TAMPERE UNIVERSITY OF TECHNOLOGY

PETRI KANNISTO

SERVICE-ORIENTED BUSINESS PROCESS MODELING IN OPERATIONS AND MAINTENANCE

Master of Science Thesis

Examiner: Professor Seppo Kuikka
Examiner and topic approved
in the Faculty of Automation,
Mechanical and Materials Engineering
Council meeting on 12th January 2011

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Engineering
KANNISTO, PETRI: Service-Oriented Business Process Modeling in Operations and Maintenance
Master of Science Thesis, 88 pages, 6 appendix pages
February 2011
Major: Automation Software Engineering
Examiner: Professor Seppo Kuikka
Supervisor: Research Scientist David Hästbacka
Keywords: industrial SOA, executable business process, life cycle simulation, XML, BPMN, WS-BPEL, Web Services, HTTP, DPWS, real-time system

Service-Oriented Architecture (SOA) is a paradigm for modeling the interaction of different parties in a distributed system. In SOA, a high abstraction level leads to platform-independent interoperability. Moreover, different parties are only loosely coupled to each other. As a result of these, SOA is a scalable and flexible architecture.

As industrial automation systems are typically inflexible and expensive to install or to modify, it would be beneficial to have all devices interact in the SOA manner. However, current technologies to implement a SOA are problematic from the devices point of view. The technologies require a lot of computational resources, and they also lack support for hard real-time functions. Work has been done to overcome these challenges, but especially hard real-time capable SOA cannot currently be implemented.

Despite their limitations, current SOA technologies can be used for several functions of industrial plants. In this study, service-oriented solutions are created for the estimation of environmental footprints and for condition monitoring. The solutions are modeled as diagrams using a standard graphical notation after which the diagrams are converted to an executable language.

Both implementations show the efficiency of the selected modeling method. The principles of SOA enable the reuse of different resources flexibly in different applications which saves work. A standard structured data format was used in both solutions, and it facilitates integration. As there is a built-in support for the format in modern applications, a solution designer can concentrate on data contents on a high level. Compatibility problems were also encountered, but they were overcome using wrapper services. There were also other integration problems with the technologies used. Despite the problems, graphical modeling saves time compared to textual methods to model communication. It was also recognized that careful design is required in distributed systems to avoid performance problems.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
KANNISTO, PETRI: Palvelukeskeiset liiketoimintaprosessit käynnissäpidon toimintojen mallintamisessa
Diplomityö, 88 sivua, 6 liitesivua
Helmikuu 2011
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Seppo Kuikka
Ohjaaja: tutkija David Hästbacka
Avainsanat: teollisuus-SOA, suoritettava liiketoimintaprosessi, elinkaarisimulaatio, XML, BPMN, WS-BPEL, web-sovelluspalvelu, HTTP, DPWS, reaaliaikajärjestelmä

Tässä työssä tarkastellaan palvelukeskeisten liiketoimintaprosessien soveltamista teollisuuden käytön ja kunnossapidon toiminnoissa. Aluksi käsitellään teknologiaperhettä, jota käytetään tyypillisesti palvelukeskeisen arkkitehtuurin toteutuksessa. Sitten tarkastellaan palvelukeskeisen arkkitehtuurin periaatteita sekä palvelukeskeisiä liiketoimintaprosesseja. Sen jälkeen käydään läpi palvelukeskeisen ajattelun mahdollisuudet ja ongelmat teollisuusautomaatiossa. Työssä luodaan kaksi palvelukeskeistä liiketoimintaprosessia teollisuuden toimintoihin. Ensimmäinen niistä arvioi laitteiden aiheuttamaa ympäristökuormaa, kun taas toinen kerää kunnonvalvontatietoa laitteista. Kummankin toteutuksen yhteydessä arvioidaan käytettyjen tekniikoiden hyötyjä sekä kerrotaan, minkälaisia ongelmia työssä tuli vastaan.

Web-sovelluspalvelut (Web Services) ovat tekniikka verkon yli toimivan järjestelmän eri osien integroimiseen. Kommunikoinnin osa-alueiden abstraktiotasoa nostetaan eri teknologioiden avulla, jolloin saavutetaan yhteensopivuus palveluiden toteutusalustoista riippumatta. Web-sovelluspalveluiden perustekniikat keskittyvät rajapintojen ja tiedon kuvaamiseen, palveluiden löydettävyyteen sekä tiedonvälitykseen. Niiden ohella on joukko lisätekniikoita, joilla saadaan tuki esimerkiksi tietoturvalliseen tai tapahtumapohjaiseen viestintään.

Web-sovelluspalvelut ovat tavallinen, joskaan ei ainoa, tekniikka palvelukeskeisen arkkitehtuurin (SOA eli Service-Oriented Architecture) toteuttamiseen. Siinä hajautetun järjestelmän eri osien yhteistoiminta mallinnetaan palvelukutsuina. Palvelukeskeisen arkkitehtuurin tärkeimpiä periaatteita ovat esimerkiksi korkea abstraktiotaso, uusien palveluiden koostaminen olemassa olevia palveluita käyttämällä, löyhät kytkennät palveluiden välillä sekä palveluiden löydettävyys. Vaikka web-sovelluspalvelut antavat edellytykset palvelukeskeisen arkkitehtuurin periaatteiden noudattamiseen, huolellisella suunnittelulla on aina suuri merkitys.

Palveluita koostamalla voidaan luoda hierarkioita, joissa monimutkainen päämäärä saavutetaan suorittamalla joukko yksinkertaisia palveluita. Tällöin puhutaan palvelukeskeisistä liiketoimintaprosesseista. Nimestään huolimatta liiketoimintaprosessi ei välttämättä liity liike-elämän toimintoihin, vaan se voi tarkoittaa mitä tahansa sovelluslogiikkaa. Liiketoimintaprosessien logiikan kuvaamiseen on kehitetty kaaviokieliä, jotka ovat paitsi helposti ymmärrettäviä myös ilmaisuvoimaisia. Jotkin mallinnusohjelmat osaavat luoda kaavioiden perusteella koneellisesti suoritettavia liiketoimintaprosesseja.

Palvelukeskeinen arkkitehtuuri toisi joustavuutta ja skaalautuvuutta myös teollisuusjärjestelmiin. Perinteisen teollisuusautomaation ongelmana ovat laitteiden konfiguroinnin vaatima suuri työmäärä, järjestelmän eri tasoilla olevien laitteiden ja tiedon heterogeenisuus sekä eri valmistajien tuotteiden yhteensopimattomuus. Nykyiset palvelukeskeisen arkkitehtuurin toteutustekniikat ovat kuitenkin liian monimutkaisia pienitehoisiin laitteisiin, ja toisaalta reaaliaikaisuutta ei ole tähän asti toteutettu palvelukeskeisesti. Tutkimusta on tehty sekä palvelukeskeisten tekniikoiden resurssien kulutuksen vähentämiseksi että reaaliaikaisen palvelukeskeisen arkkitehtuurin saavuttamiseksi. Kokonaista järjestelmää, joka täyttäisi nämä vaatimukset, ei kuitenkaan ole tähän mennessä luotu.

Nykyisten palvelukeskeisten tekniikoiden rajoitteista huolimatta on monia teollisuuden toimintoja, joissa niitä voidaan hyödyntää. Esimerkkinä tässä työssä luodaan kaksi palvelukeskeistä liiketoimintaprosessia käytön ja kunnossapidon toimintoihin. Ensimmäinen niistä arvioi laitteiden tuottamaa ympäristökuormaa, kun taas toinen kerää kunnonvalvontatietoa laitteista. Liiketoimintaprosessit mallinnetaan graafisella tekniikalla, jolla luodut kaaviot muunnetaan koneellisesti suoritettavaan muotoon.

Laitteiden ympäristökuormaa arvioiva liiketoimintaprosessi käyttää julkista ympäristötietokantaa tarvittavan tiedon etsimiseen. Liiketoimintaprosessi mahdollistaa erilaisten laitteiden elinkaaren aikana syntyvien päästöjen vertailun. Liiketoimintaprosessissa käytetään erilaisia web-sovelluspalveluihin sekä suoritettavien liiketoimintaprosessien mallintamiseen liittyviä tekniikoita. Luotujen liiketoimintaprosessien ja ympäristötietokannan välisten yhteensopivuusongelmien vuoksi käytetään niin kutsuttua wrapper-palvelua, jolla palvelupyynnöt muokataan yhteensopivaan muotoon.

Kunnonvalvontaa suorittava liiketoimintaprosessi hyödyntää uutta web-sovelluspalveluille tehtyä laiteprofiilia. DPWS (Devices Profile for Web Services) vähentää laskentatehon tarvetta perinteisiin web-sovelluspalveluihin verrattuna. Laiteprofiilin avulla luodaan verkko, jonka rakenne voi muuttua dynaamisesti ilman erillistä konfigurointia. Liiketoimintaprosessiin integroidaan myös tehdasmalli, jota käytetään laitetietojen hallintaan. DPWS-palveluiden integroiminen liiketoimintaprosesseihin ei onnistu yhtä suoraviivaisesti kuin perinteisten web-sovelluspalveluiden. Osa yhteensopivuusongelmista ratkaistaan käyttämällä viestien käsittelyyn tavallisesta poikkeavaa lähestymistapaa, kun taas osa ratkaistaan wrapper-palveluita käyttämällä.

Luotujen liiketoimintaprosessien perusteella tehdään useita johtopäätöksiä. Palveluiden integroiminen liiketoimintaprosesseihin ei ole ongelmatonta: standardeista huolimatta asiakassovellusten ja palvelinten välillä voi olla yhteensopimattomuutta. Tällöin on käytettävä wrapper-palveluita. Käytetty mallinnusmenetelmä ja sen tekniikat säästävät kuitenkin aikaa verrattuna vastaavan sovelluksen kehittämiseen perinteisillä ohjelmistotekniikan menetelmillä. Tiedon saatavuus rakenteisessa muodossa helpottaa sen käyttämistä, ja palvelukeskeinen ajattelu vähentää integrointiin tarvittavaa työtä. Työssä todetaan myös, että hajautetussa sovelluksessa on olennaista minimoida verkon yli tapahtuvien palvelukutsujen määrä, koska ne pidentävät toimintojen suoritusaikaa.

PREFACE

This Master of Science thesis was written in the Department of Automation and Systems Engineering in Tampere University of Technology. It is a part of the CODES project (Computational Models in Product Lifecycle) funded by Tekes (the Finnish Funding Agency for Technology and Innovation). In CODES, the research focus was utilizing the information produced during the phases of the lifecycle of a product.

Several persons contributed this thesis work. The inspector, Professor Seppo Kuikka, provided valuable comments and points of view related to the contents of the work. The supervisor, Research Scientist David Hästbacka helped me to get into the practical part of the work and he also gave valuable feedback. I would also like to thank all the other members of the Automation Software Engineering research group as they have always been ready to help me. Finally, I would like to thank my intimates for their support during the work.

Tampere, 13th January 2011

Petri Kannisto

email: petri.kannisto@tut.fi

CONTENTS

ABBREVIATIONS, TERMS AND NOTATION

API	Application Programming Interface; the interface of a software entity through which it can be used by other software entities.
BPMN	Business Process Modeling Notation; a diagram technique to describe the flow of business processes and to visualize WS-BPEL.
Business Process	A set of actions with a specific result as the target.
communication protocol	The set of rules that guarantees the interoperation of applications.
DCS	Distributed Control System; a control system in which controlling is distributed to different devices.
DPWS	Devices Profile for Web Services; a technology profile that defines which features shall be used to make devices interoperable using Web Services.
ELCD	European Reference Life Cycle Database; an XML-based LCI database funded by the European Commission.
ERP	Enterprise Resource Planning; an information system to manage the resources of an enterprise.
exchange	In ILCD, an input or an output of a production process.
flow data set	In ILCD, an XML document that describes one material or product flow.
flow property data set	In ILCD, an XML document that describes one flow property such as mass.
HTML	Hypertext Markup Language; A language that is used to present web pages.
HTTP	Hypertext Transfer Protocol; a communication protocol that transfer information textually.
Intalio BPMS	A software product for business process modeling.
Intalio Designer	A development environment in which the business processes of Intalio BPMS are created.
Intalio Server	A server software product in which Intalio BPMS business processes are executed.
IP	Internet Protocol; a communication protocol whose purpose is to identify devices in a network.

ILCD	International Reference Life Cycle System; A specification that includes both an XML-based format to store LCI data and a book series that provides a framework for Life Cycle Assessment (LCA).
ISO 8859-1	A character encoding method.
Java	A language for creating computer programs that can be run on different platforms.
JAX-RS	Java API for RESTful Web Services; a Java library for creating Web Services which have a HTTP interface.
JAX-WS	Java API for XML Web Services; a Java library for creating Web Services and their clients using WSDL to describe interfaces.
JMEDS	Java Multi Edition DPWS Stack; a DPWS communications stack implementation made with Java.
LCA	Life Cycle Assessment; A methodological framework with which environmental impacts can be estimated.
LCI	Life Cycle Inventory; the work to resolve and to store information of the environmental impacts caused by products and processes.
MES	Manufacturing Execution System; an information system that is used to manage manufacturing in an industrial plant.
multicast	In communications, sending the same data to several recipients simultaneously.
OPC	Open Connectivity via Open Standards; a technology family that facilitates the integration of industrial devices to information systems.
OPC UA	OPC Unified Architecture; A specification whose purpose is to replace all the earlier OPC specifications and to provide one technology to cover the integration of production systems.
PLC	Programmable Logic Controller; a device that controls other devices by using logical sequences programmed into it.
process data set	In ILCD, an XML document that contains the LCI information related to one production process.
protocol analyzer	An application or a device to observe network traffic.
real-time task	A task that has a deadline for its completion.

