc)`	returns PML data corresponding to the specified EPC™ string based on local PML database lookup
`read(epc,xql)`	returns the stripped values from between tags or attribute values for nodes specified by the xql string from the PML data corresponding to the specified EPC™ string
`write(epc,xql,values)`	locates the PML data corresponding to the specified epc string, identifies the node values or attributes identified by the xql string and sets each in turn to the specified values – which may be a string array – or multi-line string with newline as delimiter

The methods listed in Table 2 are implemented as functions within a single Perl SOAP module, Order.pm located in /home/httpd/soap

A Perl-based SOAP daemon (using the SOAP::Lite module [25] from CPAN [26]) listens on port 81 and passes incoming SOAP requests to the appropriate SOAP module, such as Order.pm, depending on the SOAP object specified in the 'uri:' parameter of the SOAP message.

Figure 9 shows an example of the PML data we used for representing batch orders in the Phase 2 demonstration. Table 3 lists examples of the XPath/XQL expressions for accessing particular parts of the batch order data. Note that XQL/XPath also defines functions such as count(), value() and text().

Figure 9: Example PML markup for a batch order in Phase 2 of the robotic manufacturing demonstration. The order parameters are specified in the `<Future>` block, as one or more configurations, each of which specifies a box type and items required and the quantity of these per configuration – in this case 3. After packing, the items used will be recorded within corresponding configurations in the `<Present>` block.

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml:stylesheet href="../autoidss0.xsl" type="text/xsl" ?>
<BatchOrder>
    <Owner>
        <Role>manufacturer</Role>
        <Entity>Tesco</Entity>
    </Owner>

