



Web Services- based Automation for the Control and Monitoring of Production Systems

Punnuluk Phaithoonbuathong

► To cite this version:

Punnuluk Phaithoonbuathong. Web Services- based Automation for the Control and Monitoring of Production Systems. International Journal of Computer Integrated Manufacturing, 2010, 23 (02), pp.126-145. 10.1080/09511920903440313 . hal-00558523

HAL Id: hal-00558523

<https://hal.science/hal-00558523v1>

Submitted on 22 Jan 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Web Services- based Automation for the Control and Monitoring of Production Systems

| | |
|-------------------------------|---|
| Journal: | <i>International Journal of Computer Integrated Manufacturing</i> |
| Manuscript ID: | TCIM-2009-IJCIM-0011.R2 |
| Manuscript Type: | Special Issue Paper |
| Date Submitted by the Author: | 15-Jul-2009 |
| Complete List of Authors: | Phaithoonbuathong, Punnuluk; Loughborough University, Mechanical and Manufacturing Engineering |
| Keywords: | AGILE MANUFACTURING, DISTRIBUTED MANUFACTURING CONTROL, COLLABORATIVE MANUFACTURING, MANUFACTURING CONTROL SYSTEMS, RECONFIGURABLE MANUFACTURING SYSTEMS, WEB-BASED MANUFACTURE |
| Keywords (user): | component-based design, Device Profile for Web Services |
| | |



International Journal of Computer Integrated Manufacturing, Vol. X, No. X, Month 200X, 000–000

Web Services- based Automation for the Control and Monitoring of Production Systems

Punnuluk Phaithoonbuathong^{*}, Radmehr Monfared, Thomas Kirkham, Robert Harrison, and Andrew West

Wolfson School of Mechanical and Manufacturing, Loughborough University, Leicestershire, UK

Autonomous and intelligent control devices within the context of factory automation are seen as an essential ingredient in making time and cost savings in factory automation environments. In moving to mass customisation scenarios where production lines are subjected to frequent changes and mix types of products, the agility and reconfigurability of automation systems is a prime requirement to support changes in manufacturing's lifecycles. In addition, intelligent functionalities including process monitoring, diagnostics and process reconfiguration are also desirable factors to facilitate an effective production unit with competitive costs and an ease of use and maintain. In this context, the adoption of Web Services on the distributed embedded control devices to enhance reconfigurability and integrability with supported manufacturing and business applications is proposed.

This paper demonstrates the use of Web Services (WS) both in building device control functionality of control components and business application integration. This WS approach presents the ability to integrate pervasive enterprise applications (e.g., process monitoring and planning systems) as well as the ability to reconfigure and manage lower level devices from higher manufacturing and business control levels through unifying Web Services interface and neutral Simple Object Access Protocol (SOAP) message communication between control systems and business applications.

Keywords: component-based design; Device Profile for Web Services (DPWS); distributed control system

1. Introduction

Traditional manufacturing automation systems are often constructed in a rigid, centralised, hierarchical, and also proprietary manner. These factors have contributed to implementation complexity, result in an often time-consuming and error-prone processes for automation system alteration and validation. These factors also have a direct impact on increasing the process ramp-up time and often lead to process performance degradation. Specifically, current automation systems are deemed to require experts in order to be commissioned and maintained. This is often due to complex and unstructured programming, and use of vendor specific technology. It is therefore difficult for end users to support and optimise their automation systems effectively (Phaithoonbuathong *et al.* 2007).

^{*}Corresponding author. Email: P.Phaithoonbuathong@lboro.ac.uk

It has been reported by Colombo (2004) that the centralised and sequential manufacturing planning, scheduling and control mechanism are increasingly being found insufficient for the current market environment. This is mainly due to a lack of agility in the control system to respond to changes from new product specification and dynamic variations in the product. In response to these rapid, continuous and unpredictable changes, organisations need to embrace more adaptive, collaborative, and responsive automation paradigms. A number of works have been proposed solutions toward this problem as outlined in section 4. Out of these paradigms developed in the recent years the concept of the agile and collaborative manufacturing approach as reported by several authors (Gunasekaran 1998, Colombo *et al.* 2004, Wang 2004) has gained considerable attention from many research institutions and organisations. Efforts to explore and develop a new manufacturing environment (Molina *et al.* 2005) in various domains such as shop floor automation control, enterprise control, and global control (e.g. the supply chain and outsourcing companies) are now evident. According to Büyüközkan (Büyüközkan *et al.* 2004), the key enabler of the agile and collaborative paradigm is to enable the easy integration of automation systems with other business entities (including marketing, engineering, product design, business process management and personnel) that support the lifecycle of the business. The approach aims to reduce the cost of product changeovers, time to market, machine down time, and machine ramp-up time through the utilisation of appropriate and effective solutions for the reconfigurable automation system as well as for the shop-floor to business system integration. In relation to this collaborative enterprise development, it was addressed by Weston (1999) that a fragmented and heterogeneous design of the industrial system has impeded interoperability among manufacturing and business entities. This problem has posed major difficulty and complexity in developing enterprise systems in this context.

Recently, the development of Service Orientated Architectures (SOA) is intrinsically linked to the growth of e-Businesses. Many current e-Business computing models rely on the presentation of applications and access to data by common standards based on Web Services. Using these services composite applications can be developed that span computing domains and organisational boundaries. For businesses, this added pervasiveness in terms of data management and utilisation has yielded greater accessibility and wider integratability to supply chains and business collaborators. It is not just e-Business systems are become more comprehensive in the “business-to-business” application level. A similar trend is now beginning to establish itself at the factory floor device level. Applying Web Services (WS) to define machine control components and how they interact as the neutral platform is attractive both to the manufacturers and enterprise system vendors. New engineering method for reconfiguration based on Web Services enabled technologies will save manufacturers time and resources in developing manufacturing systems for new products. Web services enabled control devices can facilitate device installation, reconfiguration and integration to higher control applications both in manufacturing and business contexts.

The implemented Web Services approach in this research is based on a discrete event-driven and control device interlocking machine application. The research aims to build the foundation that could be applied with further development to handle more complex applications (e.g. CNC, CMM machines). The authors’ work in this paper is in line with the ongoing works within the SOCRADES EU project framework and in collaboration with Ford Motor Company to address reliability, robustness and error recovery of the engine assembly line control systems. In this paper, a SOA capable of supporting reconfiguration of production lines and Enterprise level device management is presented and the results and findings of developments on a prototype system at Loughborough University are presented. This research has created an engineering environment for reconfigurable component-based control in a distributed environment that aims to support agile manufacturing systems.

The following section describes the modular design of the component-based automation architecture, reconfigurable manufacturing systems. The motivation and the related work in the field of collaborative and agile automation systems are presented in section 3 and 4 respectively. Section 5 describes the core implemented technologies and methodologies employed by Web Services which will be integrated into control devices. In section 6, the implementation of a Web Services- based distributed embedded controller is presented. Finally, the early results related to network performance and I/O reaction speeds are presented and the plan for future work and conclusion is discussed.

2. A modular component-based design approach

There is an earlier attempt proposed by McLean (McLean *et al.* 1982) to solve the problem in rigid manufacturing system by introducing the virtual cell to monitor, control and manage machine tools in distributed manufacturing environment. The cell function (reporting, scheduling, dispatching and monitoring) is defined as an abstract object, organised by the higher cell control, to enable dynamic changes of manufacturing process. Additionally, the early work at the authors' institution proposed by Weston (Weston *et al.* 1989) on modular design of distributed manipulator system demonstrated the decomposition and composition of machine functions for flexible design of the control system.

Previous work leading up to the research presented in this paper has been conducted on reconfiguration modular automation systems for powertrain assembly machines at Ford Motor Company. This research has applied a component-based design approach for the distributed automation system presented by Harrison (2004). The main characteristic of the proposed modular systems was the design of generic hardware and control software components that could be reconfigured to form new machine configurations to suit different production types (Harrison *et al.* 2004). The modular component-based architecture enables efficient machine build and re-use of designs in order to optimise the machine's lifecycle throughout supply chain partners.

As depicted in figure 1, the typical component-based (CB) layout of the assembly machine system for this research is built from various machine subsystems. These include hopper unit, buffering unit, processing index table unit, and handling arm unit, which are linked together via Ethernet communication systems to form the completed work process. These subsystems are constituted from sets of components which contain mechanical units, electrical units (sensors, actuator, I/O interface) and control software.

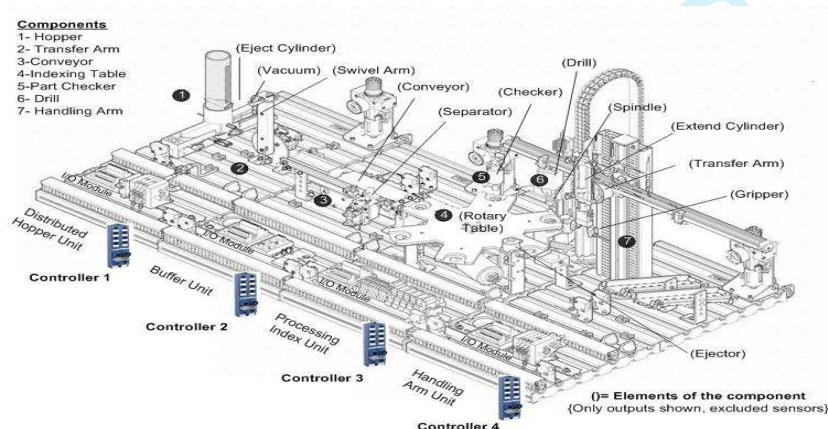
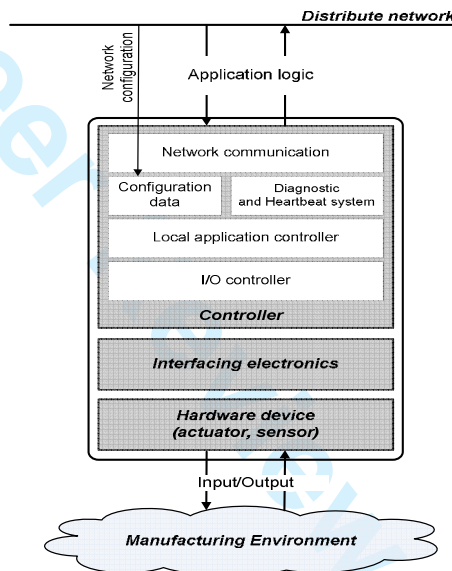


Figure 1. A distributed CB design of a powertrain assembly machine

In the previous research at Loughborough University, a machine component is defined with the constitution of: (1) controller unit which includes network communication, hardware configuration, and control application, (2) electronic interface, and (3) physical inputs and outputs. The physical view of the control component is defined in figure 2.

The required functionalities of the component for manufacturing tasks i.e. device network communication, operating mode, device behaviour, local control application logic and error information are encapsulated in the controller unit.

Within this CB design approach, the low-level control functionality has been encapsulated as a black box and exposed to the developers via the process engineering tool to define machine configuration data for the machine application.

Figure 2. Physical constituent of a component (based on Lee *et al.* 2004a)

In this distributed control application, a machine application (e.g., a manufacturing function) is a combination of distributed services on components for execution in a specific order. The driving state is achieved by formulating a behaviour model for each machine component. The reconfiguration of the process can therefore be oriented to the higher level alteration of the machine application/function design.

In the CB design architecture, these components are pre-built with the required functional and integration capabilities to enable effective composition into instances of machines based on the required specification of machine end-users. To provide the flexibility and reconfigurability, user applications are composed by a higher level engineering tool (process definition editor tool previously developed for the COMPAG project presented in Harrison *et al.* 2004). The development and reconfiguration of component-based automation systems at the abstract level allows a large number of I/O systems and communication functions to be realised by independently developed components.

In addition to the CB approach, the control system can be seen as a set of mechatronic components and each device is responsible for a basic operation, then the more complex manufacturing tasks can be created from the combination of basic components (Harrison *et al.* 2006). It can be observed that the complexity of the control system in relation to build and reconfiguration is reduced by managing the decomposition of the machine system accommodated by the process engineering tool.

3. Need for Web Services in automation

The CB approach is valued in enabling flexible and reconfigurable automation systems to more effectively support manufacturing lifecycles. However with the new perspective of manufacturing system toward agile and collaborative manufacturing system enabled by mass customisations as discussed by various research groups (Tseng *et al.* 1997, Gunasekaran 1998, Colombo *et al.* 2004, Harrison *et al.* 2006, Assen *et al.* 2000), the new automation system and integrated engineering approaches need to capture a broader view of the system integration in enterprise, supply-chain and machine and process control dimensions throughout the manufacturing lifecycle.

The utilisation of the CB approach within manufacturing is somewhat limited in terms of its ability to deliver the agile and collaborative engineering environment required. This is because integration from higher level systems to production lines is limited due to differing technologies and the complexity of integrating diverse automation peripherals. In addition, this integration is still reliant on proprietary solutions from different vendors preventing effective inter- and intra enterprise integration.

From this viewpoint, it has been discussed in Jammes and Smith (2005) and SOCRADES. EU project (2006) that the use of Service-Oriented Architectures (SOA) implemented through Web Service technologies can facilitate the adoption of a unifying technology for all level of the enterprise. Web Services are based on common communication standards such as HTTP, XML and SOAP, thus enabling homogenous linkage from the control levels to enterprise business processes. The use of SOA-Web Services technologies with the modular CB manufacturing approach can provide an open, standard approach for non vendor specific control systems as well as enabling seamless enterprise integration.

4. Related work

In recent years a number of projects have proposed open modular solutions suitable for manufacturing automation plants. *ESPRIT III OSACA* (Sperling 1997) in the area of Controls within Automation systems, proposed an object-oriented architecture to develop independent modular software structure within controls. The recent *ITEA SIRENA* project (Jammes and Smith 2005) has proposed a new approach using Web Services based on DPWS (Device Profile for Web Services standards) to support “plug-and-play” devices. In more recent projects, the *SOCRADES* project (SOCRADES.EU project 2006) is developing an open-platform for automation systems using this technology. The project has created the middleware technology presented in a SOA to enable interoperability between heterogeneous devices deployed on various platforms, as well as seamless integration into enterprise systems.

5. Web Services methodology and integration approach

This section presents the core technology of Web Services in the context of the assembly machine control system that is being applied to.

5.1 Serviced-Oriented Architecture (SOA)

Barry (2000) defined the conceptual definition of SOA as:

“A service-oriented architecture is essentially a connection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.”

where as Nickull (2005) describes SOA as:

“SOA is an architectural paradigm for components of a system and interactions or patterns between them. A component offers a service that waits in a state of readiness. Other components may invoke the service in compliance with a service contract.”

SOA- based applications are predominantly developed using Web Services, thus providing a standard information and communication infrastructure enabling data passing between field devices (Jammes and Smith 2005). Many opportunities exist to integrate Web Services enabled devices particularly in the domain of high-level enterprise and manufacturing systems. As discussed by Jammes (2005) and Machado (2006), SOA can be seen to have evolved the design of control systems into distributed application paradigms, and ubiquitous computing environments to enable flexible, reusable, reconfigurable manufacturing systems based on self-reliant, interconnected smart embedded devices.