REST	Representative State Transfer; an architectural style in which distributed capabilities are modeled as resources.
QoS	Quality of Service; in real-time systems, the amount of downtime and the number of failures that are permitted per time unit.
SCADA	Supervisory Control and Data Acquisition; a system used to monitor a production system.
SOA	Service-oriented architecture; a paradigm in information technology in which distributed resources or capabilities are modeled as services.
SOAP	A communication protocol on a high abstraction level that may be used on protocols such as HTTP and UDP.
soapUI	An application with which Web Services can be tested.
Tomcat	An open source web server for hosting web applications created with Java.
UBR	UDDI Public Registry; a public broker of Web Services that uses UDDI technology.
UDDI	Universal Discovery, Description and Integration; a registry-based technology to publish and discover Web Services.
UDP	User Datagram Protocol; a connectionless communication protocol.
UML	Unified Modeling Language; a set of diagram techniques to visualize the structure and the behavior of systems.
unicast	In communications, sending data to one recipient.
unit group data set	In ILCD, an XML document that contains the information related to one group of units related to one physical quantity.
URI	Uniform Resource Identifier; a string that identifies a resource.
URL	Uniform Resource Locator; a string that describes the location of a resource.
UTF-8	8-bit Unicode Transformation Format; a character encoding method.
Web Services	A technology to implement platform-independent communication in which resources or capabilities are modeled as services.

WS-BPEL	Web Services Business Process Execution Language; An XML-based language to describe the execution of business processes that interact with Web Services.
WS-Discovery	Web Services Dynamic Discovery; a technology for dynamic Web Service discovery inside a local network.
WSDL	Web Services Description Language; an XML-based language to describe service interfaces.
XHTML	Extensible Hypertext Markup Language; A version of HTML that is compliant to XML.
XML	Extensible Markup Language; a textual language that describes both the structure and the contents of data.
XML Schema	An XML-based language to describe the valid structure of XML documents. Alternatively, a document written in the XML Schema language.
XPath	XML Path Language; a language to address contents in XML documents.
XQuery	A language to query the data of XML documents.
XSLT	Extensible Stylesheet Language Transformations; a language to transform XML documents.

$\mathbf{C_R}$	Reference consumption
$\mathbf{E}$	Emission
$\mathbf{E_R}$	Reference emission
$\mathbf{F}$	Footprint
$\mathbf{n_C}$	Number of consumption items
$\mathbf{n_E}$	Number of emission items
$\mathbf{L}$	Life
$\mathbf{O_d}$	Operating time per day
$\mathbf{O_T}$	Total operating time
$\mathbf{O_y}$	Operating time per year
$\mathbf{t}$	Time
$\mathbf{t_L}$	Time required for a local service call
$\mathbf{t_R}$	Time required for a remote service call
$\mathbf{t_S}$	Time required for a service call (generic)

1	Exactly one
0..1	Zero or one
0..*	Zero or more
*****	Zero or more (same as above)

1. INTRODUCTION

The integration of industrial equipment to information systems is not always straightforward. During the past decades, the prices of computers have decreased and their capabilities have risen, which has made them a good platform to gather information from industrial systems. However, the structure of a production system is typically hierarchical and heterogeneous as different communication technologies are used to perform different tasks. Even though such a heterogeneous system meets its requirements from the production point of view, its integration to information systems is not straightforward. In addition, its installation and modification requires a lot of work. That is, it would be useful if there were a homogeneous technology that would on one hand facilitate integration and on the other hand require little configuration. A paradigm called SOA (Service-Oriented Architecture) could meet these needs.

In a distributed system, an important aspect is how different operations and resources are implemented and located. SOA suggests a solution with several advantages. In SOA, the capabilities of different actors are modeled as services. As SOA is an abstract concept rather than a technology, different authors describe its principles differently. However, the core semantics behind the descriptions is the same. From the point of view of this study, the most important principles are abstraction, loose coupling, discoverability and reusability. The abstraction level has to be high so that the implementation details of services are hidden and services running on different platforms can interoperate. Loose coupling means that the dependencies between services are few, and discoverability is important so that a service can be found when it is needed. If services are designed as generic as possible, they can be potentially reused in several applications. All of these principles could be an advantage also in industrial systems.

Two ways to create a SOA are used in this study. The most popular technology is Web Services that consists of several technologies and communication protocols used in the World Wide Web. The idea of Web Services is similar to the functions or the procedures of a programming language. However, Web Services can be located in a remote machine, and they also raise the abstraction level of communication to reach platform-independence. An alternative approach to Web Services is REST (Representative State Transfer) in which services are modeled as resources rather than as operations. Compared to Web Services, REST is simpler, and the most typical way to implement a REST architecture is to use a subset of the technologies used in Web Services.

In SOA, services can be orchestrated to form larger entities that perform more complex tasks than the services they compose. If such an entity has a service interface, it is called a service-oriented business process – again, such a business process can be composed into other business processes that perform even more complex tasks. Web Services and REST define the interaction between services, but they cannot define a flow of such a business process. Thus, languages such as WS-BPEL (Web Service Business Process Execution Language) have been created to describe the flow of business processes so that they can be executed by a machine. There is also a graphical notation called BPMN (Business Process Modeling Notation) to create diagrams which describe the flow of business processes. Some software products have even a function to create an executable business processes from such diagrams.

Despite all their advantages, the SOA implementations made this far cannot be used in all the functions of current industrial automation. The current technology cannot provide a SOA that meets hard real-time requirements, which is a necessity in several control systems. As there are functions such as condition monitoring that do not necessarily have strict real-time requirements, it is possible to use SOA for them and a different technology for time-critical functions. However, from the integration point of view, the optimum would be to have a single homogeneous way of communicating. Another problem is that the traditional industrial equipment does not provide enough computational capacity required to run Web Services or a similar technology. However, as SOA is expected to bring benefits, there have been several studies to overcome the challenges of industrial SOA.

In this study, two applications are created to demonstrate the use of SOA in the functions of operations and maintenance. First, a service-oriented footprint estimator application is introduced. Its purpose is to resolve the environmental footprint caused by a device during its life cycle using an environmental database that is available in the World Wide Web. Such an application can be used to compare the environmental footprints of different devices whenever there is a need for new equipment in an industrial plant. Another application created in this study is a service-oriented condition monitoring application. The application uses DPWS, a technology profile to facilitate the use of Web Services in devices.

The structure of this document is as follows. The theoretical background of the work is given by looking at Web Services related technologies and SOA (chapter 2), service-oriented business processes (chapter 3) and the work made this far for industrial SOA (chapter 4). The software solutions that were implemented during the work are presented in chapter 5 (footprint estimator) and chapter 6 (condition monitoring application). Finally, chapter 7 sums up the results and discusses future work in the context of this study.

2. WEB SERVICES AND SERVICE-ORIENTED ARCHITECTURE

A service-oriented architecture (SOA) could be implemented in several ways, and the Web Services technology family is one of them. First, this chapter introduces the technologies behind Web Services. Then, the principles of SOA are discussed after which Web Services are evaluated for them.

2.1 Introduction to Web Services

Web Services is a technology that enables applications to interoperate over a network, independent of the platform on which they are being run. A Web Service is realized by a *provider agent*, and the service is then used by a *requester agent*. Agents[1] communicate with each other by sending messages. It has not been defined how the functionality of an agent is implemented, and there is also no restriction about how the services provided by agents can be combined. [15]

As the interoperation between parties is of high importance in Web Services, the engagement of a requester to a provider is an essential process. As shown in figure 2.1, there are four basic steps. First, the parties (*requester entity* and *provider entity*) need to get known to each other (arrow 1). After that, human actors agree on the interface and the semantics of the service (arrow 2). However, it is also possible that the provider defines the service, giving the requester only the possibility to accept or to decline it. Then, both parties implement their agents to enable communication (arrow 3). Finally, the agents created by the parties can interact (arrow 4). [15]

Web Services are not limited to one specification. Additional specifications are, for example, related to reliable messaging over Web Services (WS-Reliability [50]), security (WS-Security [57]) or event-based communication (WS-Eventing [16]). All the additional specifications have their purpose, but they are outside of the scope of this document. However, the specifications that have a centric role in Web Services need to be known to understand the core functionality. According to Booth et al., the core technologies are WSDL, SOAP, UDDI, XML and HTTP (figure 2.2). WSDL describes service interfaces and makes them processable by machines. SOAP is used for communication between requester and provider agents, and it typically operates

[1]The agents of Web Services are not to be confused with the agents of artificial intelligence context.

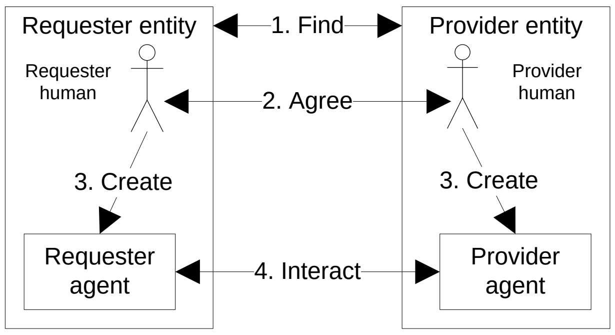


Figure 2.1: The basic steps of a Web Service engagement. [15]

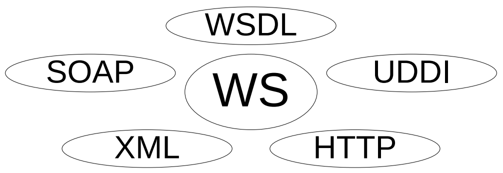


Figure 2.2: The core technologies used in Web Services (WS). [15]

over HTTP. XML is used to represent messages. UDDI provides the means to use the registry approach in service discovery. [15] The following section introduces these technologies one by one. As the benefit brought by UDDI has been questioned, another discovery technology, WS-Discovery, is also discussed.

2.2 Core Technologies of Web Services

2.2.1 XML

Extensible Markup Language (XML) is a language that describes classes of data objects. Such a description is called *XML document*, and it consists of *entities* (data storage units). To understand the main purpose of XML on a high level, entities called *element* and *attribute* have the highest importance. Each XML document must have at least one element, *root*, which wraps all the other elements (if any). Each element can have *child* elements, and each element except the root has one *parent* element. The structure that results is a tree. Elements can also have attributes, which are name-value pairs associated to an element. [19]

The syntax of XML can be demonstrated with a simple example. Let us imagine that the following information has to be provided: ”A product called ’milk’ is delivered by a vendor called ’Milk Ltd.’ The identification number of the vendor is 680492. The price of one unit of milk is 0.80 euros.” Code 2.1 presents this information in XML format. The first row begins the root element of the document (”product”). It has child elements ”name”, ”vendor” and ”price”. ”Vendor” has child elements ”title” and ”id”. Element ”price” has an attribute called ”currency”. The

```
<product>
  <name>Milk</name>
  <vendor>
    <title>Milk Ltd.</title>
    <id>680492</id>
  </vendor>
  <price currency="Euro">0.80</price>
  <!-- This is a comment -->
</product>
```

Code 2.1: An example of XML document.

row after "price" element is a *comment* that does not affect the way the document is processed, but it can be used, for example, to make the document more understandable to human readers by providing additional information.

XML has 10 design goals, including easy processing by software and easy understanding by humans, whereas terseness is declared less important. [19] As the example code shows, the syntax of XML is relatively easy to understand and its regular structure also suggests easy processability by software. However, as the relative amount of the actual payload in the document is low, its processing consumes apparently more computing power compared to a more compact data format.

Since the XML specification was released in 1998, no significant changes have been made to it. There are currently two versions of XML, namely 1.0 and 1.1. Most changes to version 1.1 are related to treating characters. For example, in XML 1.0, any character that is not allowed in names is forbidden, whereas in XML, 1.1 any character that is not forbidden is allowed. Almost all the changes make XML 1.1 less strict than 1.0, but some properties of XML 1.1 are stricter than those of XML 1.0. Thus, XML 1.1 is not completely backwards compatible. [18]

There are also additional XML related technologies that are defined separately from the main specification. The following paragraphs discuss three of them, namely XML namespaces, XML Schema and XSLT. Short overviews of XPath and XQuery are also given.