    <Date label="edited">0</Date>
    <Date label="deliverby">0</Date>
    <Future>
        <Config label="c1">
                <Msr q="1">3</Msr>
                <Price>2</Price>
                <Description>Gillette gift box</Description>
                <Date label="edited">1043060411000</Date>
                <Part label="box" epc="0000000A1000002000000000">
                <Part label="item1" epc="00000000100000A000000000" />
                        <Part label="item2" epc="00000000100000B000000000" />
                        <Part label="item3" epc="00000000100000D000000000" />
                </Part>
        </Config>
    </Future>
    <Present />
</BatchOrder>
```

Table 3

EXAMPLES OF XQL/XPATH EXPRESSIONS USED FOR ACCESSING PML DATA	
`/BatchOrder/Future/Config[ @label= 'c1'] /Part[ @label='box'] /Part[ @label= 'item2'] /@epc`	The EPC™ of item2 of configuration 'c1'
`/BatchOrder/Future/Config[ @label= 'c1'] /Msr[ @q='1'] /value()`	The number of boxes to be packed for configuration 'c1'
`/BatchOrder/Present/Config[ @label= 'c1'] /Part[ @label='box'][ \@epc= '0000000A1000002000000005'] /Part[ @label= 'item1'] /@epc`	Used to record the EPC™ of the item which was placed as 'item1' inside box EPC™ `0000000A1000002000000005` as partial fulfilment of configuration 'c1'

3. CONCLUSIONS

This paper outlines the proposed structure of a prototype PML server, and how this builds upon the existing product database held by a company. We have developed a practical prototype of a PML server, which supports a functioning robotic manufacturing cell, assisted by other aspects of Auto-ID technology (tags, readers, Savant™ etc.). In both phases of development, we identified the importance of being able to serve selected fragments of PML data, in order to return the value of a particular property, rather than returning the entire PML data held about a given object or EPC™ at a particular server. In the second phase of development we replaced our previous dedicated methods with a minimal set of operations (getpml, read, write) and identified Xpath/XQL expressions as a means of referring to a particular part of the PML data. Further practical prototyping of PML servers will continue in the Cambridge lab and elsewhere, to model co-ordination between multiple PML servers within a supply chain and integration with business information systems.

4. REFERENCES

1. **S. Sarma, D. Brock & K. Ashton, 2000, "The Networked Physical World – Proposals for Engineering The Next Generation of Computing, Commerce & Automatic Identification".**
   http://www.autoidcenter.org/research/MIT-AUTOID-WH-001.pdf.

2. **Oat Systems & MIT Auto-ID Center, 2002, "The Savant™ – Version 0.1 (Alpha)".**
   http://www.autoidcenter.org/research/MIT-AUTOID-TM-003.pdf.

3. **D. Brock, 2001, "The Electronic Product Code™ (EPC™) – A Naming Scheme for Physical Objects".**
   http://www.autoidcenter.org/research/MIT-AUTOID-WH-002.pdf.

4. **Oat Systems & MIT Auto-ID Center, 2002, "The Object Name Service (ONS) – Version 0.5 (Beta)".**
   http://www.autoidcenter.org/research/MIT-AUTOID-TM-004.pdf.

5. **Domain Name Service**
   For more information see links at http://www.dns.net/dnsrd/.

6. **D. Brock, 2001, "The Physical Markup Language (PML) – A Universal Language for Physical Objects".**
   http://www.autoidcenter.org/research/MIT-AUTOID-WH-003.pdf.

7. **D. Brock, T. Milne, Y. Kang & B. Lewis, 2001, "The Physical Markup Language".**
   http://www.autoidcenter.org/research/MIT-AUTOID-WH-005.pdf.

8. **D. McFarlane, 2002, "Auto-ID Based Control – an Overview".**
   http://www.autoidcenter.org/research/CAM-AUTOID-WH-004.pdf.

9. **S. Sarma, 2001, "Towards the 5¢ Tag".**
   http://www.autoidcenter.org/research/MIT-AUTOID-WH-006.pdf.

10. **M. Reynolds, J. Richards, S. Pathare, H. Tsai, Y. Maguire, R. Post, R. Pappu & B. Schoner, 2002, "Multi-Band, Low-Cost EPC™ Tag Reader".**
    http://www.autoidcenter.org/research/MIT-AUTOID-WH-012.pdf.

11. **XML Schema.**
    http://www.w3.org/XML/Schema.

12. **XML – extensible Markup Language.**
    http://www.w3.org/XML/.

13. **HTML – Hypertext Markup Language.**
    http://www.w3.org/Markup/.

14. **For details of XML markup languages for various sectors of industry.**
    http://web.mit.edu/mecheng/pml/standards.htm.

15. **Structured Query Language (SQL).**
    see also http://www.sql.org.

16. **Many commercial relational databases are developing translation layers for mapping between XML and relational database queries.**
For further examples of translation layers for interfacing between SQL and XML Schemas, see also: http://developer.netspective.com/xif.html and http://zsqlml.sourceforge.net .

17. **XPath – XML Path Language.**
http://www.w3.org/TR/xpath.

18. **XQL – XML Query Language.**
http://www.w3.org/TandS/QL/QL98/pp/xql.html.

19. **S. Hodges, A. Thorne, A. Garcia, J. Chirn, M. Harrison, D. McFarlane, 2002, "Auto-ID Based Control Demonstration Phase 1: Pick and Place Packing with Conventional Control ".**
http://www.autoidcenter.org/research/CAM-AUTOID-WH-006.pdf.

20. **United Nations Standard Products and Services Code – a hierarchical classification scheme for objects.**
http://www.unspsc.com.

21. **Apache webserver project.**
http://www.apache.org.

22. **ModPerl – Perl modules for Apache.**
http://perl.apache.org.

23. **Simple Object Access Protocol (SOAP).**
http://www.w3.org/TR/SOAP.

24. **JACK Intelligent Agents™ from Agent Oriented Software.**
http://www.agent-software.com.

25. **SOAP::Lite – a perl module for SOAP.**
http://www.soaplite.com.

26. **Comprehensive Perl Archive Network.**
http://www.cpan.org.

Designed and produced by Foxner. www.foxner.com