5.1.1 Web Services and SOA enterprise integration framework

Extending the SOA focused Web Services approach to low level automation devices has been proposed by a number of researches (Harrison *et al.* 2004, Jammes and Smith 2005, Machado and Mitmann 2006, Harrison *et al.* 2006). The attraction from a business perspective is the inclusion of plant activity into enterprise computing applications, to facilitate work synchronisation between the high and low level applications as this concept was early introduced by McLean (1982).

In this context, a simplified manufacturing model of SOA and Web Services derived from Nickull (2005), Hung *et al.* (2005), Machado and Mitmann (2006) is illustrated in figure 3. The basic principle of enterprise integration and communication based on a consumer-producer model is described below.

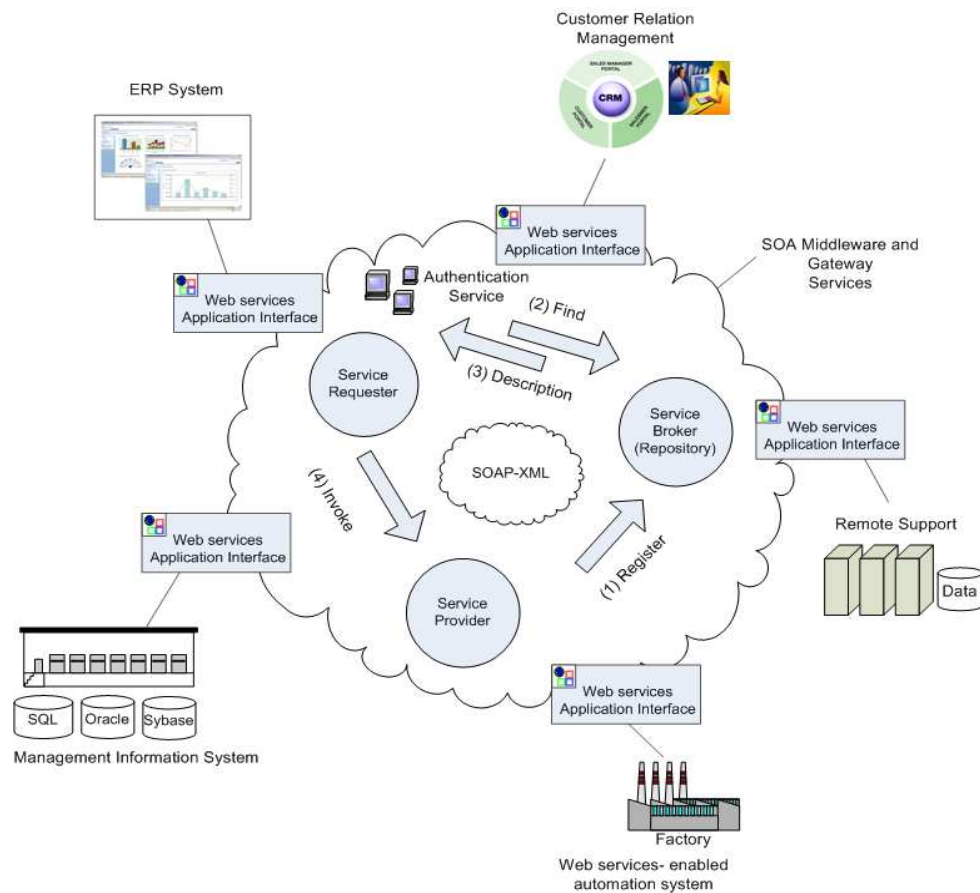


Figure 3. SOA-Web Services basic integration framework

- (1) The service provider publishes (produces) the available services to the services broker which is a repository for registered applications from producers.
- (2) The consumer finds the provided services from the broker.
- (3) The broker returns the service descriptions and policies of how to interact with the required services.
- (4) The consumer, then, invokes the remote service on a service provider.

Based on this model, Web Services ensure distributed interoperability across platforms, programming languages and applications that enable end-users to operate and maintain the supplied systems easily through SOAP, WSDL and UDDI which details as described in *Trifa et al. (2008)*, *Liu et al. (2008)*, *Yu and Chen (2003)*, *Graham et al. (2004)*, *W3C (2004)*, *Jammes et al. (2007)*, *Cachapa et al. 2007*, and *Boritz and Gyun (2005)*. Data transfer between distributed components is central to Web Services and is enabled via the use of the SOAP standard, and various XML meta-data formats (*Hung et al. 2005*).

However central to our application of Web Services-based automation system, an approach to the development of a Web Services enabled control device is needed in the SOA integration framework. The approach is not only considering the solution for enterprise integration but also the need of process reconfigurability and delivered performance of the control system to meet end user requirements. The implementation, demonstration and assessment works will be presented in this paper.

5.2 Devices Profile for Web Services (DPWS)

In the design of WS for a connecting device, a web capability has driven down to the low level controllers of control systems. The core network capability has been implemented with the standard and well defined protocol stack- Device Profile for Web Services (DPWS).

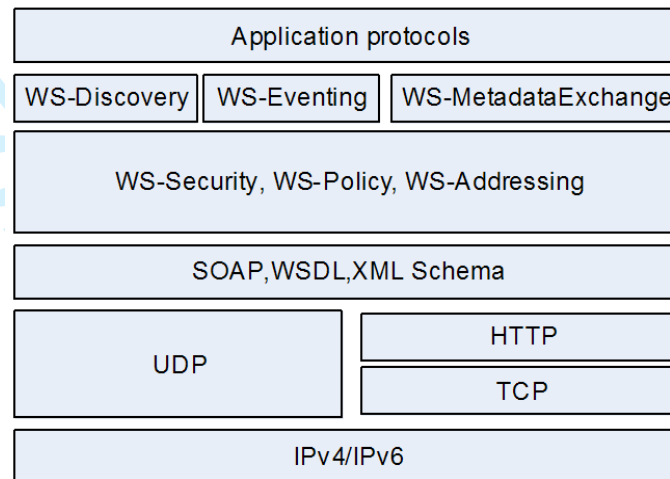


Figure 4. Devices Profile for Web Services (DPWS) protocol stack (Jammes and Smith 2005)

As presented in figure 4, the full descriptions of the Web Services protocol stack are reported by other research works in Jammes and Smith (2005), Barry (2000). The introduction of DPWS paves the way for the usage of a unifying technology base, via Web Services, across entire heterogeneous enterprise applications, from the sensor/ actuator level up to ERP/MES level.

In order to enable Web Services- based automation systems, the typical device logical functionality of the control application, such as setting outputs ON/OFF or reading sensors status of the component, is mapped to the Web Services logical function presented by the DPWS call function via the creation of DPWS interfaces on devices (referred to figure 7 for implemented codes) that enable devices to devices communications for state information exchange, and devices to control level and business application integration for services invocation and status monitoring.

In the authors' research, the implementation of DPWS protocol stack for control devices was developed in C language. An embedded microcontroller hosts the DPWS stack. The embedded controller in this research is an ARM9 based device (for information, please refer to Schneider Electric website on *Advantys FTB: System user guide*) developed and provided by the project collaborator, Schneider Electric, suitable for use in an industrial automation environment. The DPWS as well as its associated machine control applications are built and uploaded to ARM based control devices by mean of the ARM development suite tool. In addition to the control operation, the real-time operating system and the TCP/IP stack are ported to the embedded device providing the core control functionality (e.g., control thread, networking and control task scheduling). The software architecture for components is shown schematically in figure 5.

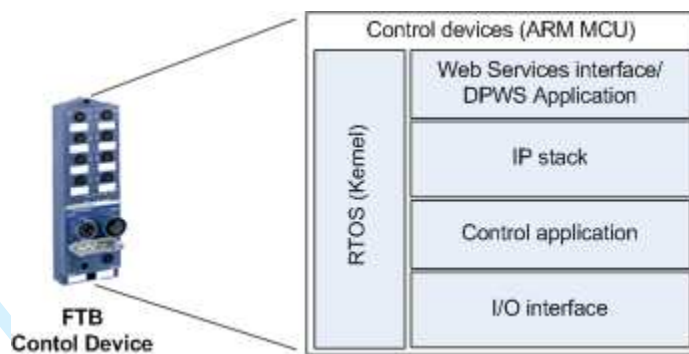


Figure 5. WS component software architecture

In our approach, predefined component functionality is presented by services to compose manufacturing tasks at the higher level in support with process management engineering tools. The hard coding of device behaviours (local control application) implemented by the component builder is encapsulated at the low level application. The alteration of the process functionality (e.g., process workflow and adding new machine components) is only done at the service level without changing the low level device code that makes system more flexible and agile.

5.3 Web Services integration

Based on the previous work on the component- based (CB) design approach for agile automation completed at MSI research institute, Loughborough University (Harrison 2004, Lee *et al.* 2004a and 2004b), a machine component has been defined with the I/O functionality (device state behaviour) of machine elements (sensors and actuators). The component is contained with its own independent control application. The operation of these distributed components is based on their defined interlocking sequence. The work has provided the automation framework for implementation of the reusable and reconfigurable control system described here.

In order to create an effectively integrated Web Services capability on the embedded controller devices in the SOA architecture, the development of the Web Services enabled device has complemented the CB approach with the capability to build machine components, which provides a manufacturing function (e.g., device execution, device information, and present device working state) as services to the business application integration through interface of service references.

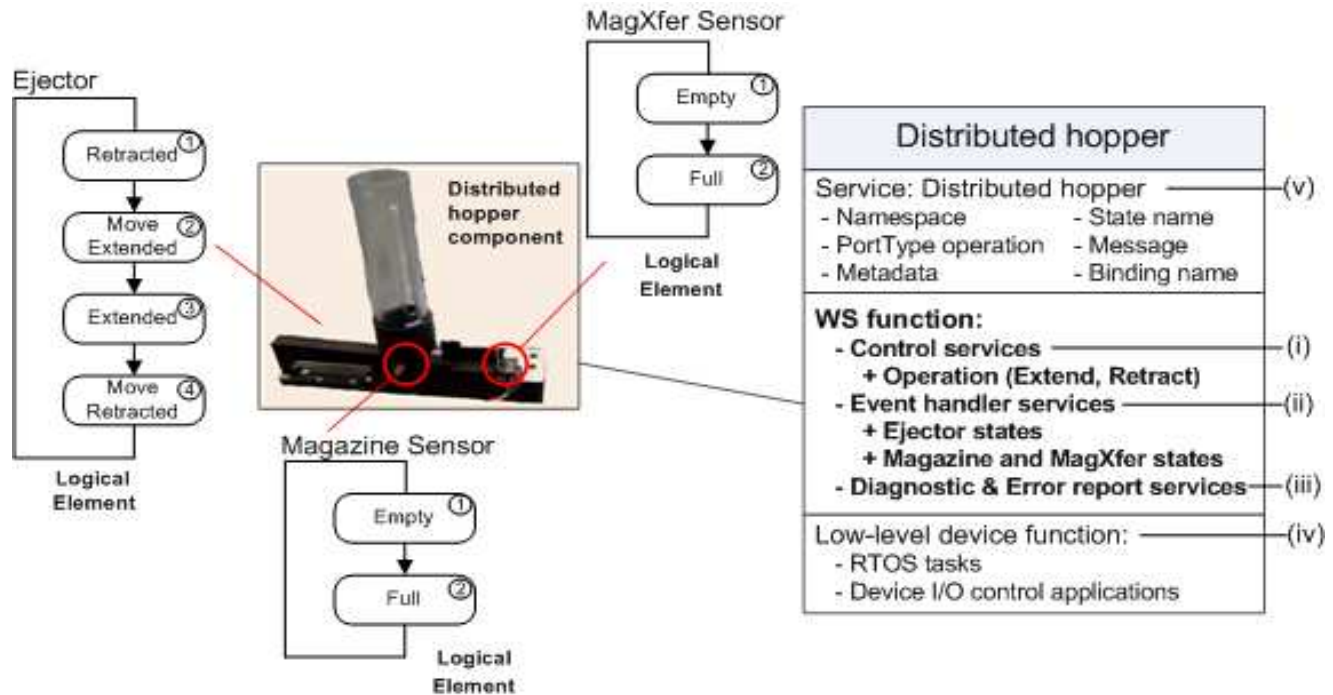


Figure 6. A Web Service- based design component

The novel implementation of Web Services-based component is shown in figure 6 as an instance of building a Web Services machine component. The hopper component is constituted with elements: Ejector actuator, Magazine sensor and MagazineXfer sensor which each contains their own state behaviour. The hopper component is represented by the Web Services component (described by WSDL- figure 7/a) which includes the information of the component based on contained elements such as Component name, Namespace location (reference web_location by service domain name), Binding port type (a collection of an abstract set of element operations and message exchange methods), and Metadata (component descriptions) required for the devices allocation on the network.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions name="Distributor"
  targetNamespace="http://www.soda-itea2.org/MSI_Lboro/Demonstrator/Distributor"
  >
  <wsdl:types>
    <xsd:simpleType name="distributorAction">
      <xsd:enumeration value="MoveRetracted"/>
      <xsd:enumeration value="MoveExtended"/> [see figure 7/b-(vi)]
    </xsd:simpleType>
  </wsdl:types>
  <wsdl:message name="distributorCmdRequest">
    <wsdl:part name="distributorAction" element="tns:distributorActionElement"/>
  </wsdl:message>
  <wsdl:message name="distributorNotifyMsg">
    <wsdl:part name="distributorState" element="tns:distributorNotifyElement"/>
  </wsdl:message>
  <wsdl:portType name="DistributorPortType">
    <wsdl:operation name="distributorCmd">
      <wsdl:input message="distributorCmdRequest"/>
      <wsdl:output message="distributorNotifyMsg"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="distributorPortType" type="tns:DistributorPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="distributorCmd">
      <wsdl:input message="distributorCmdRequest"/>
      <wsdl:output message="distributorNotifyMsg"/>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

a) WSDL descriptions of the component (figure 6- (v))

```

/*-----HANDLING ARM OPERATION COMMAND-----*/
int __dst_distributerCmd(struct dpws* dpws, enum dst_distributerAction dst_distributerActionElement,
                        struct __dst_ResponseElement *dst_ResponseElement)
{
    //Message received acknowledgement
    NumOrder++;
    dst_ResponseElement->OrderID = NumOrder;
    dst_ResultElement.OrderID = hla_ResponseElement->OrderID;
    dst_ResultElement.allRequestStatus = dst_RequestStatus__ACK;
    dpws_notify__dst_distributerCmdAck(&pState->dpws, pState->endpointRef0, &dst_ResultElement);
    switch (dst_distributerActionElement)
    {
        case dst_distributerAction_MoveExtended: -----(vi)
            MoveExtend(); //Setting output pins -----[see figure 7/c-(vii)]
            //Command acknowledgement
            dst_ResponseElement->allOperationResultStatus = dst_OperationResultStatus__DONE;
            dpws_end(&pState->dpws);
            break;
        .....other operation commands
        case dst_distributerAction__Reset:
            Shutdown(); //Cleaning and stopping DPWS and I/O routines
            //Command acknowledgement
            dst_ResponseElement->allOperationResultStatus = dst_OperationResultStatus__DONE;
            dpws_end(&pState->dpws);
            break;
        default:
            //Command failure
            dst_ResponseElement->allOperationResultStatus = dst_OperationResultStatus__FAILURE;
            dpws_end(&pState->dpws);
            break;
    }
    return SOAP_OK;
}