The motivation of **XML namespaces** is to make XML reusable. One XML document can be used by multiple applications, and, on the other hand, one application can use multiple XML documents simultaneously. Different documents can contain elements and attributes that have the same name but a different meaning. That is, there is a risk of name collisions, but it can be avoided with XML namespaces. [17]

There are two ways to indicate which namespace is being used. First, a namespace can be associated with a prefix which is then used as a part of any element names belonging to that namespace. Second, the contents of one element can be declared to use a *default namespace*, meaning that all the contents of that element with no namespace prefix use that namespace. Code 2.2 demonstrates namespace declara-

```
<elm xmlns="http://web.net/" xmlns:nnn="http://milk.com/srv/">
  <nnn:pack>02</nnn:pack>  <!-- NS: http://milk.com/srv/ -->
  <color>red</color>       <!-- NS: http://web.net/ -->
</elm>
```

Code 2.2: The ways of XML to indicate namespace usage. Modified from [17].

tions. As defined in the specification, the code uses URIs as namespace identifiers. [17] The namespace of element ”nnn:pack” is ”http://milk.com/srv/”, as prefix ”nnn” has been bound to that namespace. Respectively, the namespace of element ”color” is the default namespace ”http://web.net/”, as it has no namespace prefix.

XML Schema is an XML language that describes the structure and the constraints of an XML document. It defines, for example, what kinds of elements are allowed or required in different parts of an XML document, what kinds of attributes the elements have and what is the type of the contents of an element or an attribute. [90] The need for a way to describe the legal structure of an XML document is obvious, as XML itself does not offer means for it. With XML Schema, it is possible to check the *validity* of a document.

The basic semantics of an XML schema is created with two kinds of declarations: *type* declarations and *element* declarations. A type can be either *simple* or *complex*. [90] A simple type is inherited from a base XML Schema type such as integer, string or float, and it can define restrictions [11]. It can then be used to define the contents of an element or an attribute. A complex type can contain elements (or a sequence of them) or attributes. As simple type declarations concentrate on semantics, complex type and element declarations are used to define structures. A schema can also define references to other schemata which enables the reuse the contents of one schema in others. [90]

Code 2.3 shows an XML schema document that defines some typical elements. The schema has been simplified to draw attention on the semantics on a high level. After declaring one simple and one complex data type, the schema defines the structure of the document using the ”element” tag. Even though only one element is used, the structure of a document compliant with the schema is not necessarily simple. The type of the element can be any simple type, but it can also be a complex type, defining more complex structures. XML schema supports XML namespaces [90], but they have not been used in the example for the sake of simplicity.

To address the contents of XML documents, a language called **XPath** has been developed. Using XPath, an XML document can be processed as a tree whose contents can be referred to unambiguously. The result of a reference can be an element, an attribute, the text contained by a node or a group of nodes. In addition, XPath defines a set of functions with which the data of XML documents (such as integers, strings and booleans) can be manipulated. The current version of XPath is

```
<schema>
  <simpleType>
    <restriction></restriction>
  </simpleType>
  <complexType>
    <sequence>
      <element></element>
    </sequence>
  </complexType>
  <element></element>
</schema>
```

Code 2.3: A simplified XML schema that defines a simple type, a complex type and an element. Modified from [90].

2.0 (released in 2007) and it extends the functionality of the original XPath 1.0. [10] The importance of XPath is high because it is used by XML processing languages XSLT and XQuery.

XSLT (Extensible Stylesheet Language Transformations) is a technology to transform XML documents. XSLT documents are written in XML, and to make a distinction between the XML names of XSLT and other XML names, XSLT has its own namespace. An XSLT document is called ”stylesheet” because one purpose of XSLT is to describe the formatting used to show the contents of an XML file. However, the use cases of XSLT are not limited to visual purposes. The possible output formats of an XSL transformation are XML, HTML, XHTML and text. Two versions of XSLT have been released: 1.0 in 1999 and 2.0 in 2007. [53]

An essential aspect of XSLT is the way the transformed document is written. The elements and the attributes of the source document can be copied as they are, but the resulting document can also be completely different from the source document. New elements and attributes can be created, and as the output format is defined as text, the resulting document may contain anything. [53]

XSLT defines several structures that have analogies with procedural programming languages. *Functions* and *templates* are sections of code that may be used from anywhere in the document. The entire stylesheet can have parameters whose values are set externally when the stylesheet is executed. *Variables* can also be defined, but they are not variables in the sense that once the value has been initialized, it cannot be changed anymore. Conditional execution can be defined using constructs *if* or *choose*, and if there is a sequence of similar elements in the source document, they can be processed using the *for-each* construct. To have more control over the output of for-each, such elements may be sorted according to a condition. As the basic mathematical operators can be used in XSLT, it is also possible to perform calculations. XSLT uses the functions and operators of XPath – that is, everything that works in XPath works also in XSLT. [53]

```
<template name="SomeTemplate">
  <param name="SomeParam"></param>
  <element name="SomeElement">
    <value-of select="SomeParam"/>
  </element>
</template>

<call-template name="SomeTemplate">
  <with-param name="SomeParam">This is some text</with-param>
</call-template>
```

Code 2.4: An example of an XSLT template and its call. Namespace prefixes have been left out. Modified from [53].

Code 2.4 shows an example of using a template in XSLT. First, the template is defined: it will print an element with the name ”SomeElement” and the value of the parameter ”SomeParam” inside it. After the definition of the template, it is called using ”This is some text” as the value of the parameter. After the template has been defined, it could be called from anywhere in the document.

To retrieve data from XML documents, there is also a query language called **XQuery**. Unlike XSLT, which has a transformation point of view, XQuery processes XML documents as if they were databases. However, XSLT and XQuery specifications have been developed in parallel, and XQuery has also several similarities with XSLT. Like XSLT, XQuery can define structures such as conditional execution (*if* expression), functions and variables. In addition, XQuery uses the complete set of XPath functions and operators which makes the use of several low-level actions such a string manipulation and calculations similar to XSLT. The current version of XQuery is 1.0, and it was released in 2007. [13]

2.2.2 SOAP

SOAP is a lightweight communications protocol for a distributed environment that does not make any assumptions about the communications protocol lying under it. [43] However, the specification of SOAP defines a binding only for HTTP [44]; it refers to that HTTP is most typically used. The current version of SOAP is 1.2. Compared to its predecessor SOAP 1.1, the syntax of SOAP 1.2 has been changed, and it also defines additional semantics. Moreover, unlike the acronym ”SOAP” stood for *Simple Object Access Protocol* in version 1.1, it is not spelled out anymore in version 1.2. [63]

The specification of SOAP has only a few requirements about the message structure. SOAP messages are represented in XML 1.0. All the contents of a message is wrapped by *Envelope* element which contains one *Body* and at most one *Header* element. The purpose of Header is to contain any information that is context-specific

```
<Envelope>
  <Header></Header>
  <Body></Body>
</Envelope>
```

Code 2.5: The basic elements of a SOAP message. [43]

(for example, related to the path the message is routed along). The contents of Header may be modified by any intermediary while the message is on its way from the origin to the destination. Any data that is meant to be processed only by the final recipient is included in Body element, which is obligatory. The contents of Body must not be modified during message transportation. The specification of SOAP does not define what kind of information is obligatory in Header and Body; that is, the contents are subject to any technology that lies over SOAP. [43]

Code 2.5 shows the basic structure of a SOAP message. Despite its simplicity, there are several additional mechanisms that are optional in a message. For example, a *header block* (an element in Header) could contain the *mustUnderstand* attribute to indicate that the header block must be processed by a node that receives the message. Another example is the attribute *role* that indicates to which node a particular header block is targeted. [43]

2.2.3 WSDL

WSDL (Web Services Description Language) is an XML-based language that describes Web Services. It has the means to describe both the abstract functionality of a Web Service and its concrete details such as underlying technologies and service endpoints. [22] The two main versions of WSDL are 1.1 and 2.0, released in 2001 [23] and in 2007 [22].

Code 2.6 shows the basic elements of a WSDL 1.1 document and a WSDL 2.0 document. The documents have been simplified to demonstrate only the basic structure, but they show significant differences between the specifications. The purpose of the elements is described in table 2.1. The differences between versions 1.1 and 2.0 are mostly related to the way things are expressed. For instance, ”message” elements have been replaced with type definitions, and former ”portTypes” are now called ”interfaces” [27]. The way of defining operations has also been changed, as *operation types* have been replaced with *message exchange patterns.* [67, p. 168].

Both WSDL 1.1 and WSDL 2.0 are flexible about schema languages and communication protocols. Both define XML Schema as the primary content model definition language, but the possibility to use another schema language is mentioned in both specifications. Moreover, both versions define a binding to use SOAP and

```
<definitions>                <description>
  <import/>                    <documentation></documentation>
  <documentation>              <import></import>
    </documentation>           <include></include>
  <types></types>              <element></element>
  <message></message>          <types></types>
  <portType></portType>        <interface></interface>
  <binding></binding>          <binding></binding>
  <service></service>          <service></service>
  <!-- ext. elem. -->        </description>
</definitions>
```

Code 2.6: The basic elements of a WSDL 1.1 document [23] (left) and WSDL 2.0 document [22] (right).

Table 2.1: The basic elements of a WSDL 1.1 document [23] and a WSDL 2.0 document [22]. Even though same explanations are used for some version 1.1 and 2.0 elements, there may be differences on a detailed level. In WSDL 2.0, the concept *component* covers elements "interface", "binding", "service", "element", "types" and "description". [22].

Element (occurrences)	**Purpose**
binding (0..*)	**1.1, 2.0:** Describes the data format and the communications protocol required to access a service.
documentation (1.1: 0..1) (2.0: 0..*)	**1.1, 2.0:** A human-readable documentation about the purpose of an element. A documentation can be inside any WSDL element. In 2.0, the documentation can also be machine-processable.
element (0..*)	**2.0:** Declares the name and the content model of an element.
(ext. elem) (0..*)	**1.1:** *Extensibility elements* are used to extend the language.
import (0..*)	**1.1:** A mechanism to import elements that are defined in another document. **2.0:** A mechanism to import *components* that are located in another file and have a different namespace than the including document.
include (0..*)	**2.0:** A mechanism to include *components* that are located in another file and have the same namespace as the including document.
interface (0..*)	**2.0:** Defines an interface for a service.
message (0..*)	**1.1:** A format definition for data used in communication.
portType (0..*)	**1.1:** A set of operations offered by endpoints.
service (0..*)	**1.1:** Groups a set of related *ports* together. **2.0:** Defines a service. The service is associated to an "interface" and to a "binding", and its address can also be defined in this element.
types (0..1)	**1.1, 2.0:** A collection of data type definitions.

HTTP as the communication protocols, but it is possible to define bindings for other protocols as well. [22; 23]

As WSDL can describe all the information required to consume a Web Service, the structure of a WSDL document can be complex. However, the complexity is not necessarily a problem from the application developer point of view. There are software products both for the automatic generation of WSDL documents and for their automatic processing. Thus, when a service or a service consumer is being developed, it is not necessarily required to know the details of WSDL.

2.2.4 UDDI and WS-Discovery

There are several technologies that could be used in Web Service discovery. This subsection discusses UDDI and WS-Discovery.

Universal Discovery, Description and Integration (UDDI) is a technology to publish and discover Web Services. It provides information about service providers, services and the interfaces of the services being offered. UDDI is based on HTTP, XML, XML Schema, SOAP and WSDL. There are three versions of UDDI, namely 1.0 (released in 2000), 2.0 (2001) and 3.0 (2004). [49]

The features and functionality of UDDI have received critics. At the time UDDI was being developed, there was a vision of UDDI Business Registry (UBR), a public Web Services broker that would facilitate their discovery in Internet [49]. Several major IT companies implemented a UBR, but as far as is known, all of them have been abandoned. According to a website of Microsoft, the UBR project of IBM, Microsoft and SAP proved the ”interoperability and robustness” of UDDI [92]. However, Stollberg & Fensel claim that public UDDI registries were abandoned as the publishing and the searching functionality of UDDI was insufficient [89]. Another author criticizing the search functionality is Mintchev. He claims that when using UDDI as a private service registry, the insufficient features of the classification system encourage the usage of proprietary extensions. [62] Finally, UDDI is said to be too complex considering the functionality it offers [70].

It remains unclear how much value UDDI can bring to a Service-Oriented Architecture. Even though public UBR services do not currently exist as far as is known, UDDI could still be used in a narrower context such as inside one organization. However, another technology could bring similar functionality with less work and less complexity.