```

b) DPWS command interfaces to the device I/O operations (figure 6- (i) and (iii))

```

/*-----DEVICE I/O OPERATION-----*/
int MoveExtend () -----(vii)
{
    trGPIO_Set(8, FALSE); //channel 08 Retracted pos
    trGPIO_Set(9, TRUE); //channel 09 Extended pos
    /*Reading input sensor*/
    do
    {
        sensor1=trGPIO_Get(1); //channel 01 At retracted sensor
        sensor2=trGPIO_Get(2); //channel 02 At extended sensor
    }while(!(sensor1==FALSE&sensor2==TRUE));

    return 0;
}

```

c) Device I/O operations (figure 6- (iv))

```

/*-----DPWS DEVICE STATE NOTIFICATION-----*/
int dpwsDistributerNotify(DistributerStatus1) -----(viii)
{
    switch (DistributerStatus1)
    {
        case Extended:
            distributerNotifyElement.distributerStatus = dst__distributerStatus__Ejector_Extended;
            break;
        .....other cases
        case EJECTOR_ERROR:
            distributerNotifyElement.distributerStatus = dst__distributerStatus__EJECTOR_ERROR;
            break;
    }
    dpws_notify__dst_distributerNotifyEvent(&event0.dpws, event0.endpointRef0, &distributerNotifyElement);
    return 0;
}

```

d) The I/O device scanning interfaces to DPWS notification (figure 6- (ii) and (iii))

```

/*-----Detecting IO changes -----*/
int IOSPY(void)
{
    ...Initialising initial Distribute state
    do
    {
        ...Reading sensors state
        /*PARSING HANDLING ARM STATE*/
        if (sensor1==FALSE&sensor2==TRUE)
        {
            DistributerStatus1= Extended;
        }
        else if(sensor1==TRUE&sensor2==FALSE)
        {
            ...
        }
        .....conditions
        else
        {
            //State unknown (error)
            DistributerStatus1= EJECTOR_ERROR;
            //Fail safe routine enabled
            ...Safely releases a work piece and return to initial position
        }
    }
    /*-----DPWS Notification-----*/
    if (DistributerStatus1!= Pre_ DistributerStatus1)
    {
        ...state change detected
        dpwsDistributerNotify(DistributerStatus1); -----[see figure 7/d-(viii)]
    }
    ...
}while(1);
return 0;
}

```

e) Device I/O state monitoring and fault handling routine (figure 6- (iv))

Figure 7. Sample fragment codes of WSDL, DPWS component and Device I/O interfaces

In respect to device operations, the operations of the component are encapsulated by DPWS call references. In simple terms, these call references are concerned with services (operations) commands, state propagation (element states) of the event state change notification, and operating modes (Auto/Manual). These operations are multitasking handled by the RTOS. The implemented Web Services interfaces and I/O operations of the control system are shown in figure 7/b-7/e. A Web Service component is defined by a WSDL (Web Services Description Language- figure 7/a) file that defines the component specification, a network service, and DPWS interfaces for device programming in a XML language structure. In building the Web Services control and business application integration, the WSDL file is used for the generation of interface files for a client- server service invocation (stub and skeleton files) by the DPWS toolkit (SODA-ITEA Consortium 2007).

In the process of building a machine application enabled by the Web Service components in an event-driven control system, components are aggregated into the complete control system according to defined finite state transition conditions. To demonstrate the viability of Web Services in automation, the movement of Web Services control elements is managed by the PC based upon its control of finite state transition conditions (control logic) running in the DPWS application, which invokes/calls the embedded services provided by the component. The following section presents the service aggregation methodology for a machine application in the control system.

5.4 Service orchestration methodology

Service aggregation allows the formation of complex and business centric composite services/applications which are composed by often simpler and elementary services. The manufacture of products can be represented by a single abstract service which contains the appropriate stripped down statements for the invocation of the required sub-services (Bepperling *et al.* 2006). Production steps are represented in the behaviour of the service functions/operations that would be tied to real configuration in the control device. That means the process orientation of control levels is dependant on understandings of the lower level processes. The execution / reconfiguration of a production line can be seen as a wider application and the device level services in the SOA (Services Oriented Architecture) constitute this wider application.

Central to the success of the applications execution is the composition of the services in terms of execution order. The element functionality at the device level needs to be mapped and captured to the device management and self organisation Web Services- based application. Particularly in this research, this process of composing atomic components' services, which their execution sequences are initiated by the control logic operated outside the control device boundary, is known as "Service orchestration".

Service functionality of a device in the machine application is defined as a service to represent a more abstract production operation which can be created by combining simpler production operations. This process is referred to as orchestration and is achieved through the combination of these services to create the process application (manufacturing task). From a manufacturing system behavioural perspective, the process can be formed by defining a set of internal and external state conditions of distributed components in the control system.

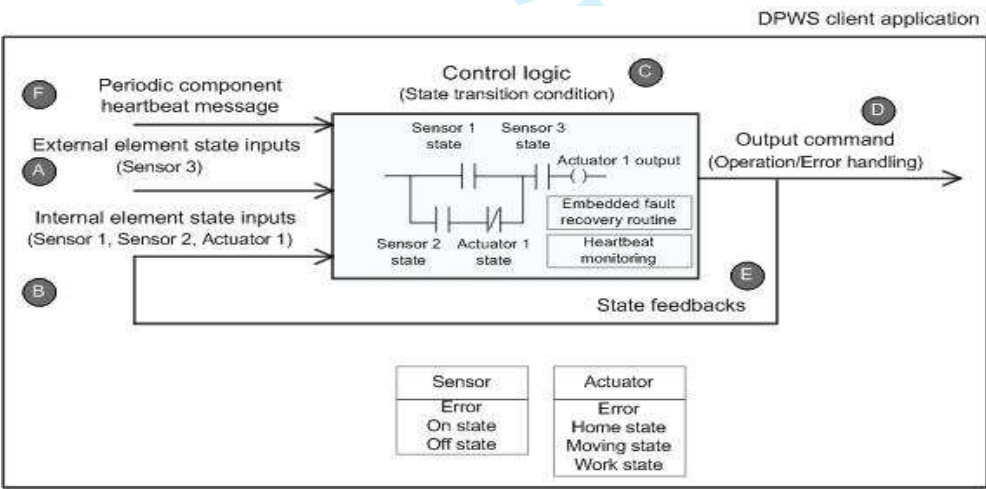


Figure 8. Component state transition diagram

As illustrated in figure 8, the state transition of components is a result of current states of internal and external elements, including the error state, in control logic applications which command the output of the distributed components hosted on controller devices. The simplified example of the control logic can be schematically presented by the ladder logic schema which has defined the component state transition condition through the

use of Boolean logic of input events in the discrete manufacturing environment. In this case, internal element state inputs are local state variables of the component and external element state inputs are state variables of other associated components (e.g. interlocking components). These variables change their state in respect with the movement of actuators (e.g., state feedbacks).

The composition of this component state transition has formed the machine sequences that determine the build of a machine application and a process work flow. In the machine operation of components, the component service is dictated by the control logic application known as “Service Orchestration Engine” necessary to orchestrate the service on the components to achieve desired manufacturing tasks.

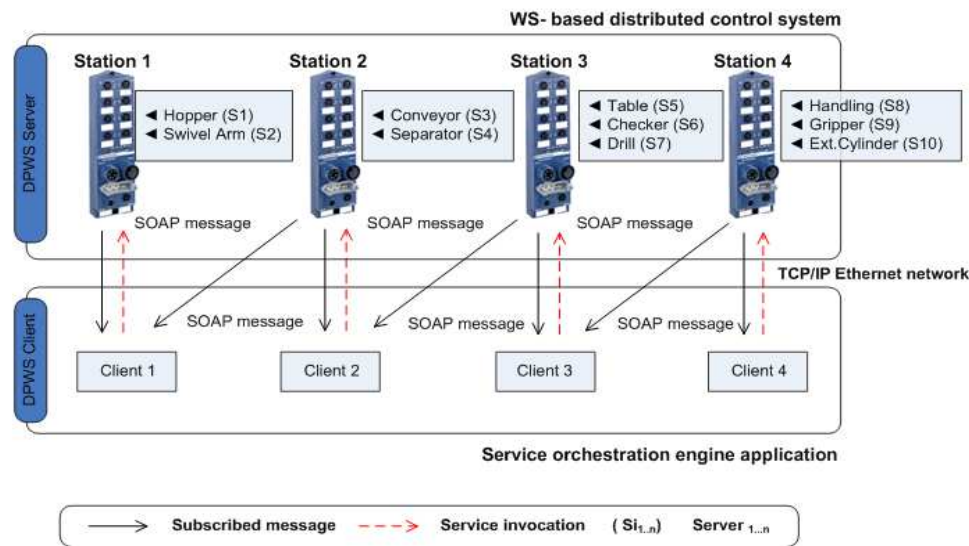


Figure 9. A PC- based orchestration engine application

In the distributed Web Services- based control system implementation, the distributed control devices host the service of components (e.g. hopper, and swivel arm component) which provide the Web Services operation and element state propagation functionality to the service orchestration engines based on a publish and subscribe approach in an event- driven architecture. As shown in figure 9, there are 4 service orchestration engines running as a client (client 1, 2, 3, and 4). Each one is responsible for controlling each specific station hosted by the servers (server 1- server10) of the components to perform manufacturing tasks in the control system.

For the control system operation, each client (via the service orchestration engine) subscribes to the specific server on the start up for the published state information of components required by the control logic application (C in figure 8) as presented earlier in this section. When any I/O (sensor and actuator) state changes occur, new I/O states are published to subscribers- clients (A, B and E in figure 8). The component operation (i.e., extend or retract the ejector element of the hopper component) as shown by D in figure 8 is based upon the state transition behaviour/logic of controlled elements. It should be noted that the movement of the control device has resulted in the I/O state changes of associated sensor and actuator elements which are propagated back to the clients for the new control event. This process of the service orchestration cycle runs repeatedly by the servers and the clients to perform the complete cycle of the machine application.

In addition to the system reliability in concern with safety of the operation on machine components, the heartbeat monitoring system and fault recovery routine (see //Fail safe routine in figure 7/e) have been implemented within the control system. In case of error occurrence on the device detected by cooperated sensors during conditioning process (e.g. EJECTOR_ERROR in figure 7/e), the pre-defined fail safe routine will be enabled automatically to safely recover from the hazard condition and then it reports the error state condition to the responsible client application to handle the fault by sending the DPWS reset command to shutdown the fail device. The periodic heartbeat message (F in figure 8) sent by the components will be used to monitor and ensure that all the control device operations (DPWS applications and I/O functions) are functioning normally. The heartbeat system is enabled by the DPWS function using a device state notification method. This function is planned to implement in the future work.

6. Implementation of the Web Services automation


The FTB (Field Terminal Block) control device which is industrially packaged to IP 67 (please refer to MeDiSol™ website for specifications), provides the processing capability local I/O and a standard embedded Ethernet connection.

As illustrated schematically in figure 5, the software structure of control devices or components is constructed with the layer of Web Services application at the top. The component also requires a suitable RTOS and IP stack laid underneath to handle the DPWS task with TCP/IP communication in order to perform the real world manufacturing tasks. The RTOS layer is responsible for the real-time task multitask scheduling in the control system. However, the research scope here does not include the examination of the methodology and techniques for control tasks scheduling. For information on this topic, please refer to Barry (2003).

Based on this architecture, encapsulated services of devices have been implemented using the DPWS toolkit provided by Schneider Electric to generate the C stub and skeleton files from the component WSDL file so as to build component services on the embedded FTB devices and service invocators (for orchestration). On the FTB, each service is interfaced to the actual I/O operation of components defined in the local control application layer. This DPWS component code is implemented with Realview debugger tool and ARM suite development platform (please refer to ARM® website for information).