Another discovery solution for Web Services is WS-Discovery (Web Services Dynamic Discovery). Its current version is 1.1, and it was released in 2009. [69]

To discover services in a simple way, WS-Discovery provides an *ad hoc mode*, in which the services in a network can be located by two means: either by service type or service name. When a consumer wants to discover a service of a specific type, it

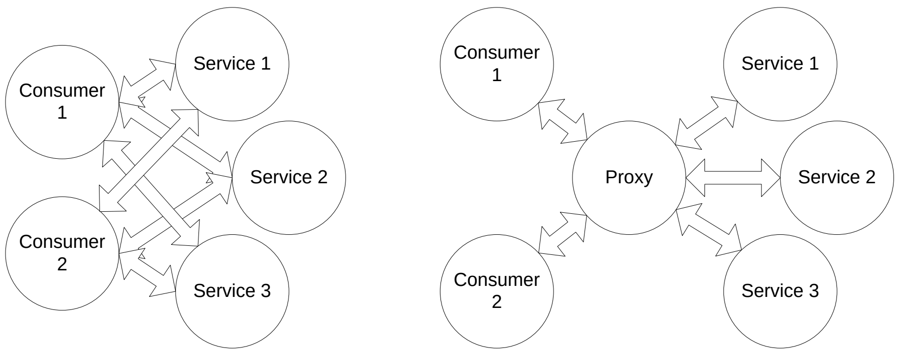


Figure 2.3: Discovery using the ad hoc mode (left) and the managed mode (right). Based on [69].

sends a *probe message* to a *multicast* group in its network. The services that match the requested type send a response to the consumer. When searching for a service by name, the consumer sends a *resolve message*, and the functionality is similar. To reduce the need to poll for services in the ad hoc mode, a service that joins the network sends a multicast *hello message* so that any potential clients in the network are aware of the new service. Moreover, when a service leaves the network, it sends a multicast *bye message* to indicate it is leaving. [69]

To enhance scalability and to make services reachable outside an ad hoc network, WS-Discovery provides a *managed mode* in which a *discovery proxy* is used. In the managed mode, services and consumers do not send their messages directly to each other but to a discovery proxy. If the discovery proxy detects any multicast probe or resolve messages, it sends an announcement about itself which makes consumers switch themselves to the managed mode. Moreover, if a discovery proxy does not respond to the messages sent by a consumer, the consumer switches itself automatically to the ad hoc mode. [69] The ability to change the discovery mode automatically makes networks more flexible and robust, and they have no single points of failure when discovery is concerned. Figure 2.3 illustrates the principles of the two discovery modes.

It has not been specified which communication protocols shall lie under WS-Discovery. However, the specification defines the usage of IP multicast and SOAP-over-UDP for multicast messaging and SOAP-over-HTTP for unicast messaging. [69]

WS-Discovery is oriented to local networks. Its intention is to require minimal networking services, and its goal is *not* to be scalable over Internet. [69] However, according to a study, it is possible to build additional layers on WS-Discovery by configuring discovery proxies, which results in better extensibility [78].

The approaches of UDDI and WS-Discovery are different, and neither of the technologies brings a universal discovery solution. UDDI is registry-oriented and suits better for manual use by humans as it provides a storage for service documentations; WS-Discovery is peer-to-peer oriented, and its model is lighter, suggesting automatic discovery of services whose type is known beforehand. During its existence, UDDI has received a lot of criticism, and it is controversial how much advantage it can bring. In contrast, WS-Discovery is currently a young technology, and it remains to be seen whether it will be a success.

2.3 REST

The traditional Web Services technology family is complex: several specifications such as WSDL and SOAP are required even when a simple service is created. However, an approach called REST (Representational State Transfer) suggests a simpler alternative. According to Fielding, REST is ”an architectural style for distributed hypermedia systems”. Its motivation is to capture the principles of the technologies behind World Wide Web, as it has turned out to be successful and scalable. [37]

REST has six principles: client-server model, statelessness, cache, uniform interface, layered system and code-on-demand. The key point of *client-server model* from the REST point of view is the separation of user interface from data storage. *Statelessness* means that any state information is always stored by clients, never by servers. *Caching* means that any information that has been labeled cacheable can be stored by clients to reduce network traffic. *Uniform interface* results in a simple system architecture and it also makes interfaces less coupled to the service behind them. *Layered system* makes it possible to hide system functionality, showing only the utmost layer to the user. The last principle, *code-on-demand*, is optional, and it allows a client to download code from a server and execute it on its own machine. [37, pp. 78–85]

Even though REST has several principles, the attention is typically on interfaces in the Web Services context. In fact, a service that is designed in the REST style commonly refers to a service that has an HTTP interface even though REST is not restricted to HTTP. According to Bean, HTTP supports several commands, but in the scope of Web Services, the most important ones are the CRUD commands. The CRUD of HTTP are POST (”create”), GET (”read”), PUT (”update”) and DELETE (”delete”). These four commands provide all the functionality required to utilize a resource. [9, p. 49] That is, as traditional Web Services define a custom interface with WSDL, the interface of REST-style services is always the CRUD commands set.

It is typical that SOAP and REST are considered opposite ways to implement a Web Service. This is not necessarily the case as SOAP is a communications protocol whereas REST is an architectural style. Moreover, the requests in SOAP version 1.2 can be sent as HTTP requests [44] which actually enables the use of SOAP in the REST style. Thus, it would be more descriptive to use terms ”resource-oriented design” for REST and ”operation-oriented design” for the traditional SOAP as used in [64, pp. 24–26].

There has been a lot of debate whether REST has advantages over SOAP in Web Services. Operation-oriented SOAP usage is said to result in needless complexity and even reduce interoperability. It has also been stated that it is needless to raise the abstraction level higher than HTTP. On the other hand, the simplicity of a REST interface is said to be problematic in complex operations, resulting in a high number of operation calls compared to a customized interface. [42]

Some authors find advantages in both technologies. According to Bean, several successful REST-style service implementations prove the effect of REST. However, a REST-style service can also be less flexible when it is about changing the service, requiring either requesters change their applications or the service provider to maintain several service versions. [9, p. 50] According to zur Muehlen et al., REST services cannot be debugged during development while SOAP services can. In addition, the customizability of a SOAP-style interface makes it able to raise the abstraction level, requiring fewer objects to be called by a service requester.

REST is also said to be looser coupled than SOAP-based design [65]. The claim refers likely to service interfaces as REST does not require WSDL or similar to declare an interface. As interfaces are always the same (”CRUD”), less configuration is required to set up a service consumer. However, if REST refers to XML-over-HTTP messaging as typical, the structure of XML messages has to be known by the consumer anyway. There is also a language for this purpose. According to Hadley, WADL (Web Application Description Language) is a language to ”provide a machine processable description of HTTP-based Web applications”. WADL can describe the parameters required to call a specific HTTP interface, and it also associates data types (for example, defined in an XML schema) to requests and their return values. [45] That is, WADL could be considered the WSDL of REST, but WADL is simpler due to the simple nature of HTTP-based services. However, as of WSDL 2.0, even WSDL can be used to describe a HTTP interface [21].

Neither operation-oriented nor resource-oriented (REST) design can be declared the winner. There are valid arguments for and against both of them, and the importance of an argument depends on the context. Thus, when a service is being designed, it is meaningful to select carefully between resource-orientation and operation-orientation. Neither of the choices is likely to ruin an architecture, but choosing the better alternative may result in less work in the design phase.

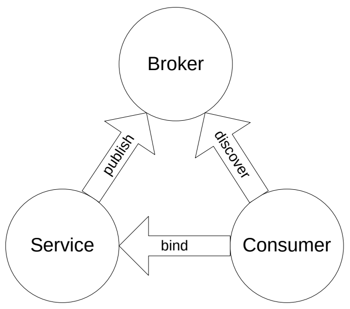


Figure 2.4: The roles in SOA. [24, p. 66]

2.4 Basic Principles of SOA

Web Services may bring several advantages to a design of a distributed system. However, similar advantages could as well be reached with another technology if it only follows the same principles. They are captured by an architectural model called Service-Oriented Architecture (SOA). As SOA is an abstract term, there are different definitions for it. MacKenzie et al. define SOA as follows:

> Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. [60, p. 8]

It is unclear who first introduced the term ”service-oriented architecture” [9, p. 1]. However, it is said that service-oriented thinking has its roots in different distributed component technologies (that do not include Web Services) [8; 81]. Despite that, Web Services is the most typical technology to implement SOA.

On an abstract level, the basic model of SOA is simple. In SOA, actors have capabilities that they offer to other actors by setting up *services*. The ones offering the services are called *service providers*, and the users of the services are called *(service) consumers* respectively. [60, p. 9] To use a service offered by a service provider, a consumer has to *discover* it first. Thus, the SOA model receives its third role: *(service) broker* serves as a registry for the services. [24, p. 66] In principle, it is not necessary to have a broker, but discovering a service for a specific purpose without a broker turns difficult if there is a high number of different services. [9, p. 12] Figure 2.4 shows the roles and their actions in the model: after a service has been published via a broker, consumers can first discover it using the broker and then bind to it for usage.

Services are suggested to have three key characteristics: visibility, interaction and (real world) effect (figure 2.5). *Visibility* means that the provider of a service and the ones that are using it can find each other. One aspect of visibility is description, meaning that the users of a service need to know which conditions shall be met so that the service can be used. *Interaction* covers the actions that are involved in the

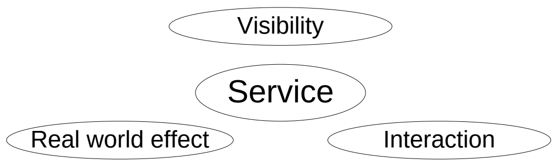


Figure 2.5: The key concepts of a service. [60, p. 13]

Table 2.2: The principles of service-oriented design as they are listed by Bean [9, p. 25] and Erl [33].

Principle	Bean	Erl	Meaning
Abstraction		x	The implementation details of a service are hidden as far as possible.
Autonomy		x	Services need control over their execution environment to be more reliable.
Composability		x	Services can be composed into larger and more complex entities to complete more complex tasks.
Discoverability	x	x	Consumers can find enough information about services so that they can understand their purpose and connect to them.
Governance	x		A framework is used to make sure that the other SOA principles are met. According to Erl, governance is not a SOA principle: it is a term that covers the management of different services owned by an enterprise [34, p. 88].
Interoperability	x		The interaction between the parties of a service is possible regardless of the technologies that lie under them.
Loose coupling	x	x	Dependencies between services are as few as possible, still making interoperability possible. There are no direct physical dependencies.
Reusability	x	x	Services are designed generic enough so that their reuse potential is as high as possible.
Standardized service contracts		x	Services describe their capabilities and requirements in a standardized way.
Statelessness		x	Statefulness is avoided so that services preserve their availability and scalability.

usage of a service. It is typical that the interaction is a series of message submissions which are followed by actions that the messages invoke. *Effect* is what happens as the result of the interaction in ”the real world”. The effect may, for instance, be a receipt of requested information or a change in the state of the parties that are involved in the interaction. [60, pp. 13–18]

The key principles of service-oriented design are defined differently depending on the author as shown in table 2.2. Even though there is no conflict between these two definitions lists, their differences point out that the principles of SOA do not have any generally accepted exact definition. Besides, the existence of such lists indicates that creating a service-oriented design may not be straightforward or simple.

Another popular programming paradigm, Object-Oriented Programming (OOP), has common characteristics with SOA. However, the differences between the paradigms are obvious. Both of them intend to represent the real world and both of them also gather data and related operations to one place. However, the units (objects) of OOP must be created at runtime whereas the units (services) of SOA exist ”somewhere else”. Moreover, the operations in OOP are considered to be a part of an object that is being called whereas services do not refer to a direct connection with any object. [60, p. 10] It is obvious that the strength of OOP is the suitability for tight coupling whereas SOA is more scalable and better suitable for an environment where instances are more autonomous. It can be said that the two paradigms are not in any case rivals to each other but they are rather complementary. OOP suits for organizing the functionality of one application whereas SOA suits for organizing the interoperation of applications.

A design based on SOA is suggested to have several benefits. In general, the distributed nature of SOA suits to communication between organizations. Moreover, the integration of two different SOA systems is supposed to require less work than the integration of two non-SOA systems. As a SOA system makes few assumptions on the underlying network, it is scalable. [60, p. 11] According to Cohen, SOA provides also easier retrieval of existing data and easier software maintenance compared to traditional approaches. [24, pp. 10–11] To verify the benefits of a specific SOA implementation, it would be useful if its return on investment (ROI) could be calculated. Some methods have been developed to perform the calculation [59; 77]. However, it has been claimed that it is not even sensible to try to calculate a ROI as SOA cannot be considered a separate element of a system [61]. That is, it is difficult to point out concretely how big the benefits of SOA are. However, the points suggesting advantages are well argued.

2.5 Evaluation of Web Services for SOA

Web Services can be used as the platform of service-oriented design. They offer a rich functionality, but some questions remain: what are the problems of Web Services from the system point of view, and do they really meet the principles of SOA?

Performance and response times are a potential challenge in Web Service applications. The use of XML is an advantage from the extensibility point of view, but according to Cohen, the processing of large XML documents is resource-consuming. However, the final performance can be affected by design-time choices such as which SOAP encoding style is selected. [24, pp. 76–80] Performance can also be improved by message optimization [9, pp. 334–337]. It is obvious that the predictability of Web Service response time is a challenge especially when multiple services are composed in a multi-user network. It is a result of the loosely-coupled and the distributed nature of the technology. However, there are application areas in which response times and their predictability are not of the highest interest.

Table 2.3: Discussion how Web Services meet the principles of service-oriented design.

SOA principle	**Response of Web Services**
Abstraction	WSDL and SOAP enable a high abstraction level.
Composability	Web Services can be composed to form more complex entities that solve more complex problems.
Discoverability	UDDI or WS-Discovery could be used to achieve this.
Interoperability	WSDL interfaces are used similarly whatever the application that performs service functionality.
Loose Coupling	From the connection point of view, both WSDL and HTTP provide loose coupling.
Reusability	Web Services can be reused if they are designed carefully enough (also, the lower is the granularity of a service the higher is its reuse potential).
Standardized service contracts	WSDL provides an established way to describe the capabilities and requirements of a service.
Statelessness	HTTP and SOAP are stateless, but stateful properties can be designed over them.