[The DPWS invoked command]

```
dpws_call_dst_distributorCmd(&Distributor, ERPDistributor, NULL, dst__distributorAction__MoveExtended, &ResponseDistributor)
```



```
POST /13814002-1dd2-11b2-bc3d-0040af000032
HTTP/1.1..Host:150.1.0.102:9882..User-Agent: gSOAP/2.7..
Content-Type: application/soap+xml; charset=utf-8..Content-Length: 773..Connection: close....

<?xml version="1.0" encoding="UTF-8"?>.
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:blt="http://www.soda-itea2.org/MSI-Lboro/Demonstrator/Distributor">
  <SOAP-ENV:Header>
    <wsa:To>http://150.1.0.102:9882/13814002-1dd2-11b2-bc3d-0040af000032</wsa:To>
    <wsa:Action>http://www.soda-itea2.org/MSI-Lboro/Demonstrator/Distributor/distributorCmdRequest</wsa:Action>
    <wsa:MessageID>urn:uuid:41c616ba-3e14-11dd-8430-001c42935fe4</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
    </wsa:ReplyTo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <dst:distributorCmd>
      <dst:DistributorAction>MoveExtended</dst:DistributorAction> [see figure 7/d-(vi)]
    </dst:distributorCmd>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 10. Output SOAP message of a DPWS invoked command

The services orchestration engine running the component state transition logic (C in figure 8) is implemented with both C and Java language to investigate interoperability of the implemented DPWS control platform. The engine is used to manipulate the action on embedded controllers (FTBs) by sending out the DPWS invocation command in SOAP message (figure 10) format through Ethernet to the target component using XML transport.

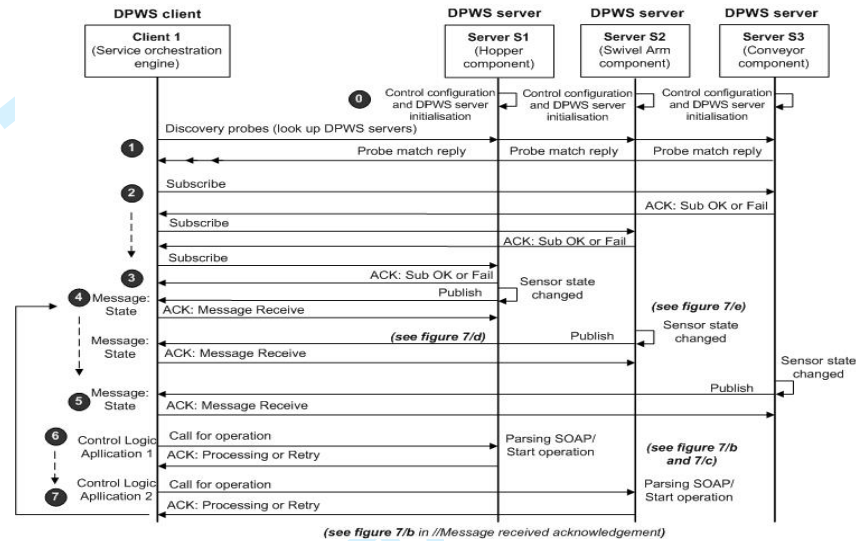


Figure 11. Test rig WS control system use case

The Web Services- based automation system is implemented with the DPWS on the distributed FTB devices executed by the distributed DPWS clients on PC- based orchestration of each station. As shown in figure 11, the Web Services application of the station 1 is demonstrated.

- 0: Server applications (S1, S2 and S3) start up; control devices initialise hardware/software configuration and DPWS applications of components.
- 1: The client application (running DPWS utility) sends probe messages to discover interest components hosted by server applications; the match component returns the message with the address. The client application keeps the location of component as an endpoint reference based on Universally Unique Identifier (UUID) format (i.e. urn: uuid: 13814002-1dd2-11b2-bc3d-0040af000032) in the memory for the DPWS application, such as the service subscription, and invocation.
- 2-3: The client (client 1) of the station 1 which contained 2 components subscribes to the DPWS components of station 1 (Hopper-S1 and Swivel drive-S2) and station 2 (Conveyor-S3) for the contained element states used by the service orchestration engine. The subscriber (the client) location/address is saved in the device registry handled by WS-Eventing.
- 4-5: During the machine operation, when the element has changed its state (i.e. work piece present), it publishes the state information to the client 1. The message is sent to the receiver addressed (allocated at the subscription process) by Uniform Resource Identifier (URI) scheme (i.e. IP: Port/UUID→ http://150.1.0.201:8873/158cb99e-693b-11dd-8abd-001c42935fe4) in the SOAP Header.
- 6-7: Orchestration engine on the client 1 aggregates the action according to the defined state transition behaviour. Like state notification method, the addressing scheme for the service invocation is provided by URI scheme. Please see figure 10 for a sample of UUID format and URI scheme.

In this application, having completed step 1 and 2-3, only step 4-5 and 6-7 have been run repeatedly through out a machine cycle. To ensure the system robustness in term of sending and receiving the SOAP message, the acknowledgment of the DPWS operation receipt has been implemented in addition to a TCP packet synchronisation (see TCP/IP networking and DPWS transaction time in the next section) to guarantee that every DPWS message has been successfully received at application level. For the implementation, the incremental order ID variable has been defined in the DPWS application to identify and keep track every message received by the components (please see //Message received acknowledgement in figure 7). The RTOS timer task on the sender (client) will be initiated and expected response from the receiver within a time limit. When the client receives this message, it notifies the sender with the SOAP message (i.e., ACK). In this case, missing acknowledge messages will result in a timeout after 50 milliseconds to raise a suitable error. However, this acknowledgment process has been currently implemented on only one side from the server to client application. Acknowledge in both ways will be next implemented.

In addition to the process control and visualisation, the system HMI provides an environment to allow users to collate data from and send command to distributed components in the system via a simple browser interface. For the visualisation, the HMI application interacts with the test rig through the service orchestration engine where live data of components is gathered. The data then has been populated from a state publication utility (built in the service orchestration engine) to the HMI application. For the system control, the HMI application is able to invoke the service on Web Services components for a machine operation via interface in the service orchestration engine that sends the SOAP command message to the component (as shown in figure 9 and 11).

7. Testing and preliminarily result

In this section, Component-based design of intelligent devices as exhibited in section 6 will be preliminarily evaluated in the real automation environment to measure the performance in term of communication speed and response time. In addition, the reconfiguration of the test rig process is determined to assess the degree of complexity in term of activities involved in reconfiguring component configurations, and control logics. An assessment on the business application integrability of the approach is also presented.

7.1 TCP/IP networking approach and DPWS transaction time

Based on our implemented Web Services control system, the protocol analyser has been installed to capture the network performance during the test rig operation. These analysis data (shown in Table 1 and 2) are average values derived from a measurement of around 2,000 packets out of 10 cumulated experiments. In this Web Services environment, present SOAP messages of the DPWS operations are state information, service call functions, discovery probe and metadata return. These SOAP messages packet size are 750 bytes up to a maximum TCP/IP packet size of 1514 bytes. In the DPWS service invocation (Table 1) and the device state notification (Table 2), there are 9 packets sent and received between Host A (FTB device as a server) and Host B (Orchestration engine application as a client) for each of the DPWS operation. According to a TCP byte- oriented sequencing protocol (InetDaemon Enterprise 1996a and 1996b, Bentham 2000), the analysis of packet synchronisation as shown in table 1 and 2 is included the TCP connection (3-way handshake) for a connection negotiation, the SOAP message, and a connection synchronisation and termination. In order to measure the DPWS processing (marshalling and demarshalling) time on the embedded device, the timer in the resolution of microsecond has been initiated in the DPWS application. The DPWS processing time on the embedded ARM 966 core control device with 96M Hz CPU speed is around 8 to 9 milliseconds.

Table 1 Packet synchronisation for the DPWS command invocation from Host A (client) to Host B (server)

| Packet No. | Packet size (Bytes) | Data load (Bytes) | 0-Initial packet received time (ms) | Time different between packets (ms) | Event | Seq No* | Ack No* | Diagram | |
|------------|---------------------|-------------------|-------------------------------------|-------------------------------------|---|------------------------------|---------|---------|--------|
| | | | | | | | | HOST A | HOST B |
| 1 | 78 | 0 | 0 | 0 | Host A sends a TCP synchronize packet to Host B to initiate a connection | 3976 | 0 | send | t |
| 2 | 60 | 0 | 0.37 | 0.37 | Host B replies Host A to acknowledge the request on the connection | 6001 | 3977 | recv | t+1 |
| 3 | 60 | 0 | 1.07 | 0.70 | Host A acknowledges Host B; The connection between A-B is established | 3977 | 6002 | send | t+2 |
| 4 | 969 | 915 | 1.082 | 0.012 | Host A sends a DPWS message (SOAP message) to Host B | 3977 0; Data load 915; | 6002 | send | t+3 |
| 5 | 60 | 0 | 1.086 | 0.004 | Host A sends another packet to Host B as the one to be acknowledged and also a request to close the connection from Host A to B | 4892 | 6002 | send | t+4 |
| 6 | 60 | 0 | 1.874 | 0.788 | Message received; Host B acknowledges Host A | 6002 | 4893 | recv | t+5 |
| 7 | 810 | 756 | 10.912 | 9.038 | Host B sends a replied DPWS message to Host A and confirm the connection closed from Host A to B (B to A still active) | 6002 756; Data load | 4893 | send | t+6 |
| 8 | 60 | 0 | 10.919 | 0.007 | Host B sends a packet to Host A that requests a termination of the connection from Host B to A | 6758 | 4893 | send | t+7 |
| 9 | 60 | 0 | 13.536 | 2.617 | Message received; Host A acknowledges Host B to terminate the connection from Host B to A | 4893 | 6759 | recv | t+8 |
| | | | | | Message sending/receiving is completed with both connections between Host A and B are closed to start a new TCP connection | 8603 | 0 | send | t+9 |

Table 2 Packet synchronisation for the DPWS message publication from Host B (server) to Host A (client)

| Packet No. | Packet size (Bytes) | Data load (Bytes) | 0-Initial packet received time (ms) | Time different between packets (ms) | Event | Seq No* | Ack No* | Diagram | |
|------------|---------------------|-------------------|-------------------------------------|-------------------------------------|---|------------------------|---------|---------|--------|
| | | | | | | | | HOST B | HOST A |
| 1 | 60 | 0 | 0 | 0 | Host B sends a TCP synchronize packet to Host A to initiate a connection | 4001 | 0 | send | t |
| 2 | 60 | 0 | 0.402 | 0.402 | Host A replies Host B to acknowledge the request on the connection | 0225 | 4002 | recv | t+1 |
| 3 | 60 | 0 | 0.726 | 0.324 | Host B sends an acknowledgment to Host A; The connection is established | 4002 | 0226 | send | t+2 |
| 4 | 794 | 740 | 2.852 | 2.126 | Host B sends a SOAP message to Host A | 4002 740; Data load | 0226 | send | t+3 |
| 5 | 60 | 0 | 2.856 | 0.005 | Host B sends another packet to Host A as the one to be acknowledged and also a request to close the connection from Host B to A | 4742 | 0226 | send | t+4 |
| 6 | 60 | 0 | 3.349 | 0.493 | Message received; Host A acknowledges Host B | 0226 | 4743 | recv | t+5 |
| 7 | 168 | 114 | 5.645 | 2.296 | Host A sends another packet to Host B to confirm the closing connection from Host B to A | 0226 114; Data load | 4743 | send | t+6 |