Table 2.3 discusses how the features of Web Services meet the design principles of SOA that were introduced in table 2.2. The principles that are considered completely design-specific (abstraction, autonomy, and governance) are not included. Even though HTTP could be, in principle, replaced by another communications protocol, its usage is assumed. To conclude the table, Web Services offer the means required for service-oriented design. However, as most of the principles discussed are more or less dependent on design, using Web Services as the architecture basis does not guarantee a ”well-designed” SOA. Moreover, it depends on the context whether UDDI or WS-Discovery can offer a sufficient discovery functionality.

It can be said that the features of Web Services technology provide a platform for a service-oriented design. However, it is typical in information technology that careful design is required to avoid problems. It is also important to recognize the limitations of the technology. There are some trade-offs such as the loss of predictable response times due to loose coupling and high resources consumption due to high abstraction level. Whether these trade-offs cause problems, depends on the context.

2.6 Conclusions

In this chapter, different technologies that belong to the Web Services technology family were discussed. Then, the features of Web Services were evaluated for the principles of SOA paradigm.

Web Services is a technology family to implement the communication between the parts of a distributed system. The interaction of applications is modeled as service calls, and the abstraction level is high so that platform-independence is reached. In the world of Web Services, XML is one of the most important technologies. It is intended to be both human-readable and machine-processable, and its simple basic principles can be extended to create complex semantics. As a result, XML is used as the basis for several other languages such as XSLT, SOAP and WSDL. XSLT is used to transform XML documents. SOAP and WSDL belong to the most important Web Service technologies: SOAP raises the abstraction level of communication, and WSDL is a platform-independent language to define service interfaces.

Two different Web Service discovery technologies were discussed. UDDI is intended to be used by humans in large networks. It has been questioned whether UDDI brings enough value compared to its complexity. In contrast, WS-Discovery brings a dynamic approach in which discovery is performed automatically by machines inside a local network.

An alternative architecture for service interaction is REST in which all the services have a similar interface. In Web Services, it typically means the use of XML-over-HTTP instead of SOAP-over-HTTP and WSDL for interface definitions. There has been a lot of debate whether REST is a better approach, and no universal result can be declared.

SOA is a paradigm in which the capabilities of the parts of a distributed system are modeled as services. SOA has different principles including abstraction, interoperability, loose coupling, reusability and discoverability. SOA suggests several benefits such as scalability, easy integration and easy maintenance. Web Services is one alternative to implemented a SOA as it enables an architecture in which the principles of SOA are followed.

3. BUSINESS PROCESSES

One of the key principles of SOA is that a service can be composed to be a part of another service. This way, complex actions can be accomplished by executing several simple services. If the result of such a composition has a certain business result, it is called orchestration, and the resulting service is called a service-oriented business process. Such a business process is also a service itself, which means that it can be orchestrated to be a part of even more complex business logic. First, this chapter describes the basic concepts related to service composition after which business processes are introduced. Then, business process technologies are observed, and finally, an overview of business process modeling software is given.

3.1 Service Composition

Composability is one of the key principles of SOA. Laskey et al. define service composability as follows: A service can be either *atomic* or *composite*. An atomic service is used via a single interface and its functionality is not dependent on other services. A composite service is also used via a single interface, but as it interacts with other services, its functionality depends on them. The services that a composite service uses may also be composite, enabling multi-layered compositions. [56, p. 74] According to Erl, the key idea behind composition is that it makes it possible to use several small solutions of small problems to create a big solution to solve a big problem [33].

Figure 3.1 shows an example of an atomic and a composite service. S1 is an atomic service whereas S2 is a composite service as it uses services S3 and S4. S3 is also a composite service as it uses S5.

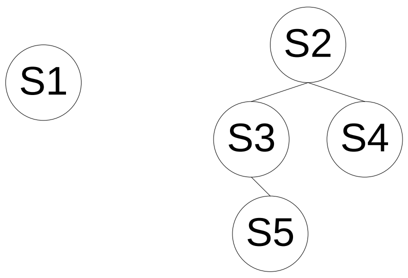


Figure 3.1: An atomic service (S1) and a composite service (S2).

Table 3.1: A general comparison of fine-grained and coarse-grained services. [9, p. 284]

Aspect	**Fine-grained services**	**Coarse-grained services**
Optimization	Simpler	More complex
Reuse potential	Higher	Lower
Expected amount of work on a change	Lower	Higher
Number of services to manage	Higher	Lower
Number of services to develop	Higher	Lower

The term *granularity* is important in composition. According to Erl, granularity describes the complexity of an item which can be, for instance, a data item or a service. If the granularity of a data item is low, it means that it contains only a small number of atomic information units whereas coarse granularity refers to a higher complexity. Correspondingly, service granularity refers to the complexity of the actions that a service does. [32] As table 3.1 shows, service granularity has a significant effect on service characteristics. In general, a fine-grained service is easy to optimize, its reuse potential is high, and the required amount of work on system changes is low. However, fine granularity requires typically more services to be developed and managed. Thus, it is important to find a compromise that minimizes any granularity disadvantages. [9, p. 282–284]

Services can be composed hierarchically to form *service-oriented business processes.* A business process is a set of activities that gives a certain business result when it is executed in a logical sequence. The composition of such a business process is called *orchestration.* [56, pp. 75–76] Figure 3.2 shows an example of a service-oriented business process. The business process itself is used via a service interface (Service A), and during its execution, it calls another service (Service B) if the business process logic considers it appropriate. According to Laskey et al., orchestration is typically made using an orchestration scripting language which makes it possible to run the business process in an orchestration engine. The orchestration engine has the role of a *conductor*: it is a single agent that coordinates the entire business process. [56, p. 76]

The interaction between business participants is referred to as *business collaboration.* If a business collaboration is service-oriented, its composition is called *choreography.* Figure 3.3 shows an example of a choreography where business processes of two different organizations collaborate with each other. The main difference between orchestration and choreography is that a choreography is not coordinated by a single conductor. Instead, participants run their own business processes that interact with each other. However, like orchestration, choreography is typically implemented using a scripting language. [56, p. 77–78] A collaboration is not necessarily coop-

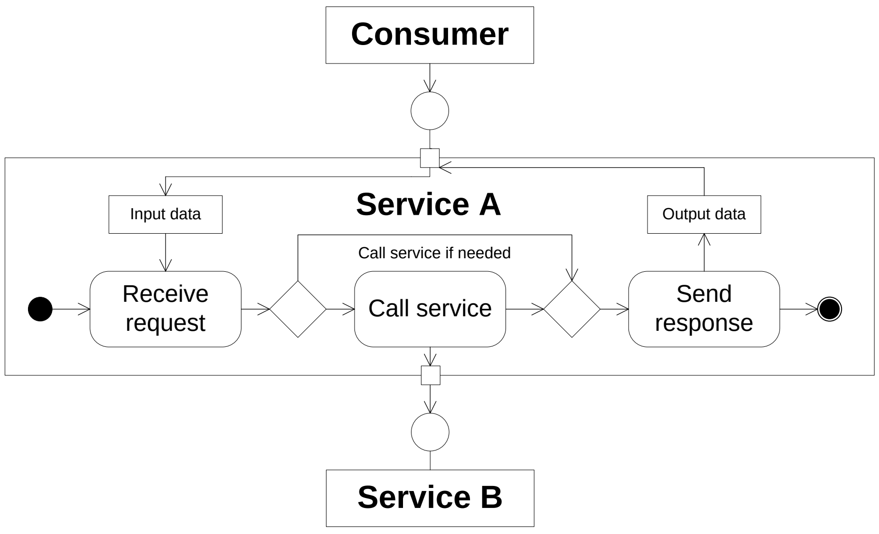


Figure 3.2: An example of service-oriented business process. [56, p. 77]

erative: the participants can also compete with each other. For example, auction scenarios have been discussed in [6; 26; 96].

Figure 3.4 summarizes the relationships between composition-related concepts. A composite service contains other services and it is also a service itself. Creating composite services is called composition. The creation of service-oriented business processes is called orchestration, and a service-oriented business process is a composite service. The creation of service-oriented business collaborations is called choreography, and service-oriented business collaborations contain service-oriented business processes. A service-oriented business collaboration can be either cooperative or competitive by nature.

On an abstract level, composability is a powerful feature of SOA. Services can be created generic to be reused and composed to several service-oriented implementations. Together with loose coupling, composability enables an efficient way to distribute functionality and to enable communication between different parties.

3.2 Service Layers and Models

As service compositions can lead to complex hierarchies, it may be necessary to categorize services to have control over them. In this section, two different methods of categorization are discussed: service layers and service models.

According to Erl, business process logic can be divided into *business logic* and *application logic*. Business logic describes the logic of a business process whereas application logic is an "automated implementation" of it. Business logic is located in a layer called *business process layer* which is the topmost layer of a business process. Application logic is in *application layer* which is the downmost layer of a business process respectively. Service-oriented design results in a third layer called *service*

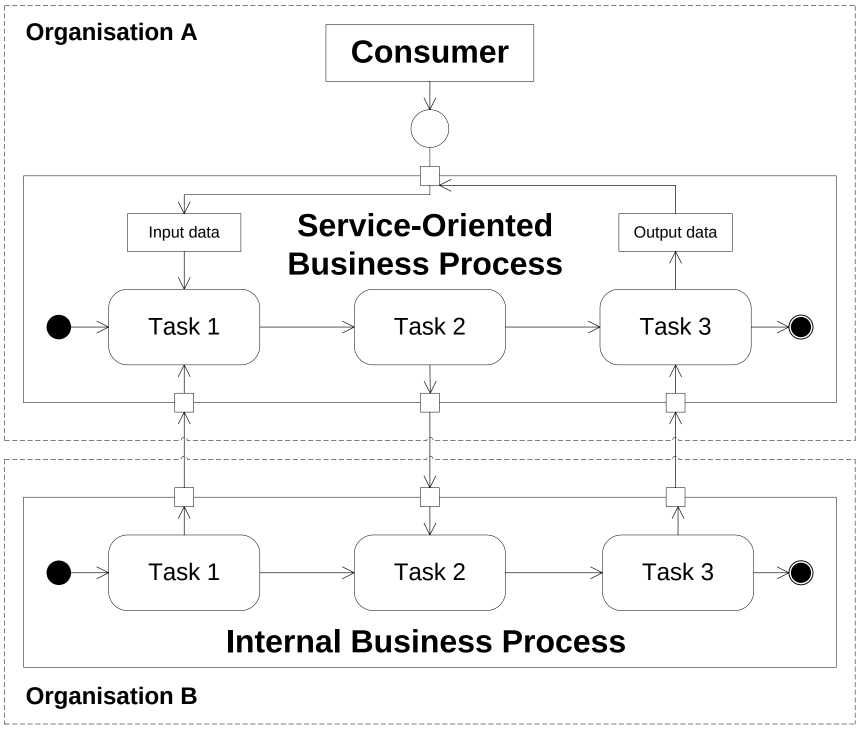


Figure 3.3: An example of service-oriented business collaboration. [56, p. 78]

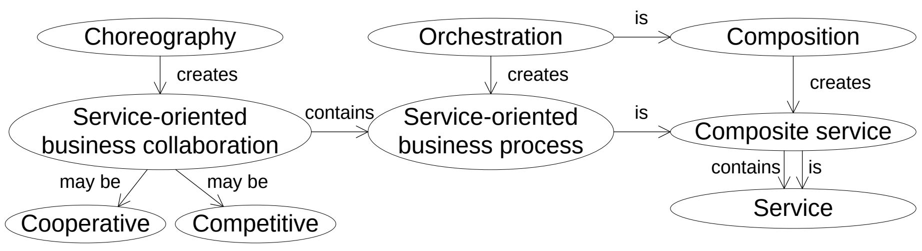


Figure 3.4: The concepts related to choreography, orchestration and composition.

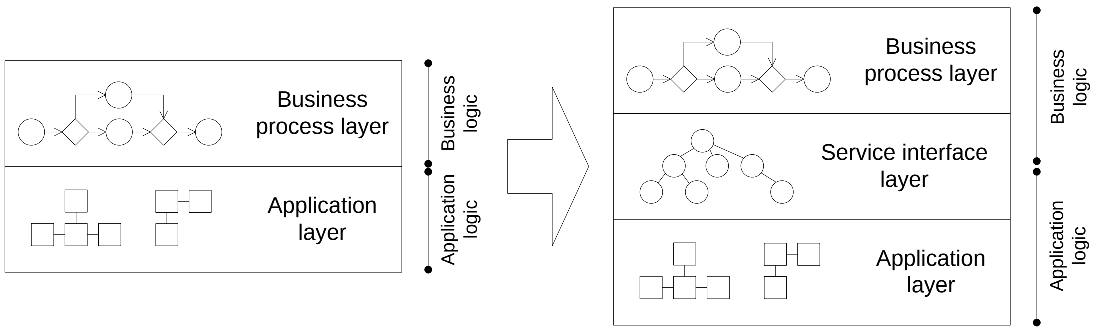


Figure 3.5: The service interface layer creates a new abstraction level between the business process layer and the application layer. [31, pp. 281–282]

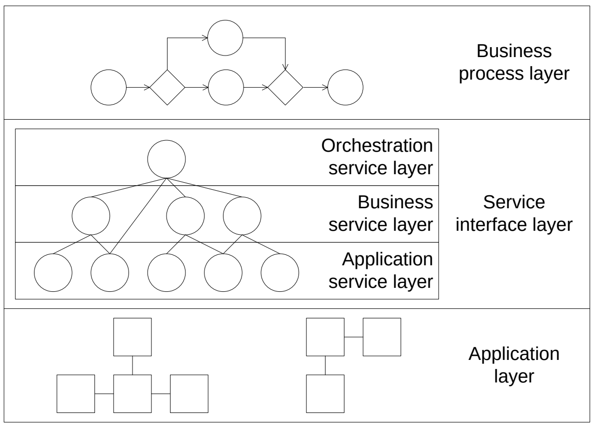


Figure 3.6: Services can be categorized to abstraction layers on the basis of the degree of abstraction. [31, p. 337]

interface layer which is located between the business process layer and the application layer. A service can contain application logic, but it can also be composed of other services, which means that it can contain business logic as well. That is, service interface layer has a dual role as shown in figure 3.5. [31, pp. 280–282]

To follow the principle of loose coupling, it is necessary to divide the service interface layer to three abstraction layers, namely *application service layer*, *business service layer* and *orchestration service layer* (figure 3.6). Services are categorized according to degree of abstraction: services representing application logic belong to the application service layer and the ones representing business logic belong to the business service layer respectively. The topmost service layer, orchestration service layer, is for services that hold the logic required to describe an entire business process. [31, pp. 334–336]

Services can not only be situated in different abstraction layers but they can also be categorized to service models according to their purpose (table 3.2). It is important to note that the categories do not exclude each other; a service can belong

Table 3.2: The service models that are relevant to this study.

Service model	Explanation
Utility service	Generic and reusable by design. Typically situated in the application service layer, but even the reusable services of higher layers can be utility services. [31, p. 127, 341, 344]
Wrapper service	Enables the integration of an incompatible component by "wrapping" its interface. Typically located in the application service layer. [31, p. 339]
Hybrid service	A service with both application and business logic. If service abstraction layers are used, hybrid services belong to the application service layer. [31, p. 719]
Task-centric business service	Contains the business logic of one task or business process. Reuse potential is low. [31, p. 342]
Entity-centric business service	Contains one business entity or a resource. Can be designed reusable. [31, p. 342]
Process service	Represents an entire business process. Located in the orchestration service layer. [31, p. 719]

to more than one service model. There are even more service models, but only the ones that are relevant in the scope of this study are explained. [31]

3.3 Implementing Business Processes

Service-oriented architecture and business processes are terms on a high abstraction level. Implementing the layers of a service-oriented business process shown in figure 3.5 is a complex task, and there are several alternative technologies that could be used for such an implementation. At the service level, this study concentrates on Web Services and the technologies related to it. Web Services were chosen as they on one hand have a high market acceptance and on the other hand take the abstraction level high enough to provide interoperability. For the same reasons, WS-BPEL and BPMN were chosen to be used at the highest abstraction levels. WS-BPEL is an XML-based language that describes the execution of a business process whereas BPMN is a graphical notation for the same purpose. Each member of the chosen technology set has its position either in the business process layer or in the service interface layer (figure 3.7). It is worth noting that, in the chosen technology set, there is only one technology in the business process layer, namely BPMN, whereas service interface layer contains five different technologies[1]. It suggests that the biggest challenge is not to describe a business process but to make services interact.

BPMN is not the only notation to describe business processes. Several software vendors provide business suites that use a proprietary notation to describe business processes. For example, SAP NetWeaver BPM uses a notation that has some

[1]As WS-BPEL describes the execution of business processes, it has also some characteristics of the business process layer.

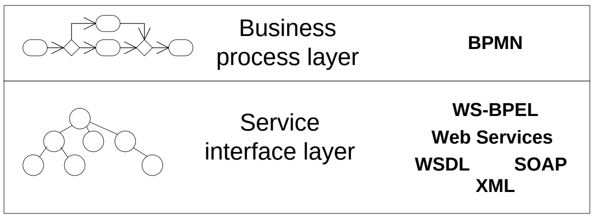


Figure 3.7: Together with Web Services, BPMN and WS-BPEL are an alternative to create service-oriented business processes.

common characteristics with BPMN [85]. Due to its several differences, NetWeaver BPM cannot be called a pure BPMN modeler. IBM WebSphere has a notation that is based on BPMN 1.1, but it uses proprietary graphical icons [95]. Microsoft has its own BPM software, too. Business processes are modeled with Microsoft Visio using a notation called ODBA (Business Process Orchestration Diagram) and then exported to Microsoft BizTalk Server. [97]

The next subsections introduce BPMN and WS-BPEL. Then, an overview of current business process software is given.

3.4 BPMN

Business Process Modeling Notation (BPMN) is a notation to visualize business processes. It has two major design goals. First, BPMN shall be understood by "all business users", including business analysts, technical developers and business managers, and second, it shall be capable of visualizing XML-based business process execution languages. [98, p. 1] The current version of BPMN is 1.2, and also two beta versions of 2.0 have been released. This document deals with version 1.2 as version 2.0 is still informal.[2] [20]

BPMN specifications provide a mapping for a business process execution language. The specification of version 1.2 describes a mapping to BPEL4WS[3], but it is non-normative and insufficient [98, p. 1]. As the syntactical changes of WS-BPEL are few compared to BPEL4WS [7], it is obvious that the mapping applies at least partially also to WS-BPEL. The mapping has been developed further for BPMN version 2.0 (now, it has been made for WS-BPEL instead of BPEL4WS), but it is still stated that not all BPMN-to-WS-BPEL mapping can be defined straightforwardly [2, p. 461]. That is, the specifications offer only a guideline for business process software vendors, likely resulting in vendor-specific features in BPMN-to-WS-BPEL software implementations.

[2]The changes made for version 2.0 this far include notational changes such as new diagram elements and two new diagram types. Moreover, BPMN 2.0 introduces technical changes such as exchange formats and BPMN Diagram Interchange (BPMN DI) to facilitate the exporting and importing of data between applications. [2, p. 377, 491, 495]

[3]BPEL4WS and WS-BPEL are explained in the next section.

As the design goals of BPMN to be both intuitive and powerful to accomplish complex tasks are potentially conflicting, there are two groups of graphical elements in BPMN. The *extended element set* provides the full functionality of BPMN whereas its subset, *core element set*, visualizes simple processes. In addition, the functionality is extended by non-graphical attributes that are attached to the graphical elements. [98, p. 17] The core element set is shown in figure 3.8, and the meanings of the elements are explained in the next paragraphs.

Flow objects describe the flow of a business process. An *event* is used when something "happens" in the business process. An event can have either a trigger that defines what causes the event or a result that is thrown by the event when the process flow reaches it. An *activity* describes something that is performed by the organization in which the process is executed. An activity can be either atomic or compound depending on whether it contains more detailed functionality in it. *Gateways* are used to control a process flow by branching, forking, merging or joining different execution paths. [98]

Connecting objects are used to connect objects to each other. A *sequence flow* connects activities indicating the order of their execution. A *message flow* indicates when process participants send messages to each other, and an *association* attaches information to process elements. [98]

Swimlanes are used to divide elements to a group. A *pool* contains the activities of one process participant, and a *lane* is a part of a pool that is used to categorize activities. [98]

Artifacts provide process-related information that does not affect process flow directly. *Groups* are used to categorize activities, and *data objects* describe how data is used in the flow. *Text annotation* is used to give additional information, and it can be connected to an element in a diagram. [98]

The specification of BPMN does not define the notation strictly as it allows, for example, additional graphical elements, different colors and different line styles as long as they do not conflict with any element in the specification. Moreover, some features are declared optional regarding the way they are displayed or whether they shall be supported. However, it is meaningful to preserve the BPMN look-and-feel to maintain the understandability of the notation. [98]

The simple fictional business process in figure 3.9 demonstrates some of the elements of BPMN. In the process, a person applies for study grants. The process has two participants, "Applicant" and "Study Grants Board". First, the applicant fills in an application formula after which the formula is sent to the Study Grants Board. After receiving the application, the Study Grants Board decides whether the application shall be approved or not. If the application is approved, study grants are granted for the applicant. Finally, the Study Grants Board sends its response to

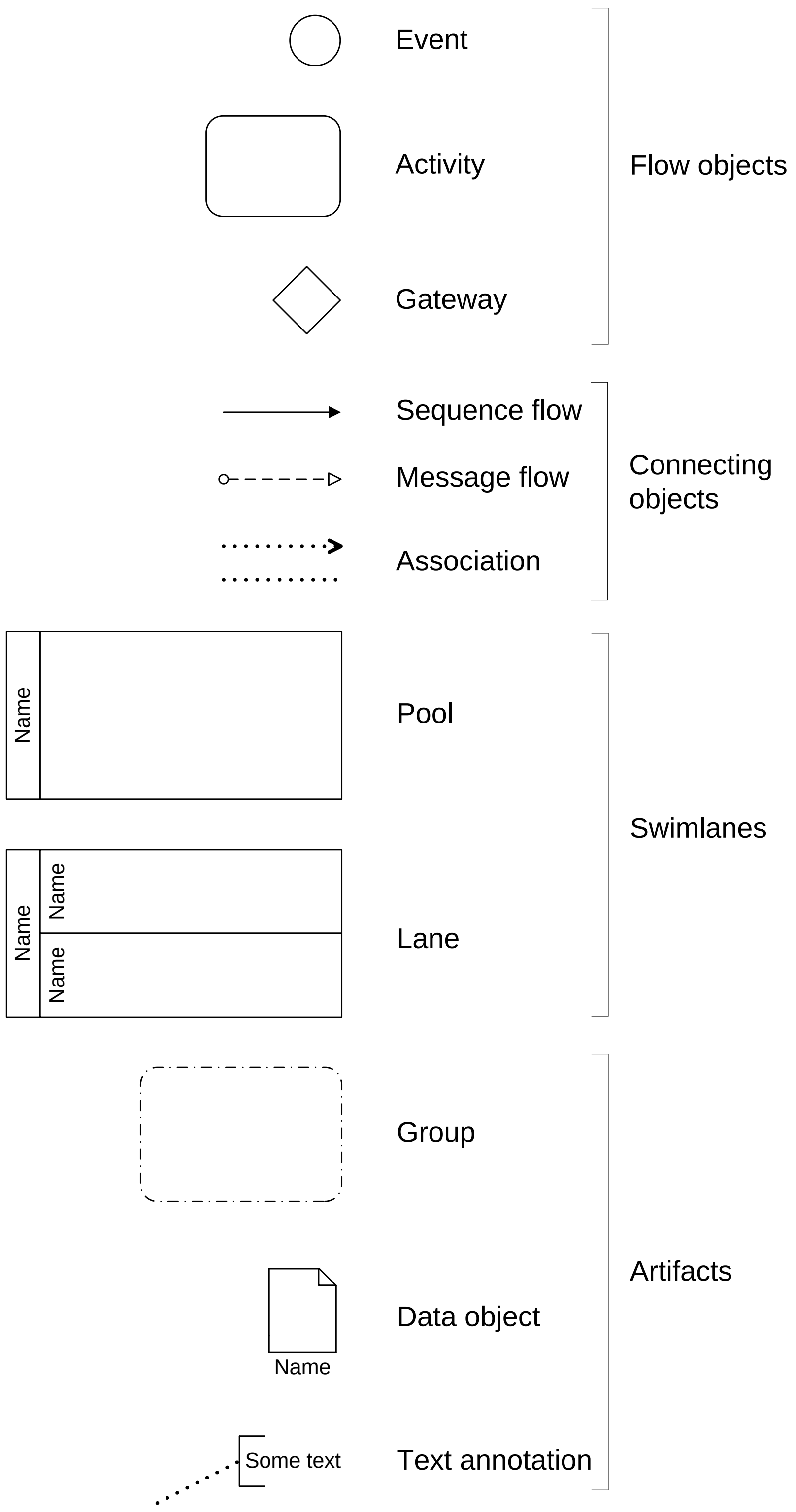


Figure 3.8: The core element set as defined in BPMN 1.2. [98, pp. 17–20]

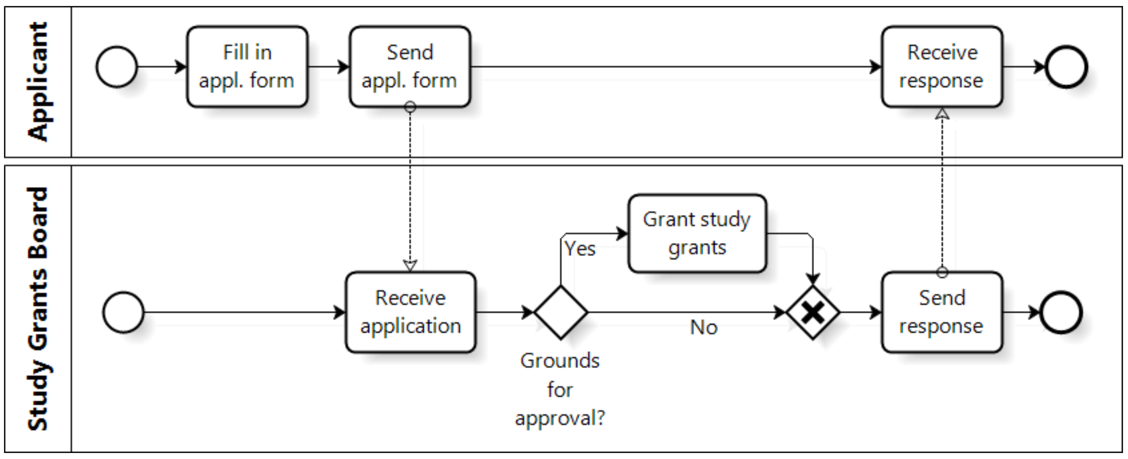


Figure 3.9: A business process of applying for study grants modeled with BPMN.

the applicant. According to White et al., the features of BPMN are far beyond the example: for instance, timed waiting, assignment of variables and looping execution can be defined [98].