| 8 | 60 | 0 | 5.657 | 0.012 | Host A sends a packet to Host B that requests a termination of the connection from Host A to B | 0340 | 4743 | send | t+7 |
| 9 | 60 | 0 | 6.042 | 0.385 | Message received; Host B acknowledges Host A to terminate the connection from Host A to B | 4743 | 0 | recv | t+8 |
| | | | | | Message send/receive is completed with both connections are closed for Host A and B to start the new TCP connection | 8001 | 0 | send | t+9 |

* For the simplicity of demonstrating the packet synchronisation, packet sequence number (Seq No.) and acknowledge number (Ack No.) are only displayed by the last 4 digits of a full number.

In the implemented WS control system, the I/O response time (the delay between the occurrence of an input event and the corresponding of an output event as described by Denis (2007) is accounted by the summary delivery time of:

1. The packet time on a network,
2. The DPWS processing time
(2.1. DPWS encoding and decoding on the FTB control device and 2.2. Orchestration Engine),
3. The local I/O processing time on the FTB and,
4. Processing time on the orchestration engine.

In the experiment, separated analysis works have been carried out on the server and the client side to measure consumption time of each task for the control operation. In this case, Table 1 and 2 have covered the time analysis on the server tasks (included 1, 2.1, and 3). The time analysis on the orchestration engine (client) application has been captured by setting stopwatch (micro-second resolution) in processing tasks (included 2.2 and 4). However, the summary of time analysis is presented as followed. The orchestration engine application requires component states to process the output command in order to operate the control task (Table 2). This is achieved by sending DPWS event messages (when inputs/outputs change states) from control devices to subscribers (clients). This process takes approximately **6-7 milliseconds per DPWS message (subscription)** for a device to detect the I/O state change, encode the DPWS message, and network time. The orchestration engine application takes approximately **10-12 milliseconds for decoding received message, encoding a DPWS invocation message** which occurs after I/O logic scanning time. The DPWS command transaction is then initiated by the client (Table 1) which takes around **13- 14 millisecond per DPWS command (service invocation)** to complete. This process includes time on the network, decoding the SOAP message, activating output operation and encoding for the SOAP response message. Therefore, in the complete cycle of the control application from an input sensor to initiate the operation on an output drive is estimated **29 -33 milliseconds**. It is noted that this DPWS time analysis is measured and averaged by a number of captured data in fully operating mode of the control test rig which has 4 distributed controller nodes of 7 components composed of 21 elements. It is evident that the Web Services- based control system implemented in this research performs within the soft real-time constraint (20-250 milliseconds) required by the machine end-user.

7.2 Control system reconfiguration

In assessing the degree of reconfigurability of the automation system, the test scenarios have been set to capture the activities involved in hardware and software reconfiguration based on this Web services based automation system. The test scenarios have been considered the real industrial cases in modifying process work flows and adding new components to suit the new product types. The component which provides unique services is operated according to its internal and external states (interlocking) among devices observed by the service invocator.

Control devices are loosely coupled through device services at a higher level where the device working behaviours are formulated. Within this approach, there is no direct hard coding of control devices and the low level I/O device program. That means in the real automation environment replacing one device will not affect the hard coding (encapsulated I/O device programming) of other devices. The change will only occur at the interlocking level without any change in the low level I/O program.

It can be observed that the component's internal implementation is separated from the application specification corresponding to machine tasks. This allows the system integrator to develop and verify the control application without requiring to understand the complexity of the low-level implementation details since the application is clearly organised and visualised at the higher level. On the other hand, changing of the system behaviours in traditional PLC-based systems typically has to be made through an SFC (Sequential Function Chart) or the ladder logic program at the PLC code programming level. This requires skill and considerable time to ensure that all the low-level implementation code related to the modification has been correctly reviewed and changed.

In addition to the implementation of the new installation device for the reconfigured process, introducing new components can be done with ease by using Web Service discovery to locate the target devices. Once discovered, a device publishes the services it provides to the system. The plug-and-play concept significantly reduces the manufacturing life-cycle time associated with machine (re)configuration process. This reduction in configuration time facilitates the introduction of new products or product variants, reducing production cost and time.

Based on this approach, the reconfiguration of production lines both in term of new application logic and the addition of new components is considered to be a relatively easy task which perhaps takes only a couple of hours to deploy the new system reconfiguration since it occurs at the system level, avoiding any significant changes at the devices operational program. In addition to the design of machine components, the new component has been developed by reusing the control code (DPWS interface and low level programming) from the existing components. Hence, the flexibility and agility of the automation system is increased.

7.3 Business system integration

Current business to shop floor integration is tailored to specific control systems with vendor specific sets of device drivers acting as a gateway solution to be programmed and configured by the user. In contrast, in the implementation of this Web Services- based automation system, no vendors of specific device driver interfaces are required for the integration. The integration of the control system is achieved via the common DPWS interface (SOA device middleware). The required runtime device information and functionality required by the business level are embedded in the Web Services enabled device that provides device information and live states to integrate applications throughout the manufacturing control and business system. In addition, the state and error information of devices can be directly sourced from devices directly to the higher control application by mean of state subscription and publication using a standard DPWS protocol via WS-Eventing.

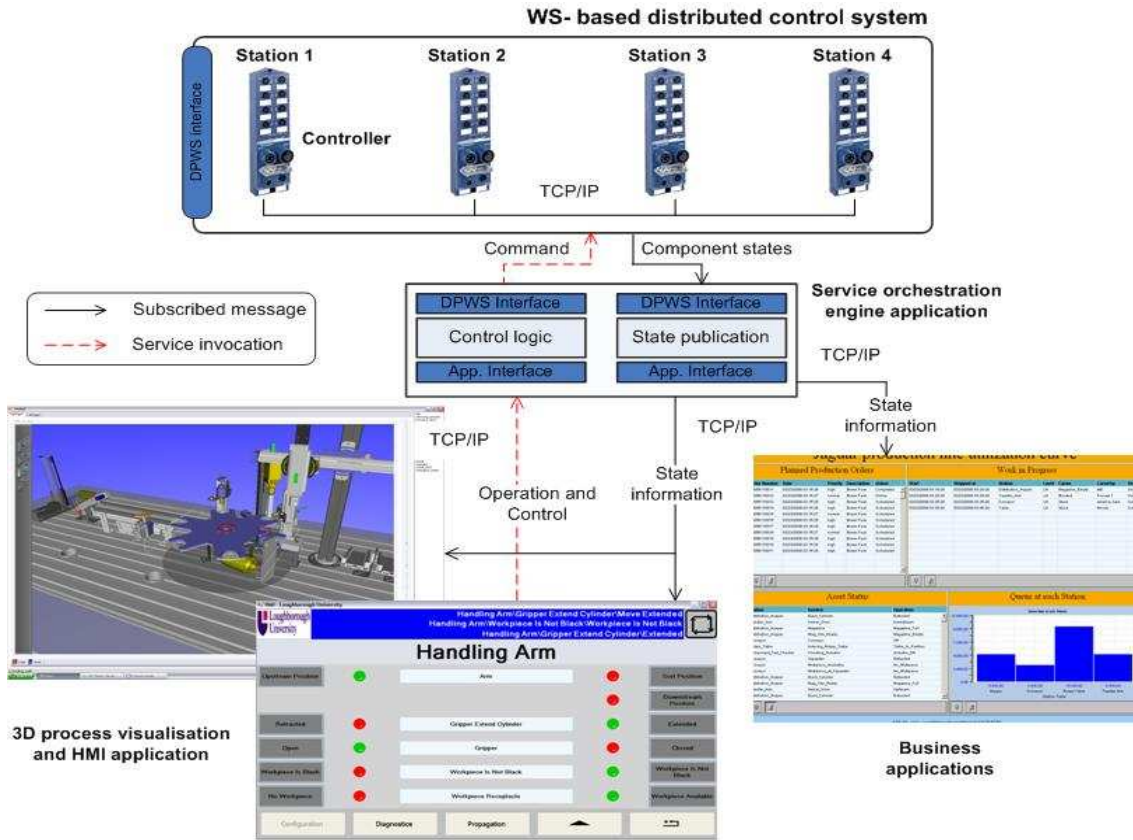


Figure 12. Business system integration via Service orchestration engine

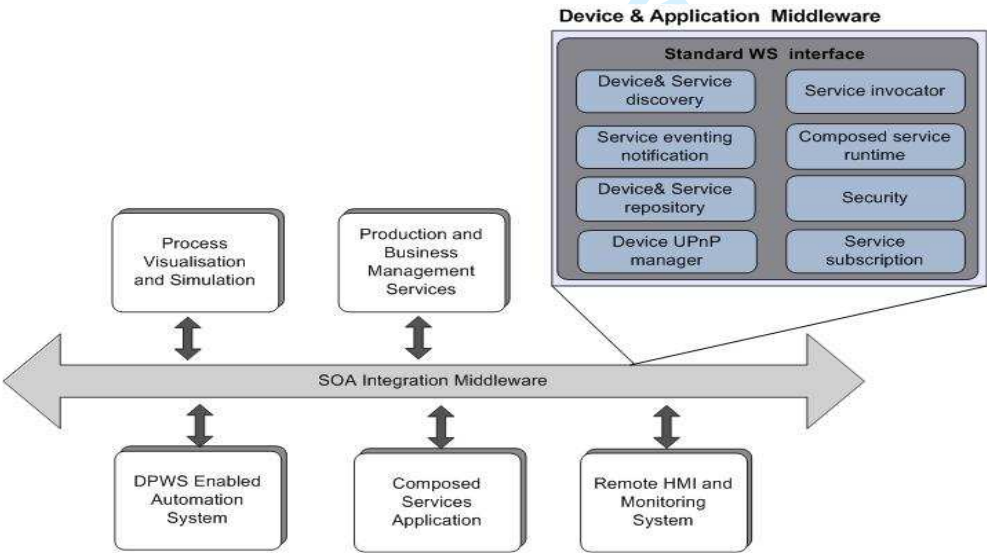


Figure 13. SOA integration middleware

In our current implementation, the integration to the control rig is achieved via the service orchestration engine application as a mediator to remote HMI and the process visualisation through the application interface and TCP/IP connection (figure 12). For the seamless integration enabled by the SOA middleware (figure 13), Web Services on the devices via DPWS provide direct communication through the device discovery service by broadcasting the Web Services discovery message over the network (WS- discovery) to the inter-connected applications. The device description service is used as the Meta-data file of the device defined in the DPWS protocol stack. This file describes the function/ application services the device provides, including the part name, serial number, programme version and etc. In this case, the interaction between automation devices and connected applications is loosely coupled and achieved in the same manner as DPWS enabled devices and the orchestration engine application. Nonetheless, the business and manufacturing applications need to be Web Services enabled to integrate to the control system via the standard Web Services interface in the SOA middleware.

8. Future work plan

The next step in the project is to extend the current design of the Web Service- based control system to a peer-to-peer communication approach on the powertrain assembly test rig. Extension of the Loughborough University developed suite of engineering tools to support the building of Web Services- based control via automated configuration of the embedded devices. Further investigation is also needed to determine the reliability of message passing and I/O response times between the control devices in this discrete event-driven control environment.

9. Conclusion

In this paper the conceptual framework of a SOA in connection with a Web Services- based paradigm has been outlined. The implemented Web Services approach for automation systems has been achieved with proven soft real-time capability, business and process application integration as well as control system reconfigurability. The utilisation of a Web Services protocol stack enables the evolution of an open standard for manufacturing automation, which is technology neutral and provides interoperability between various device vendors through common SOAP messages. Dynamic configuration of intelligent embedded devices using loosely coupled services provides significant advantages for highly dynamic and ad hoc distributed applications, as opposed to the use of more rigid technologies such as those based on distributed objects. For example, device description is embedded into the components and intelligence is devolved down to the device level. This eliminates the need for system integrators to understand completely how the devices are programmed. High-level configuration tools to build and reconfigure automation systems from Web Services enabled components can be used throughout the system lifecycle.

Based on the results obtained in this research, it is evident that the key functionality in DPWS is the provision of online discovery and service invocation for devices based on XML message passing among devices. Higher control levels in manufacturing automation systems are able to connect to the DPWS enabled automation system via standard Web Services and SOAP applications. Moreover, the Web Services functions embedded in the control device provide sufficient performance for industrial use under soft-real time constraints. There are, however, some drawbacks of Web Services realised that can be stated as:

a) The development of the DPWS component, the DPWS coding and I/O device interfaces, has **currently** been commissioned manually. This task is time consuming, as well as **requiring** substantial knowledge and expertise. **The large** memory footprint of the DPWS stack (around 240 **kbytes**) on the controller is another concern on embedded devices with **limited** memory.

b) The large size of a SOAP message (around 1 kbyte) could potentially contribute to high network bandwidth/**loads** that may affect system integrity **for** large **scale automation systems**. Further **investigations into** networking performance and optimisation are required.

c) As evident from the DPWS transaction time analysis, **the current performance of Web Services is not mature** enough to be **readily utilised in time constrained** and precise positioning control applications such as CNC, CMM machines, and **robotic systems** where maximum processing speed and network determinism are **required**. Despite this, the current Web Services could potentially be exploited within these applications **for higher level** process synchronisation and **work flow control** between machines where the response time is **not critical to success/ failure**.

Research on the process engineering tools for the dynamic deployment of the Web Services to minimise the human engineering effort and development time is ongoing. This work will enhance the interoperability and scalability of WS-based automation for more complex systems. Furthermore, significant research work and industrial evaluation are required for the current implemented Web Services to be achieved a level of maturity and be **applicable** in complicated engineering systems. However, with the fast technological development of Web Services in **terms** of optimising network **speeds** and **reducing the** DPWS parsing time using **the** SOAP binary form **coupled with** more powerful control devices and supported **by** engineering **configuration and evaluation** tools, it is **expected** that **limitations** could be readily resolved.

Acknowledgment

The authors gratefully acknowledge the support of the EU FP6 SOCRADES and the EPSRC, IMCRC, GAIN and BDA projects and their collaborators in enabling various aspects of this research.

Reference

- ARM®, *RealView Debugger*. Available from: <http://www.arm.com/products/DevTools/RVD.html> (accessed 1 February 2009).
- Assen, M.F., Hans, E.W., and Velde, S.L., 2000. An agile planning and control framework for customer-order driven discrete parts manufacturing environments. In *International Journal of Agile Management Systems*, 2(1), 16-23.
- Barry, D. K., 2000. *Service-Oriented Architecture (SOA) Definition*. Available from: http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html (accessed 16 January 2009).
- Barry, R., 2003. *Real Time Application Design Using FreeRTOS in small embedded systems*. FreeRTOS™. Available from: <http://www.freertos.org/tutorial/> (accessed 18 January 2009).
- Bepperling, A., *et al.*, 2006. D 2.1 The Mechatronic automation framework and its architectural requirements. In *Radically Innovative Mechatronics and Advanced Control Systems (RI-MACS) project*, Internal document, 19-26.
- Bentham, J., 2000. *TCP/IP Lean Web Servers for Embedded Systems*. Kansas: CMP Books, CMP Media, Inc. 155-168.

- Boritz, J.E., and Gyun, W., 2005. Security in XML-based financial reporting services on the Internet. In *Journal of Accounting and Public Policy*, 24(1), 11-35.
- Büyüközkan, G., Derel, T., and Baykasoğlu A., 2004. A survey on the methods and tools of concurrent new product development and agile manufacturing. In *International Journal of Intelligent Manufacturing*, 15(6), 731-751.
- Cachapa, D., et al., 2007. An Approach for Integrating Real and Virtual Production Automation Devices Applying the Service-oriented Architecture Paradigm. in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, September 2007, 309-314
- Colombo, A.W., et al., 2004. A Collaborative Automation Approach to Distributed Production System. In *2nd International Conference on Industrial Informatics (INDIN'04)*, June 24-26 2004, Berlin, Germany, 27-32.
- Denis, B., et al., 2007. Measuring the impact of vertical integration on response times in Ethernet fieldbuses. In *IEEE conference on Emerging Technologies and Factory Automation*, 25-28 Sept. 2007, 532-539.
- Engelen, R.V., 2004. Code Generation Techniques for Developing Light-Weight XML Web Services for Embedded Devices. In *proceeding of the 2004 ACM symposium on Applied computing conference on Embedded systems: applications, solutions and techniques (EMBS)*, March 14-17, 2004, Nicosia, Cyprus. ACM: New York, USA, 854-861.
- Graham, S., et al., 2004. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*. 2nd ed. Sams publishing.
- Gunasekaran, 1998. Agile manufacturing: enablers and an implementation framework. In *International Journal of Production Research*, 36(5), 1223-1247.
- Harrison, R., Lee, S.M., West, A.A., 2004. Lifecycle Engineering of Modular Automated Machines. In *2nd International Conference on Industrial Informatics (INDIN'04)*, June 24-26 2004, Berlin, Germany, 501-506.
- Harrison, R., et al., 2006. Reconfigurable modular automation systems for automotive power-train manufacture. In *Journal of Flexible Manufacturing System*, Springer, 18, 175-190.
- Hung, M.H., Cheng, F.T., Yeh, S.C., 2005. Development of a Web-Services- Based e-Diagnostics Framework for Semiconductor Manufacturing Industry. In *IEEE Transactions on Semiconductor Manufacturing*, 18(1), 122-135.
- InetDaemon Enterprise, 1996a. *TCP 3-way Handshake*. Available from: http://www.inetdaemon.com/tutorials/internet/tcp/3way_handshake.shtml (accessed 18 January 2009).
- InetDaemon Enterprise, 1996b. *TCP Connections*. Available from: <http://www.inetdaemon.com/tutorials/internet/tcp/connections.shtml> (accessed 18 January 2009).
- Jammes, F., Smith, H., 2005. Service-Orient Paradigms in Industrial Automation. In *International Journal of Industrial Informatics*, 1(1), 62-70.
- Jammes, F., Mensch, A., and Smit, H., 2007. Service-Oriented Device Communications using the Devices Profile for Web Services. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, Grenoble, France, 1-8.
- Lee, S.M., Harrison, R., West, A.A., 2004a. A Component-based Distributed Control System. In *2nd International Conference on Industrial Informatics (INDIN'04)*, June 24-26 2004, Berlin, Germany, 33-38.
- Lee, S.M., 2004b. *A Component-Based Distributed Control Paradigm for Manufacturing Automation System*. Thesis (PhD). Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University.
- Liu, H., Li, M.L., and Lin, X., 2008. Mapping Web Services Standards to Federated Identity Management Requirements for m-Health. In *International Conference on Internet Computing in Science and Engineering*, 459-466.

- Machado, G.B., Mitmann, R., 2006. Embedded Systems Integration Using Web Services. In *proceeding of International Conference on System and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL)*. IEEE Computer Society.
- McLean, C.R., Bloom, H.M., Hopp, T.H., 1982. The Virtual Manufacturing Cell. In *Proceeding of the IFAC/IFIP Conference on Information control Problems in Manufacturing Technology*, October 1982, Gaithersburg.
- MeDiSol™, *IP Protection class*. Available from: <http://www.medisol.org/en/products/knowledge-technology/ip-protection-classes.html> (accessed 1 February 2009).
- Molina, A., et al., 2005. Next-generation manufacturing systems: key research issues in developing and integrating reconfigurable and intelligent machine. In *International Journal of Computer Integrated Manufacturing*, 18, 25-536.
- Nickull, D., 2005. *Service Oriented Architecture*. Adobe Systems Incorporated. Available from: http://www.adobe.com/jp/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf (accessed 16 January 2009).
- Phaithoonbuathong, P., Harrison, R., McLeod, S., 2007. A Web Services Based Automation Paradigm for Agile Manufacturing. In *4th International Conference on Responsive Manufacturing (ICRM'2007)*, 17-19 September 2007, School of Mechanical, Materials and Manufacturing Engineering, The University of Nottingham.
- Weston, R.H., Harrison, R., Booth, A.H., and Moore, P.R., 1989. Universal machine control system primitives for modular distributed manipulator systems. In *International Journal of Production Research*, 1989, 27(3), 395-410.
- Weston, R.H., 1999. Model-driven, component-based approach to reconfiguring manufacturing software systems. In *International Journal of Operations & Production Management*, 1999, 19(8), 834-855.
- Schneider Electric, Web Services team, Grenoble, France, *Advantys FTB: System user guide*. Available from: http://www.automation.schneider-electric.com/system_user_guides/pdf_files/8/Twido_Altivar_Magelis_OTB_FTB_EN.pdf (accessed 1 February 2009).
- SOCRADES. EU project, 2006. Service-oriented cross-layer infrastructure for distributed smart embedded systems. Available from: <http://www.socrades.eu/Project/Presentation/default.html> (accessed 16 January 2009).
- SODA-ITEA Consortium, 2007. *Software component: DPWS C stack*. Available from: <http://www.soda-itea.org/Downloads/SoftwareComponents/default.html> (accessed 18 January 2009).
- Sperling, W., Lutz, P., 1997. Designing Application for An OSACA Control. In *the International Mechanical Engineering Congress and Exposition*, November 1997, Dallas/USA.
- Trifa, V. M., Guinard, D., and Koehler, M., 2008. Messaging Methods in a Service-Oriented Architecture for Industrial Automation Systems. In *International Conference on Networked Sensing Systems (INSS)*, June 2008, 35-38.
- Tseng, M. M., Lei, M. and Su, C. J., 1997. A Collaborative Control System for Mass Customization Manufacturing. In *Annals of the CIRP- Manufacturing Technology*, 46(1) 373-376.
- Wang, L., 2004. Web-based decision making for collaborative manufacturing. In *International Journal of Computer Integrated Manufacturing*, 22(4), 334-344.
- W3C® World Wide Web Consortium, 2004. *XML Schema 1.1*. Available from: <http://www.w3.org/XML/Schema> (accessed 16 January 2009).
- Yu, S.C., and Chen, R.S., 2003. Web services: XML based system integrated techniques. 2003. In *Emerald Journal Electronic library*, 21(4), 358-366.

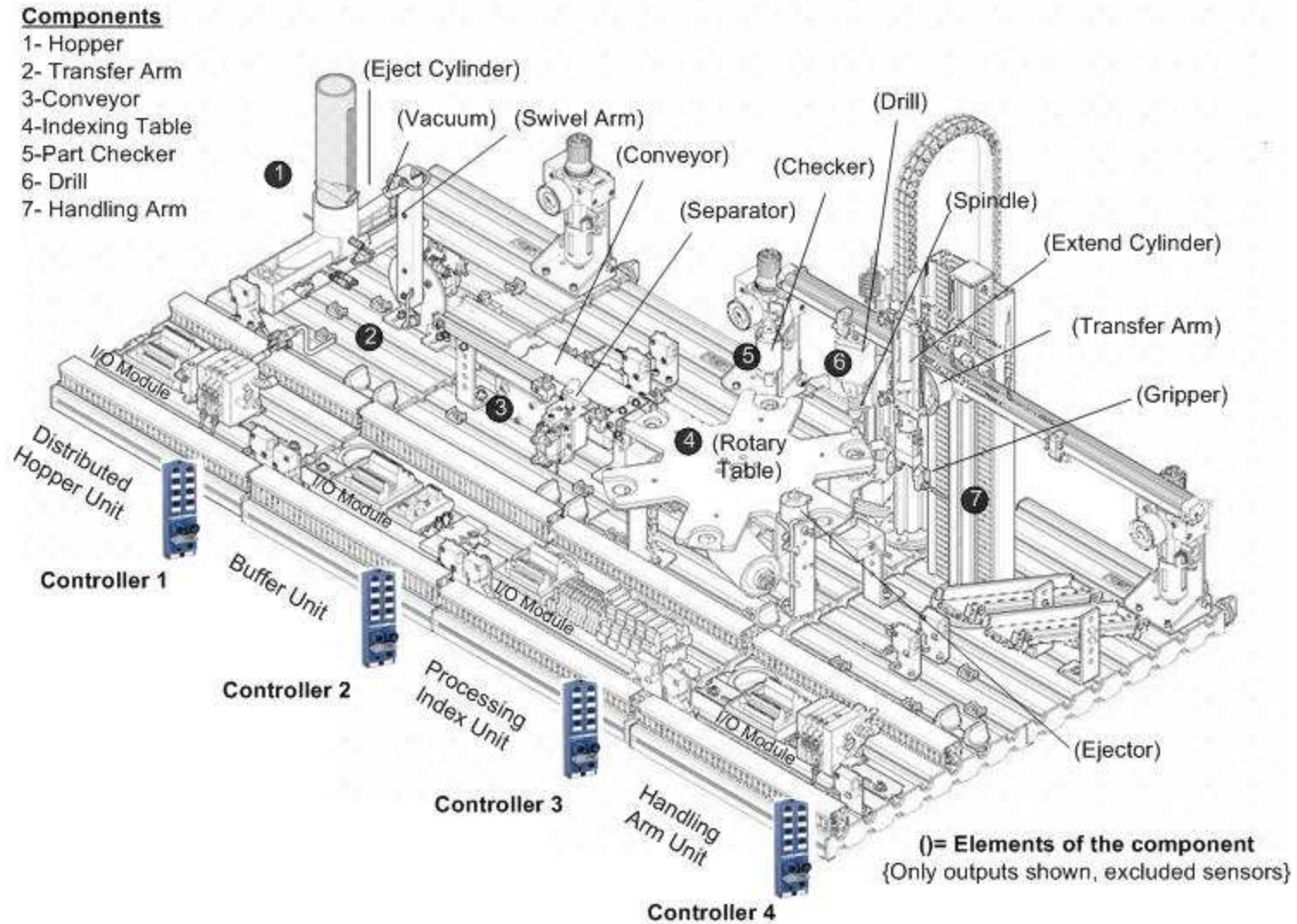


Figure 1 A distributed CB design of a powertrain assembly machine

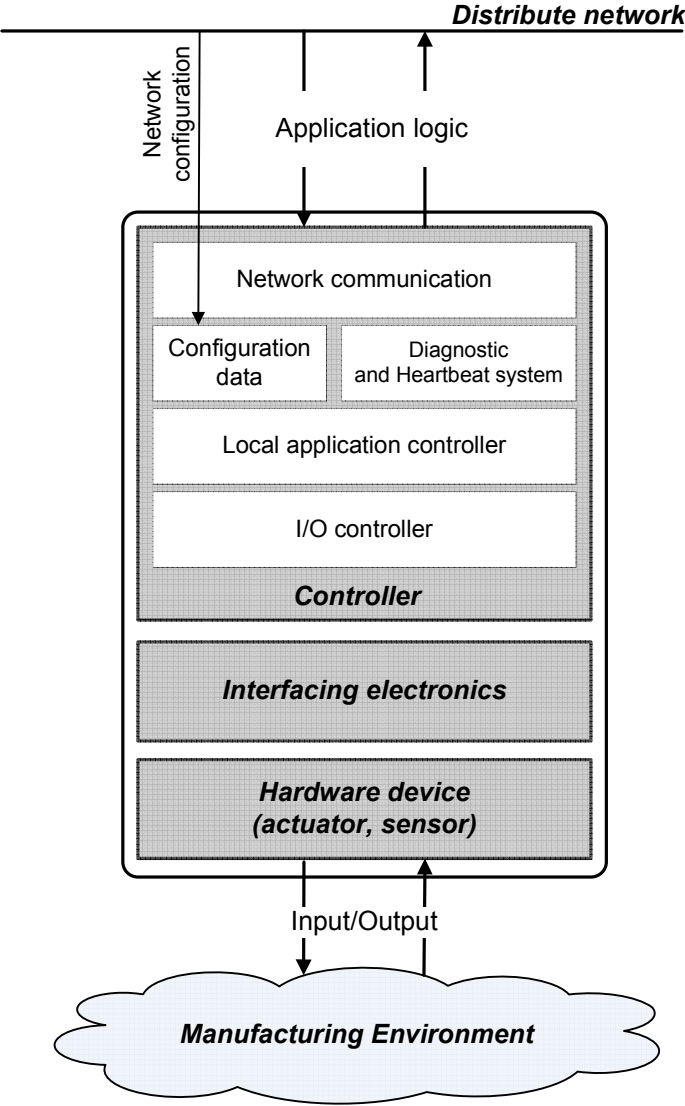


Figure 2 Physical constituent of a component

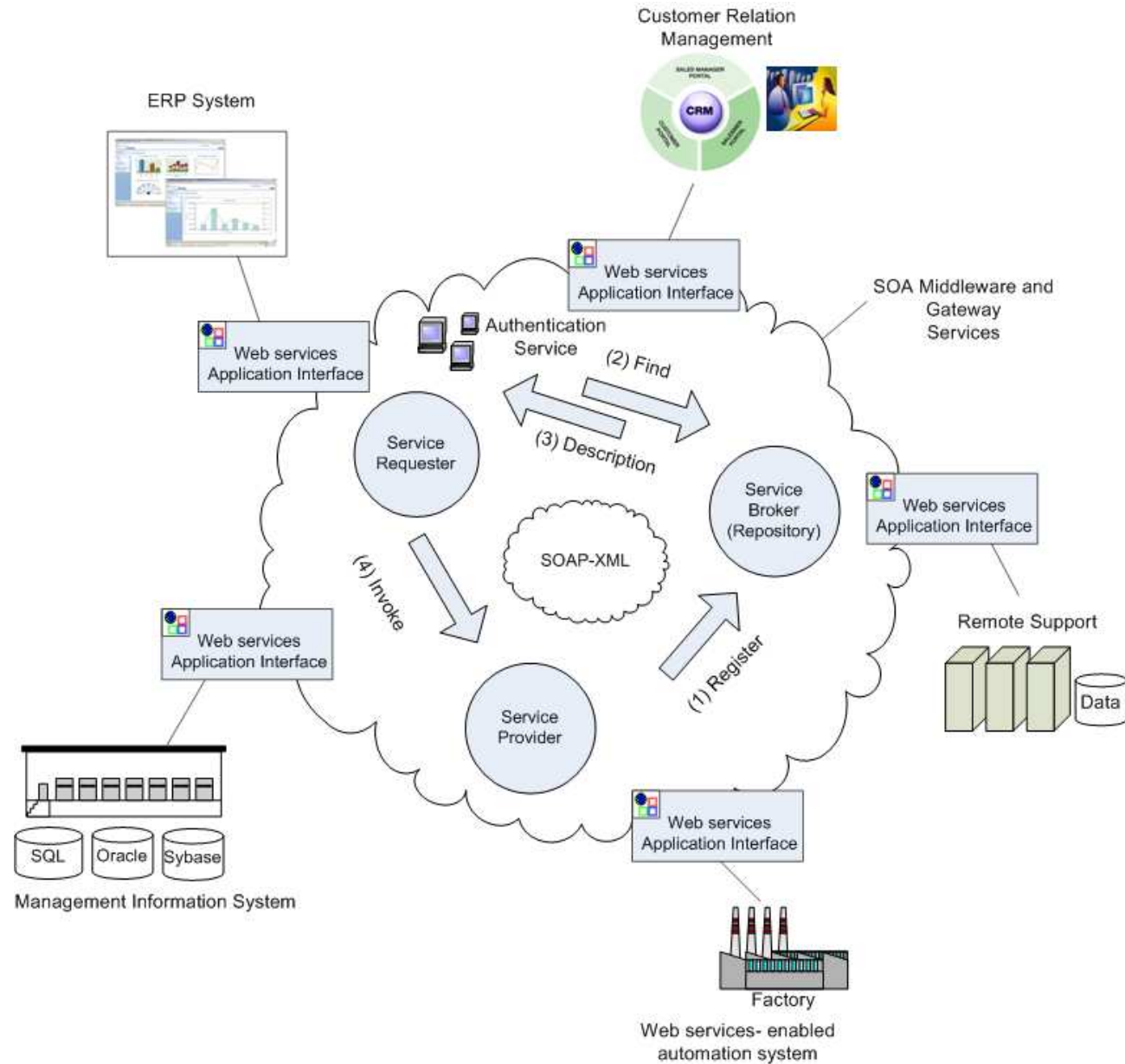


Figure3. SOA-Web Services basic integration framework

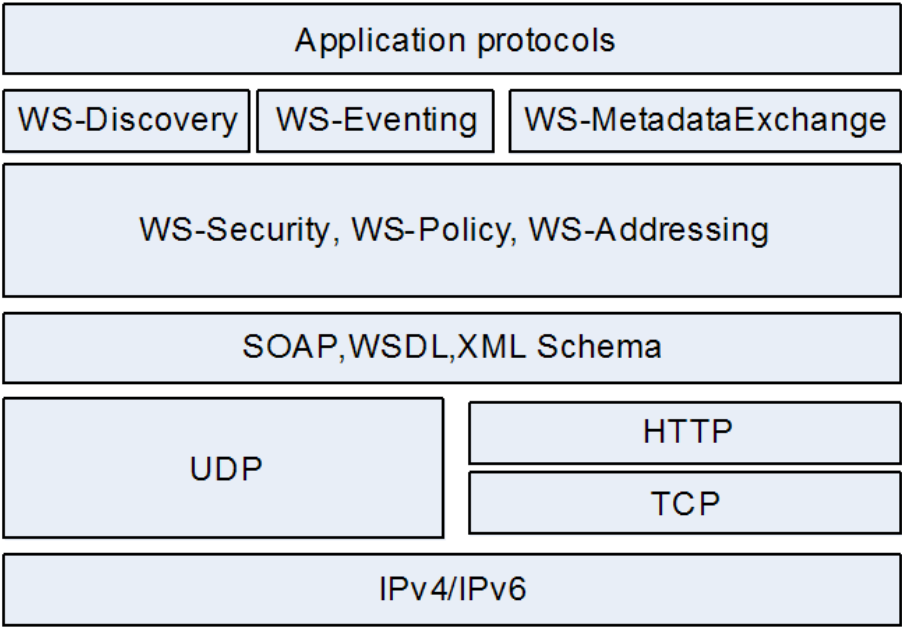


Figure 4. Devices Profile for Web Services (DPWS) protocol stack

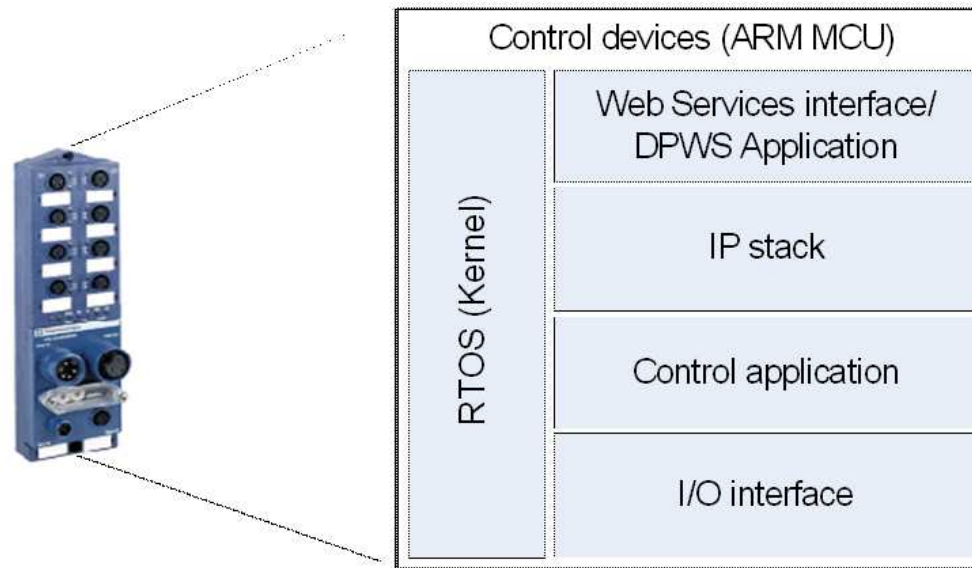


Figure 5. WS component software architecture

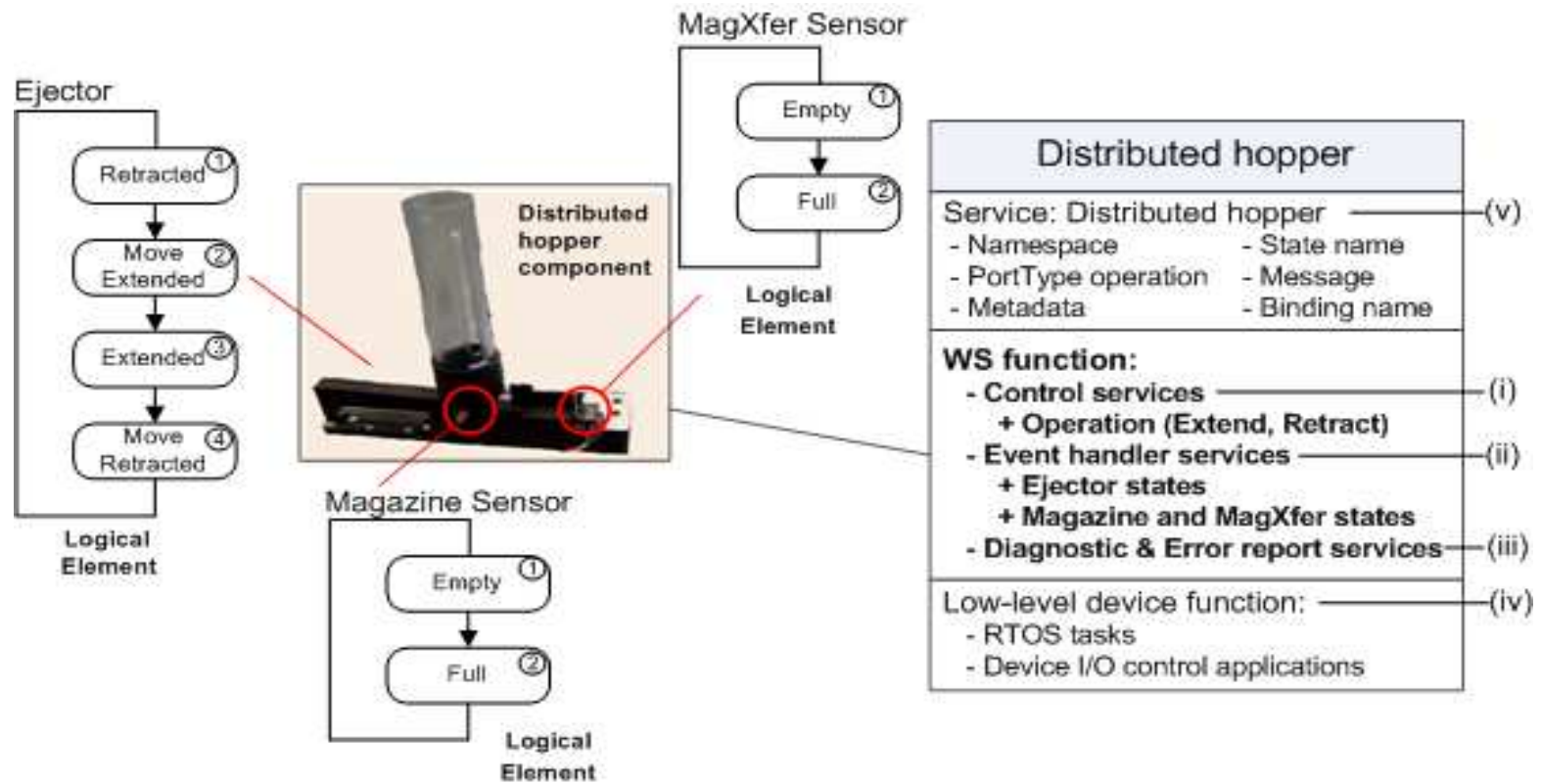


Figure 6. A Web service- based design component

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions name="Distributor"
  targetNamespace="http://www.soda-itea2.org/MSI_Lboro/Demonstrator/Distributor"
  .....