To conclude, BPMN is a notation whose purpose is to bring the design of business processes and their implementation closer to each other. BPMN can be mapped to business process execution languages, and it provides both an intuitive notation and features to describe complex functionality.

3.5 WS-BPEL

WS-BPEL (Web Services Business Process Execution Language) is an XML language that orchestrates service-oriented business processes. The initial version of WS-BPEL was first called BPEL4WS (Business Process Execution Language for Web Services). Its version 1.0 was released in 2002 and it was influenced by IBM's Web Service Flow Language (WSFL) and Microsoft's XLANG. Version 1.1 was released in 2003, and the current version, WS-BPEL 2.0, was released in 2007. [7] WS-BPEL uses several other XML specifications such as WSDL 1.1, XML Schema 1.0, XPath 1.0 and XSLT 1.0. WSDL is not only used to call the services being orchestrated. WS-BPEL business processes themselves have a WSDL description which makes it possible to use them like any Web Service and to compose them into larger entities. [3]

WS-BPEL can describe both *abstract* and *executable* business processes. Abstract processes are specified only partially and their purpose is not to be executed but to describe a business process. [3] This document concentrates only on executable processes. Code 3.1 shows a simplified WS-BPEL document to give an overview of a typical document structure, and table 3.3 discusses the purposes of the elements in the document.

```
<process>
  <extensions></extensions>
  <import></import>
  <partnerLinks></partnerLinks>
  <messageExchanges></messageExchanges>
  <variables></variables>
  <correlationSets></correlationSets>
  <faultHandlers></faultHandlers>
  <eventHandlers></eventHandlers>
  <!-- Activities here -->
</process>
```

Code 3.1: The basic elements of a WS-BPEL document. [3]

Table 3.3: The possible child elements of the ”process” element in a WS-BPEL document. Activities (the elements that define business process flow) are not included. [3]

Element (occurrences)	**Purpose**
extensions (0..1)	Contains extensions to WS-BPEL language.
import (0..*)	Imports an external document, such as an XML Schema or a WSDL document.
partnerLinks (0..1)	Contains *partner links* that model the services that the business process uses.
messageExchanges (0..1)	Contains definitions that point out to which reply activity an *inbound message activity* (IMA) is bound if there are multiple IMA-reply pairs.
variables (0..1)	Contains variable definitions. A variable can be one of the following types: a WSDL message type, an XML Schema type or an element.
correlationSets (0..1)	Contains declarations how to target a message to the right business process instance.
faultHandlers (0..1)	Contains fault handling mechanism definitions.
eventHandlers (0..1)	Contains event handling mechanism definitions.

The actual flow of a business process is described by *activities*. Activities are, for example, control structures, variable handlers or message handlers. There is a total of 21 activity elements: for instance, ”receive”, ”reply”, ”assign”, ”if”, ”while” and ”scope”. The comment ”Activities here” in code 3.1 indicates where activities are located in a WS-BPEL document. If there are any fault handlers defined, they can also contain activities as they define what to do in case of a fault. ”Scope” is an important activity: it defines a behavioral context, restricting the visibility of the elements inside it. All of the elements in table 3.3 except ”extensions” and ”import” can also situate in any ”scope” activity. [3]

The activities of WS-BPEL are divided into two groups: basic activities and structural activities. Basic activities define the basic steps of a business process whereas structural activities control the flow of a business process. *Basic activities* are, for instance, ”wait”, ”throw”, ”assign”, ”receive” and ”reply”. ”Wait” activity provides a mechanism to wait for a specified time or until a deadline has been reached. ”Throw” is used to indicate that a fault has occurred. ”Assign” is used, for example, to assign data from an existing variable to another or to a new variable. ”Receive” and ”reply” activities are used to define interaction with services. *Structural activities* can do three patterns: *sequential control*, *concurrency and synchronization* or *deferred choice*. Sequential control defines that activities are performed one after another. The activities that implement it are ”sequence”, ”if”, ”while”, ”repeatUntil” and the serial variant of ”foreach”. Concurrency and synchronization are defined with ”flow” or the parallel variant of ”foreach”. Deferred choice is made with ”pick” activity. It means that a group of events with associated activities is specified, and whichever of the events comes first will have its activity is launched. [3]

Code 3.2 demonstrates ”if” activity. Its logic is the same as the if-elseif-else structures of programming languages like C++ or Java. The conditions are evaluated one after another, and the first one that matches will be executed. If no condition matches, ”else” branch will be executed.

In conclusion, WS-BPEL is a relatively complex XML language whose purpose is to describe the execution of service-oriented business processes. It provides a rich functionality, such as concurrent execution, event handling and nested scopes.

3.6 Support Software Overview

There are different business process modeling software products available such as ActiveVOS [1], Fujitsu Interstage [38] and BizAgi [12]. Some software products provide both BPMN modeling and business process execution while others have only the capability to represent BPMN. Here, Intalio BPMS is used to demonstrate an executable business process. Another modeling application called SOA Tools BPMN Modeler for Eclipse is also discussed.

```
<if>
  <condition>$temperature > 10</condition>
  <!-- Do activities 1 -->
  <elseif>
    <condition>$temperature > 5</condition>
    <!-- Do activities 2 -->
  </elseif>
  <else>
    <!-- Do activities 3 -->
  </else>
</if>
```

Code 3.2: The basic structure of an "if" activity. [3]

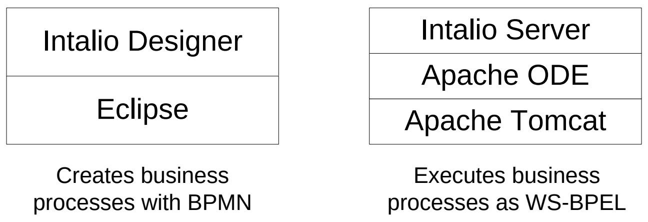

Intalio Designer
Eclipse
Creates business processes with BPMN
Intalio Server
Apache ODE
Apache Tomcat
Executes business processes as WS-BPEL

Figure 3.10: The two Intalio applications required to create and execute business processes. [48]

Intalio BPMS is a business process management suite that consists of two parts: Intalio Designer and Intalio Server (figure 3.10). Intalio Designer is based on Eclipse integrated development environment. It provides the tools to model business processes with BPMN and to convert them into BPEL. Processes are executed on Intalio Server which uses Apache ODE as the BPEL engine. The BPEL version used by Intalio BPMS is WS-BPEL 2.0. [48]

Intalio BPMS is available as two different editions: community edition and enterprise edition. The download and use of the community edition are completely free-of-charge, but the only technical support offered is the community forum on Intalio website. In contrast, the enterprise edition has to be paid for, but its customers receive technical support on the basis of a service level. [48]

Figure 3.11 demonstrates a simple fictional service-oriented business process created with Intalio BPMS. In the process, an employee requests a holiday, and a manager either accepts or declines the request. If the request is accepted, the holiday period is saved into the holiday database of the company. Finally, the employee receives a message that shows whether the holiday was accepted or not.

To understand the interaction between the parties of the holiday request business process in figure 3.11, it is essential to understand the meaning of its four pools. Both human roles, employee and manager, have a pool. "Process" pool describes the flow of the process, and the downmost pool represents the holiday database. The

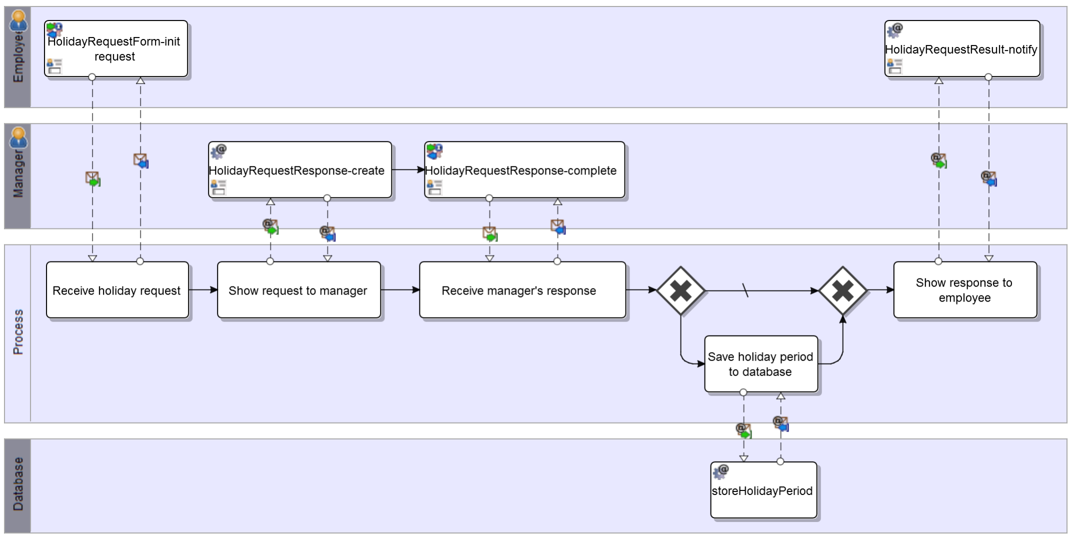

Employee
Manager
Process
Database
HolidayRequestForm-init request
HolidayRequestResult-notify
HolidayRequestResponse-create
HolidayRequestResponse-complete
Receive holiday request
Show request to manager
Receive manager's response
Show response to employee
Save holiday period to database
storeHolidayPeriod

Figure 3.11: A fictional holiday request business process created with Intalio BPMS.

boxes in "Employee" and "Manager" pools represent forms that are used in a WWW browser. They are requested from the server where the business process is being run. The forms have been created with Intalio Designer. The holiday database has a WSDL-defined interface, and the box in "Database" pool represents one operation call to the interface.

Creating a business process in Intalio Designer is relatively intuitive and straightforward though some business process functionality requires more expertise. The activities to be used are dragged and dropped into the business process, and a sequence of activities is created by simply connecting them using a mouse. To define which Web Services are called in a business process, any WSDL files and XML schemata are imported. Then, the service operations being used are dragged and dropped into the lane on which they are called. Variables-related tasks, such as assignments and variables-based conditions, are defined in the graphical "Mapper" view. However, not all business process functionality can be defined graphically. That is why Intalio Designer also allows WS-BPEL input in some activities.

Like Intalio BPMS, SOA Tools BPMN Modeler[4] is run on Eclipse, but it is downloaded as an Eclipse plugin, not as a distribution. It does not support the export of business processes to WS-BPEL, to BPEL4WS or to any other business process execution language. It is possible to install an Eclipse plugin to perform the export [86], but as far as is known, such a plugin is not available. STBM has been developed by Intalio [86]. Thus, it is not a surprise that the graphics of STBM and Intalio Designer have similarities. As STBM has only the functionality to create BPMN diagrams, it has fewer features. However, for creating a BPMN diagram with no need to create an executable process, STBM in even better than Intalio

[4]To reduce repetition, SOA Tools BPMN Modeler is referred to as "STBM".

BPMS. As it does not try to convert diagrams to another language, STBM allows a more flexible BPMN usage.

In summary, several vendors provide their own BPM software. Intalio BPMS can be downloaded free of charge, and it has the functionality to describe and execute relatively complex business processes. SOA Tools BPMN Modeler for Eclipse offers a lighter alternative if the main interest is only to draw diagrams in BPMN.

3.7 Conclusions

Composition is an important aspect of SOA: by composing services so that they become a part of another service, more complex tasks can be accomplished. Different services with a different degree of composition can be categorized to different service layers according to their complexity. This way, composition makes SOA a layered architecture. If a composed service has a certain business result, it is called a service-oriented business process; then, the composition process is called orchestration. There can also be interaction between two or more business processes of different organizations; in that case, the result is a service-oriented business collaboration and the creation of such a collaboration is called choreography.

Technologies have been developed to facilitate orchestration. BPMN is a graphical notation with which the flow of a business process can be described so that both business and software professionals understand it. BPMN can visualize WS-BPEL which is an executable XML-based language to describe business processes. WS-BPEL enables the use of several Web Services and XML related technologies such as WSDL, XML Schema, XPath and XSLT.