```

Figure 7 a)

```

/*-----HANDLING ARM OPERATION COMMAND-----*/
int __dst_distributorCmd(struct dpws* dpws, enum dst_distributorAction dst_distributorActionElement,
  struct
  _dst_ResponseElement *dst_ResponseElement)
{
  //Message received acknowledgement
  NumOrder++;
  dst_ResponseElement->OrderID = NumOrder;
  dst_ResponseElement.OrderID = hla_ResponseElement->OrderID;
  dst_ResultElement.allRequestStatus = dst_RequestStatus_ACK;
  dpws_notify__dst_distributorCmdAck(&pState->dpws, pState->endpointRef0, &dst_ResponseElement);
  switch (dst_distributorActionElement)
  {
    case dst_distributorAction_MoveExtended: -----(vi)
      MoveExtend(); //Setting output pins -----[see figure 7/c-
      (vii)]
      //Command acknowledgement
      dst_ResponseElement->allOperationResultStatus =
      dst_OperationResultStatus__DONE;
      dpws_end(&pState->dpws);
      break;
      .....other operation commands
    case dst_distributorAction_Reset:
      Shutdown(); //Cleaning and stopping DPWS and I/O routines
      //Command acknowledgement
      dst_ResponseElement->allOperationResultStatus =
      dst_OperationResultStatus__DONE;
      dpws_end(&pState->dpws);
      break;
      default:
      //Command failure
      dst_ResponseElement->allOperationResultStatus =
      dst_OperationResultStatus__FAILURE;
      dpws_end(&pState->dpws);
      break;
  }
  return SOAP_OK;
}

```

Figure 7 b)

```
/*-----DEVICE I/O OPERATION-----*/
int MoveExtend () -----(vii)
{
    trGPIO_Set(8, FALSE); //channel 08 Retracted pos
    trGPIO_Set(9, TRUE); //channel 09 Extended pos

    /*Reading input sensor*/
    do
    {
        sensor1=trGPIO_Get(1); //channel 01 At retracted sensor
        sensor2=trGPIO_Get(2); //channel 02 At extended sensor
    }while(!(sensor1==FALSE&sensor2==TRUE));

    return 0;
}
```

Figure 7 c)

```
/*-----DPWS DEVICE STATE NOTIFICATION-----*/
int dpwsDistributerNotify(DistributerStatus1) -----(viii)
{
    switch (DistributerStatus1)
    {
        case Extended:
            distributerNotifyElement.distributerStatus = dst__
            distributerStatus__Ejector_Extended;
            break;
        .....other cases
        case EJECTOR_ERROR:
            distributerNotifyElement.distributerStatus = dst__ distributerStatus__EJECTOR_ERROR;
            break;
    }
    dpws_notify__dst__distributerNotifyEvent(&event0.dpws, event0.endpointRef0, &distributerNotifyElement);
    return 0;
}
```

Figure 7 d)

```
/*-----Detecting IO changes -----*/
int IOSPY(void)
{
    ...Initialising initial Distribute state
    do
    {
        ...Reading sensors state
        /*PARSING HANDLING ARM STATE*/
        if (sensor1==FALSE&sensor2==TRUE)
        {
            DistributerStatus1= Extended;
        }
        else if (sensor1==TRUE&sensor2==FALSE)
        {
            ...
        }
        .....conditions
    }
    else
    {
        //State unknown (error)

        DistributerStatus1= EJECTOR_ERROR;
        //Fail safe routine enabled
        ...Safely releases a work piece and return to initial position
    }

    /*-----DPWS Notification-----*/
    if (DistributerStatus1!= Pre_ DistributerStatus1)
    {
        ...state change detected
        dpwsDistributerNotify(DistributerStatus1); -----[see figure 7/d-(viii)]
    }
    ...
}while(1);
return 0;
}
```

Figure 7 e)

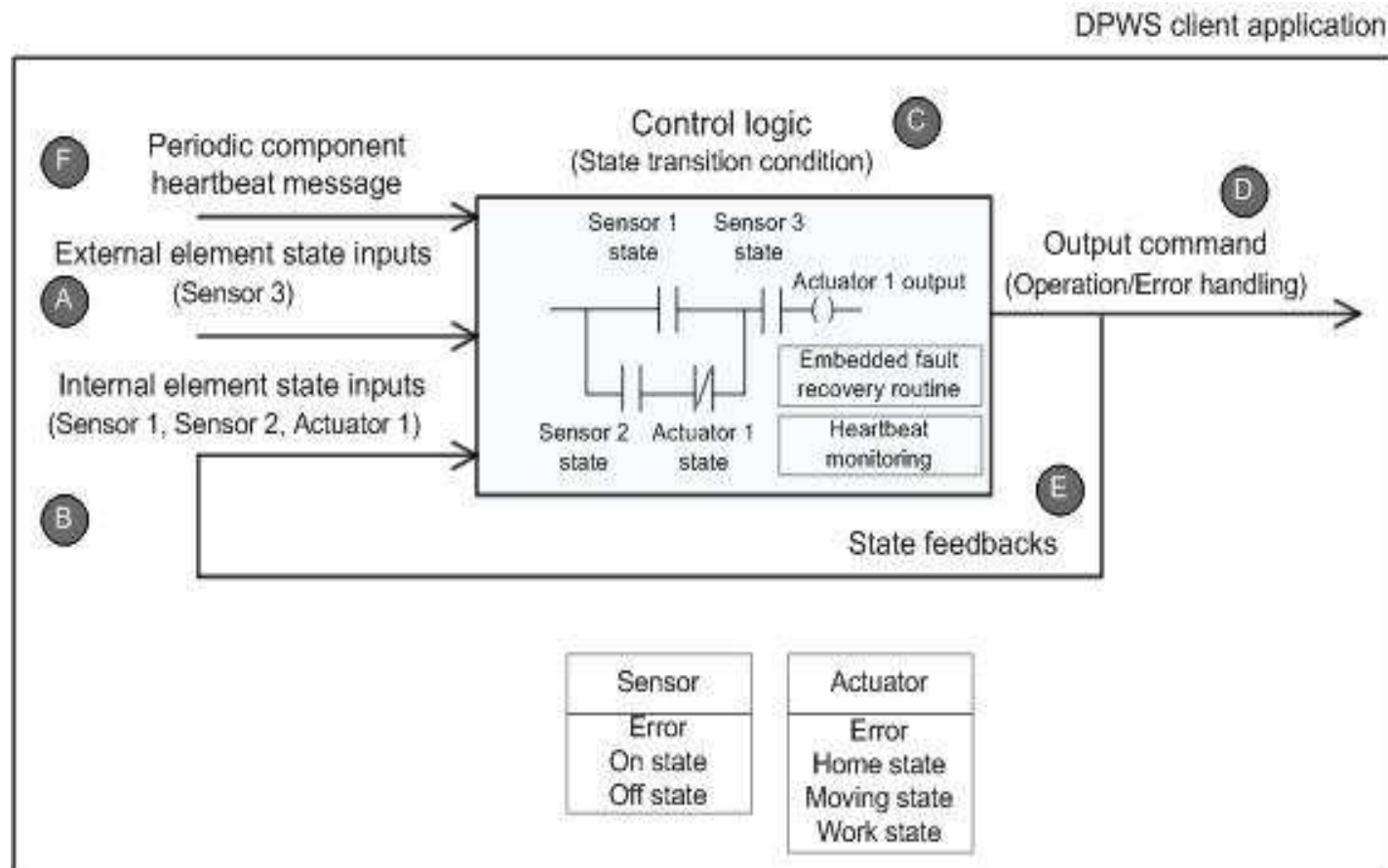


Figure 8. Component state transition diagram

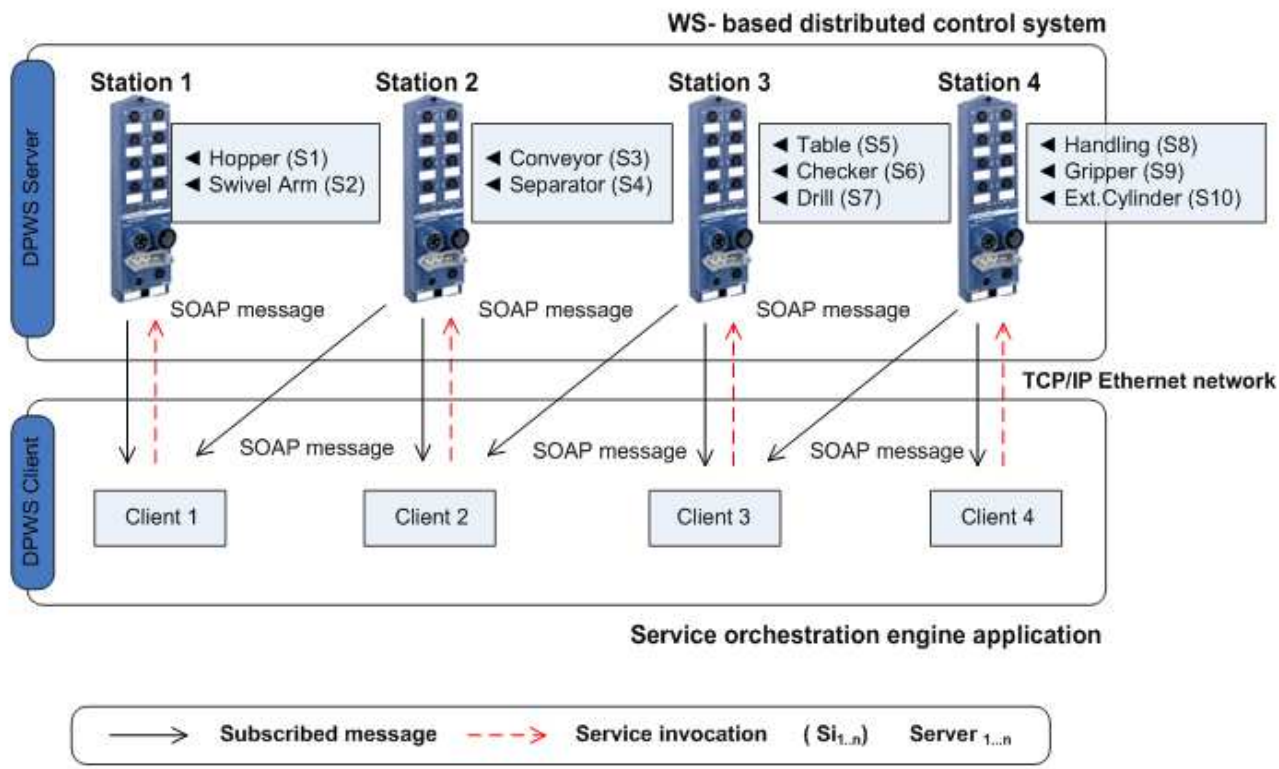


Figure 9. A PC- based orchestration engine application

[The DPWS invoked command]

dpws_call_dst_distributorCmd(&Distributor, ERPDistributor, NULL, dst__distributorAction_MoveExtended,&ResponseDistributor)



```

POST /13814002-1dd2-11b2-bc3d-0040af000032
HTTP/1.1..Host:150.1.0.102:9882..User-Agent: gSOAP/2.7..
Content-Type: application/soap+xml; charset=utf-8..Content-Length: 773..Connection: close...

<?xml version="1.0" encoding="UTF-8"?>.
<SOAP-ENV:Envelope      xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
                        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                        xmlns:blt="http://www.soda-itea2.org/MSI-
Liboro/Demonstrator/Distributor">
  <SOAP-ENV:Header>
    <wsa:To>http://150.1.0.102:9882/13814002-1dd2-11b2-bc3d-0040af000032</wsa:To>
    <wsa:Action>http://www.soda-itea2.org/MSI-Liboro/Demonstrator/Distributor/distributorCmdRequest</wsa:Action>
    <wsa:MessageID>urn:uuid:41c616ba-3e14-11dd-8430-001c42935fe4</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
    </wsa:ReplyTo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <dst:distributorCmd>
      <dst:DistributorAction>MoveExtended</dst:DistributorAction> [see figure 7/d-(vi)]
    </dst:distributorCmd>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Figure 10. Output SOAP message of a DPWS invoked command

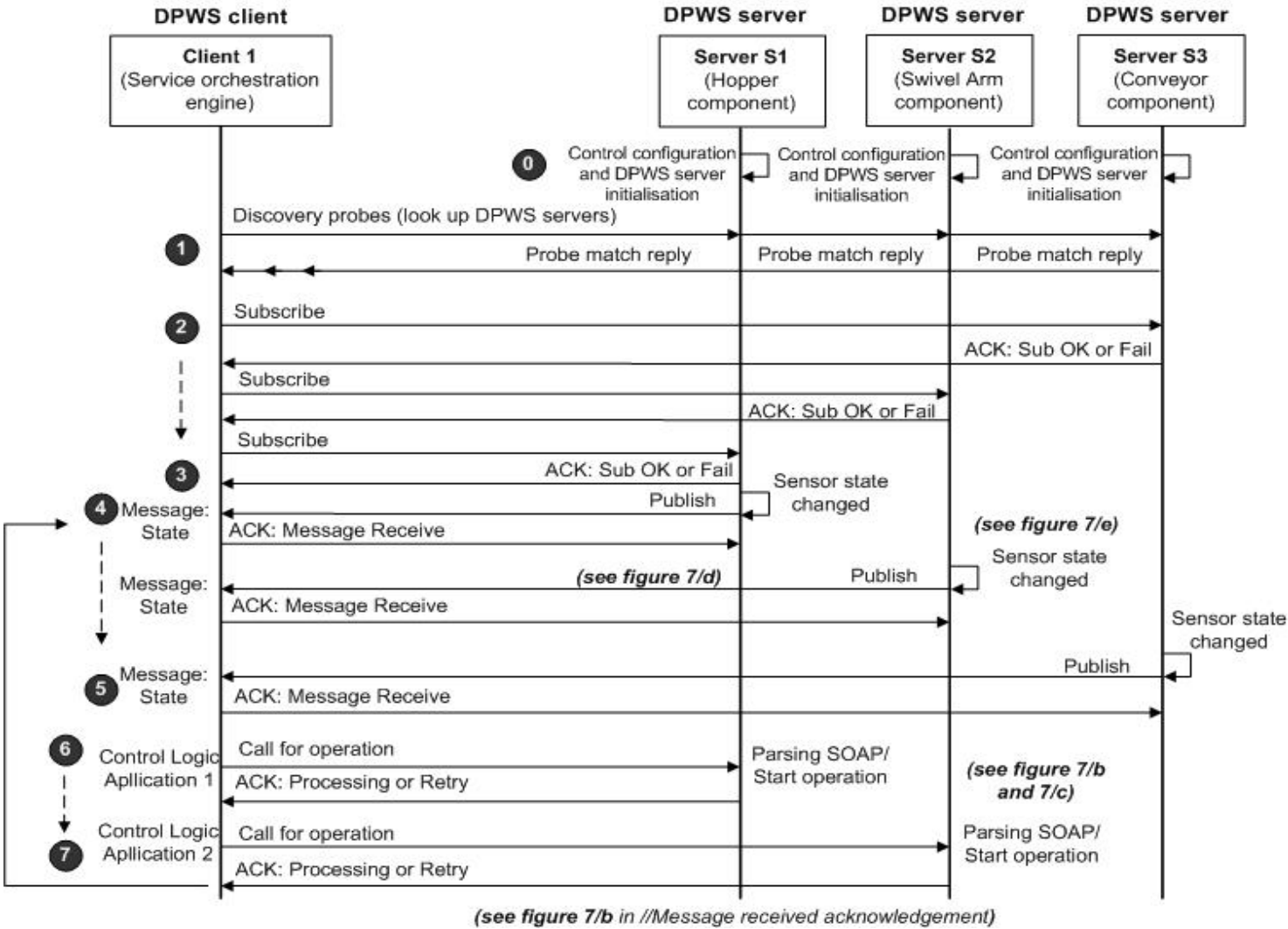


Figure 11. Test rig WS control system use case

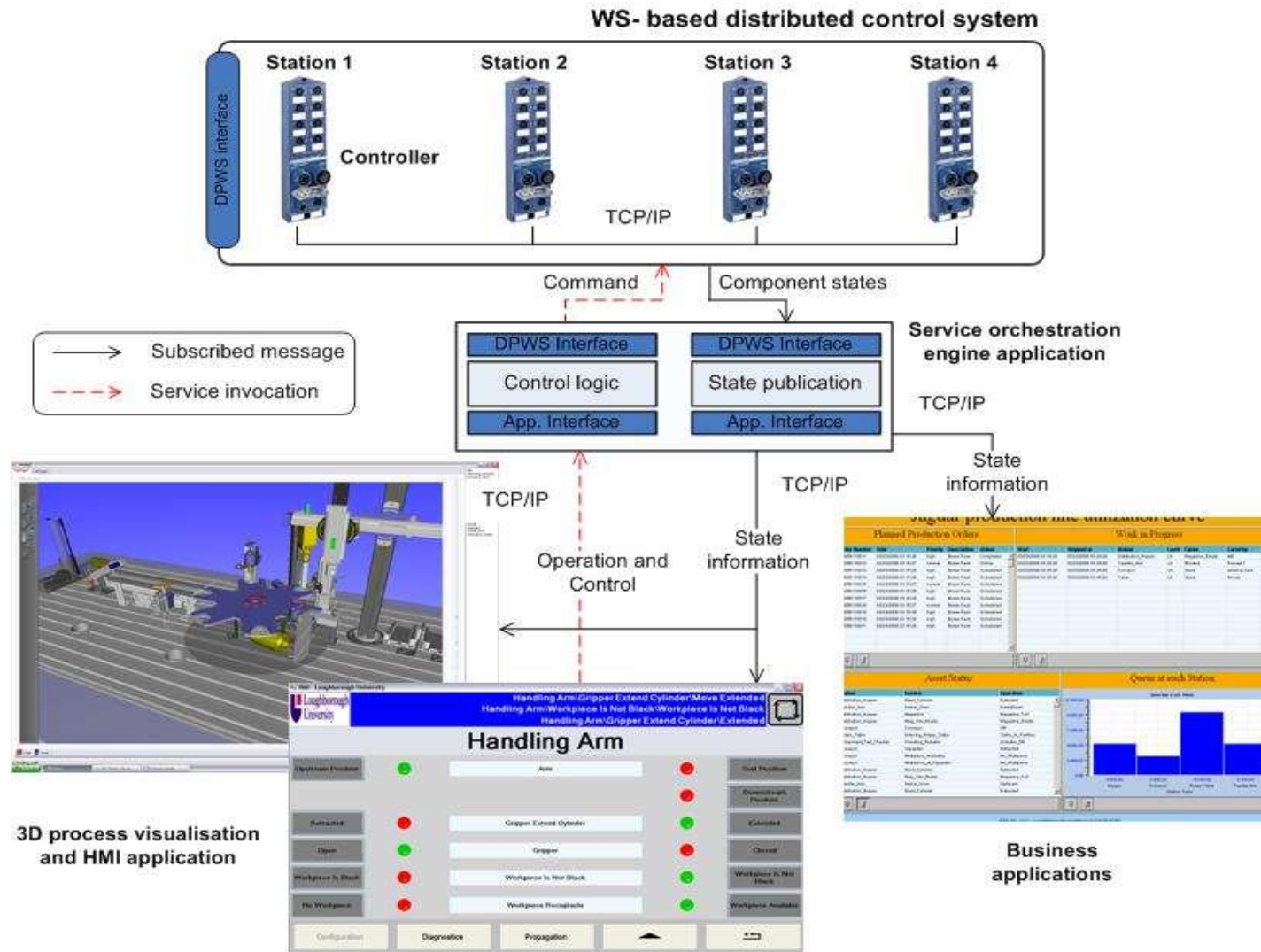


Figure 12. Business system integration via Service orchestration engine

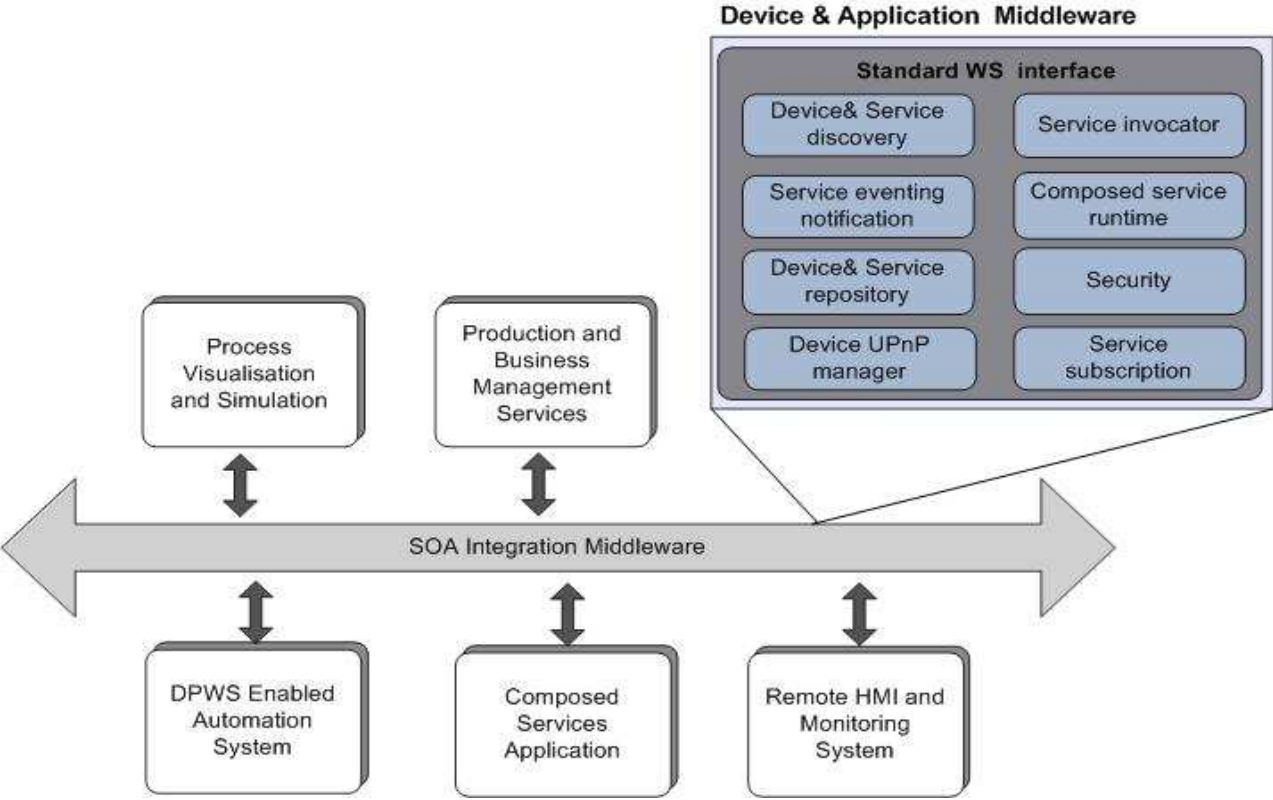


Figure 13. SOA integration middleware

Table 1. Packet synchronisation for the DPWS command invocation from Host A (client) to Host B (server)

| Packet No. | Packet size (Bytes) | Data load (Bytes) | 0-Initial packet received time (ms) | Time different between packets (ms) | Event | Seq No* | Ack No* | Diagram | |
|------------|---------------------|-------------------|-------------------------------------|-------------------------------------|---|------------------------|---------|-----------|-----------|
| | | | | | | | | HOST A | HOST B |
| 1 | 78 | 0 | 0 | 0 | Host A sends a TCP synchronize packet to Host B to initiate a connection | 3976 | 0 | send t | t+1 rcv |
| 2 | 60 | 0 | 0.37 | 0.37 | Host B replies Host A to acknowledge the request on the connection | 6001 | 3977 | rcv t+3 | t+2 send |
| 3 | 60 | 0 | 1.07 | 0.70 | Host A acknowledges Host B; The connection between A-B is established | 3977 0; Data load | 6002 | send t+4 | t+5 rcv |
| 4 | 969 | 915 | 1.082 | 0.012 | Host A sends a DPWS message (SOAP message) to Host B | 3977 915; Data load | 6002 | send t+6 | t+7 rcv |
| 5 | 60 | 0 | 1.086 | 0.004 | Host A sends another packet to Host B as the one to be acknowledged and also a request to close the connection from Host A to B | 4892 +1 | 6002 | send t+8 | t+9 rcv |
| 6 | 60 | 0 | 1.874 | 0.788 | Message received; Host B acknowledges Host A | 6002 0; Data load | 4893 | rcv t+11 | t+10 send |
| 7 | 810 | 756 | 10.912 | 9.038 | Host B sends a replied DPWS message to Host A and confirm the connection closed from Host A to B (B to A still active) | 6002 756; Data load | 4893 | rcv t+13 | t+12 send |
| 8 | 60 | 0 | 10.919 | 0.007 | Host B sends a packet to Host A that requests a termination of the connection from Host B to A | 6758 +1 | 4893 | rcv t+15 | t+14 send |
| 9 | 60 | 0 | 13.536 | 2.617 | Message received; Host A acknowledges Host B to terminate the connection from Host B to A | 4893 | 6759 | send t+16 | t+17 rcv |
| | | | | | Message sending/receiving is completed with both connections between Host A and B are closed to start a new TCP connection | 8603 | 0 | send t+18 | -----> |

Table 2. Packet synchronisation for the DPWS message publication from Host B (server) to Host A (client)

| Packet No. | Packet size (Bytes) | Data load (Bytes) | 0-Initial packet received time (ms) | Time different between packets (ms) | Event | Seq No* | Ack No* | Diagram | |
|------------|---------------------|-------------------|-------------------------------------|-------------------------------------|---|------------------------|---------|-----------|-----------|
| | | | | | | | | HOST B | HOST A |
| 1 | 60 | 0 | 0 | 0 | Host B sends a TCP synchronize packet to Host A to initiate a connection | 4001 | 0 | send t | t+1 recv |
| 2 | 60 | 0 | 0.402 | 0.402 | Host A replies Host B to acknowledge the request on the connection | 0225 | 4002 | recv t+3 | t+2 send |
| 3 | 60 | 0 | 0.726 | 0.324 | Host B sends an acknowledgment to Host A; The connection is established | 4002 | 0226 | send t+4 | t+5 recv |
| 4 | 794 | 740 | 2.852 | 2.126 | Host B sends a SOAP message to Host A | 4002 740: Data load | 0226 | send t+6 | t+7 recv |
| 5 | 60 | 0 | 2.856 | 0.005 | Host B sends another packet to Host A as the one to be acknowledged and also a request to close the connection from Host B to A | 4742 | 0226 | send t+8 | t+9 recv |
| 6 | 60 | 0 | 3.349 | 0.493 | Message received; Host A acknowledges Host B | 0226 0: Data load | 4743 | recv t+11 | t+10 send |
| 7 | 168 | 114 | 5.645 | 2.296 | Host A sends another packet to Host B to confirm the closing connection from Host B to A | 0226 114: Data load | 4743 | recv t+13 | t+12 send |
| 8 | 60 | 0 | 5.657 | 0.012 | Host A sends a packet to Host B that requests a termination of the connection from Host A to B | 0340 | 4743 | recv t+15 | t+14 send |
| 9 | 60 | 0 | 6.042 | 0.385 | Message received; Host B acknowledges Host A to terminate the connection from Host A to B | 4743 | 0 | send t+16 | t+17 recv |
| | | | | | Message send/receive is completed with both connections are closed for Host A and B to start the new TCP connection | 8001 | 0 | send t+18 | -----> |

* For the simplicity of demonstrating the packet synchronisation, packet sequence number (Seq No.) and acknowledge number (Ack No.) are only displayed by the last 4 digits of a full number.