Several software products are available for business process modeling. Some of them use BPMN or another graphical notation. Some products can even convert diagrams to an executable format which can be WS-BPEL or some proprietary implementation. Intalio BPMS is an open source business process modeling suite which uses both BPMN and WS-BPEL.

4. TOWARDS INDUSTRIAL SERVICE-ORIENTED ARCHITECTURE

SOA is said to have several benefits in distributed systems. Industrial automation systems are also typically distributed which suggests that SOA would be a good choice for them as well. However, the requirements of automation systems are partially different to the requirements of typical business systems, and the current service-oriented technologies cannot meet all of them. This chapter discusses the work made this far to overcome these challenges.

4.1 Motivation

The traditional industrial automation technology suffers from inflexibility. According to Bangemann et al., industrial automation systems have traditionally been hierarchical, and several different technologies have been used at different levels (figure 4.1). The diversity leads to different data formats and incompatibility between the levels, making their integration problematic. [5, p. 2] Moreover, current systems are expensive to install and to change or to expand. Component vendors favor their proprietary technologies. As a result, when a new component is installed or an obsolete one is replaced, high costs result especially if the new component has another vendor than the existing ones. However, flexibility is of high importance as the market situation keeps changing rapidly and new technologies are developed all the time. [51, p. 62]

The adoption of SOA in industrial automation suggests several benefits. It would break the strict system hierarchy and facilitate the communication even between the systems performing strategic control (such as ERP) and field devices [36, p. 5]. In addition, as the implementation and the interface of a service are separated from each other, the interoperation of components manufactured by different vendors would not be a problem anymore [55]. Besides, replacing a component with another component with the same interface would require minimal configuration. Systems would be more flexible and more adaptable, and there would be more potential for component reuse. As new hardware and communication techniques are developed, it is possible to make components more intelligent, enabling the distribution of decision-making into field devices. This decentralization would make systems more

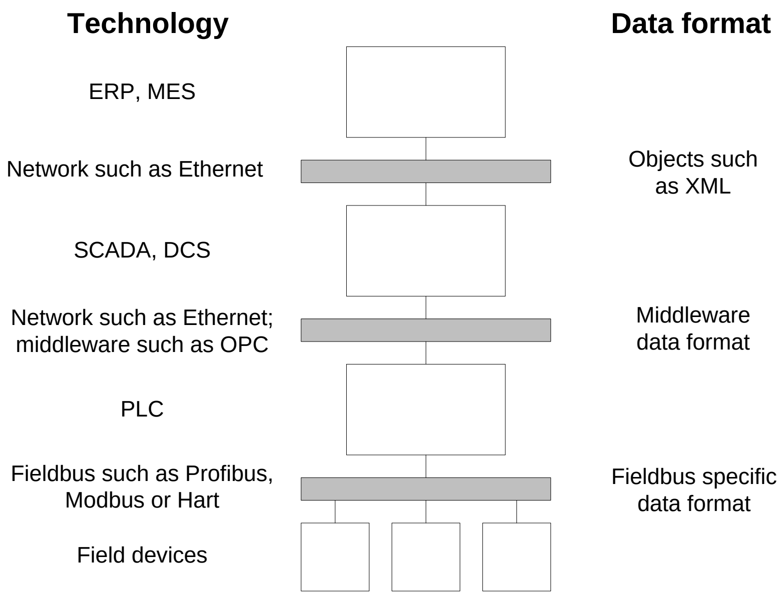


Figure 4.1: The typical hierarchy of a traditional industrial automation system. Different communication technologies are used on different levels. Combined from [5, p. 2] and [40].

robust as there would no longer be a single point of failure. Intelligent devices could also communicate directly in peer-to-peer manner. [51]

4.2 Research and Implementations

Although SOA has some apparent benefits in industrial systems, its implementation is currently problematic. According to Komoda, the highest level of industrial automation systems ("factory management system") is not a problem: it is comparable to any typical business information system, and suitable service-oriented technologies exist already. However, the industrial devices in the productive level are challenging from the SOA point of view as their operation may have real-time requirements. [54, pp. 2–3] Moreover, their computational resources are low [36, p. 1]. Thus, the most important question is how to make devices with different capabilities and different requirements capable of service-oriented communication with each other.

The next subsections sum up the most recent research work that has brought SOA and industrial systems closer. First, OPC UA and DPWS are introduced. Then, some real-time SOA frameworks that are relevant for this domain are looked at. They are RTSOA, RT-Llama, RI-MACS and a middleware that uses JMS API.

4.2.1 OPC UA

OPC UA is a specification to facilitate the integration of industrial software products made by different vendors. For that, OPC UA provides an information model, a message model, a communication model and a conformance model. OPC UA is intended to be platform-independent so that it can operate on different devices with different capabilities. Thus, different application areas such as field devices, MES and ERP are possible. Before OPC UA, several OPC specifications were released for different purposes: to retrieve current data, to retrieve historical data and to receive notifications about events. OPC UA provides the functionality of all of them in one technology. [72]

The OPC UA specification consists of 13 parts. They have been released separately, some in 2009 and some in 2010. The current version is 1.01 for some parts and 1.00 for the others whereas parts 12 and 13 have not been released yet. [71] All the parts of the OPC UA specification except part 6 (Mappings) are written on a high abstraction level. As the mappings to concrete communication technologies are defined in one part, it is the only one to modify if new technologies are adopted in the future. [73, p. 5]

To perform communication, there are two data encoding methods and two transport protocols. The *data encoding methods* are called *OPC UA Binary* and *OPC UA XML*. The purpose of the binary encoding is to enable fast encoding and decoding as well as to have only a small overhead. It does not provide a way to include field or type names inside data as any utilizer of those messages is supposed to know their structure beforehand. In contrast, as the name of OPC UA XML suggests, it uses XML to represent data. One of the *transport protocols* uses SOAP-over-HTTP, and the other has been built on TCP. The SOAP-over-HTTP protocol uses WSDL to describe services, and XML Schema is used to define the structure of XML documents. [73] The designer of an OPC UA application can provide a support for both the encoding methods and both the transport protocols. This way, the end user decides which is more important: to have an easy integration with XML and Web Services or to get a better performance with other alternatives. [72, p. 9]

The performance of the OPC UA binary encoding and the TCP transport protocol has been experimented by Salmenperä & Salonen. In their study, an OPC UA communication stack implemented in Java was tested with different data packet sizes and different levels of security. In the experiment, if no security features were used and the packet size was 1024 bytes, the response time never exceeded 2 milliseconds. With encryption, the response times with the same packet size were typically under 3 milliseconds, exceeding it sometimes. When the packet size was 10240 bytes, the response time with no encryption remained under 10 ms whereas the delivery of encrypted messages took between 8 and 60 milliseconds. [82]

The question is whether OPC UA is suitable for real-time functions. 10240 bytes is enough to carry complex data, and 1024 bytes or even less is enough for simple messages. A typical response time under 10 milliseconds with these packet sizes is

Table 4.1: The areas on which DPWS concentrates and the specifications that are used. [68]

Area	**Technologies**
Messaging	SOAP, HTTP, UDP, WS-Addressing, RFC 4122, MTOM
Discovery	WS-Discovery
Description	XML Schema, WSDL, WS-MetadataExchange, WS-Policy, WS-PolicyAttachment, WS-Transfer
Eventing	WS-Eventing
Security	AES/TLS, HTTP Authentication, SHA, TLS, RFC 4122, X.509.v3, WS-Security

promising for soft real-time functions. However, the study does not address how loaded the network was during the experiment. If there had been more traffic in the network, the response times could have been longer and more indeterministic. Still, it can be said that at least the stack used in the experiment is an alternative for soft real-time systems if the OPC UA binary encoding is used over TCP. For hard deadlines, the results suggest too much unpredictability.

4.2.2 DPWS

DPWS is concerned with bringing Web Services support to devices. Devices with a Web Service interface can be resource-constrained whereas Web Service clients can offer a higher flexibility. To enable communication between different parties, DPWS gathers a set of existing technologies and specifies a way to use them. [68] That is, rather than creating a new technology, DPWS concentrates on specifying common usage rules. The current version of DPWS concentrates on five different areas: *messaging*, *discovery*, *description*, *eventing* and *security* [68]. Table 4.1 shows a list of the specifications that are used in these areas. Several specifications on the list are outside the scope of this document, but they are listed to indicate the high number of the technologies lying under DPWS.

The initial step in the adoption of DPWS was the SIRENA project (Service Infrastructure for Real-Time Embedded Networked Devices). The goal of SIRENA was to integrate embedded devices in industry, telecommunications, automotive and home automation by creating a service-oriented framework. DPWS was not the only technology whose usage was considered, but it was chosen due to the limitations of other alternatives. SIRENA was begun in 2003 and finished in 2005. [14]

Another project, SODA (Service Oriented Device & Delivery Architecture), followed from the results of SIRENA [88]. The goal of SODA was to develop a ”device-level service-oriented ecosystem”. Subgoals were, for instance, creating a complete tool set for service-oriented architectures of devices, integrating services provided by devices with business processes and improving both the performance and the security of SOA on devices. [83] SODA was begun in 2006 and finished in 2008 [88].

From the industrial automation point of view, the SOCRADES project (Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded Systems) was a fundamental step. SOCRADES is a sequel of SIRENA and SODA [88]. Its goal was to enable a Web Service based service-oriented architecture in both wired and wireless automation devices with requirements such as robustness, autonomy, intelligence and reusability. DPWS was used in the project. [5, p. 3] SOCRADES was begun in 2006 and finished in 2009 [87].

Despite the results of SIRENA, SODA and SOCRADES, DPWS is not an "all-round" solution for industrial automation. Both the resource consumption of DPWS and its lack of real-time support are problematic in some systems. According to Kannisto, several studies have been made to overcome these problems. DPWS gateway devices may be used to wrap devices that are not DPWS capable. Gateways may be implemented for each device alone, but they can also wrap a group of devices or an entire network. It is also possible to provide aggregated data from the wrapped devices. There have also been studies to create a real-time capable DPWS system. One possibility is to make the communication protocols under DPWS real-time capable. Another solution is to have two network connections in a device: one for a real-time capable protocol and another for DPWS. Finally, the lack of deterministic communication may not be the only problem. It has been stated that response times may be long if devices cannot provide powerful computation. [52]

4.2.3 RTSOA

RTSOA is a study to create a real-time SOA framework. This far, no complete framework has been implemented. The requirements of such a framework have been considered on a theoretical level, and different service composition algorithms have been simulated. [91]

In RTSOA, real-time requirements are considered during the phases of the entire life cycle of an application. In the *modeling phase* of an application, the real-time constraints of individual services and the workflow of the application are considered. In the *assembly phase*, the functionality of the application is analyzed and evaluated, and even the phase itself has a response time constraint. After assembly comes the *deployment phase* in which the application is deployed so that it can be run. Any resources required by the nodes in the network are reserved. The final phase is called *management phase* in which the application is executed and managed. If the real-time functions of the application do not meet their deadlines, the management phase is followed by another modeling phase for which it provides feedback. That

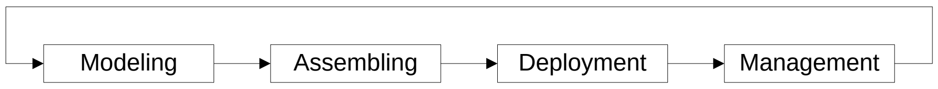


Figure 4.2: The life cycle of an SOA. [46, pp. 20–23]

is, the life cycle can be modeled as a loop of repeating phases as illustrated in figure 4.2. [91, pp. 1–2]

Several real-time factors must be considered in RTSOA. When a service is being modeled, the following things must be addressed: minimal and maximal response times, service invocations capacity, the maximum number of consumers and the required resources. As a service can be composed dynamically at runtime, available services need to be known by the system beforehand. Moreover, service information has to be accessible in real time. As service deployment is performed in real-time, it needs network bandwidth reserved for it. When an application is being run, it is analyzed in real-time to observe whether it meets its real-time requirements. The execution environment of RTSOA has to provide real-time support, enabling different levels of quality of service (QoS). [91, pp. 2–5]

In the study, simulations were performed for dynamic service composition algorithms. Such an algorithm needs to optimize two targets: the end-to-end execution time of the application being composed and the total resources consumption caused by the algorithm itself. Two different algorithms were analyzed: an exhaustive algorithm and a heuristic algorithm. The exhaustive algorithm finds always the optimal solution, but its resources consumption grows rapidly as the function of the number of the services being composed. In contrast, the heuristic algorithm can only find a suboptimal solution but it consumes less computational capacity. It was stated that, on the basis of resources consumption, the heuristic algorithm is more suitable for an RTSOA implementation. [91, p. 6–8]

4.2.4 RT-Llama

Panahi et al. introduced another real-time SOA framework, namely RT-Llama. In RT-Llama, the resources required for business processes are reserved in advance, and the CPU bandwidth of hosts is controlled. As a result, it is possible to schedule all the parts of a business process. As service availability may vary, a new business process is generated for each run even if a user wants to run a business process with the same requirements. The resource reservations themselves do not have real-time support. [74, pp. 460–461]

The environment required by a real-time SOA has several requirements. Both the *operating system* and the *communications infrastructure* lying under the middleware must be real-time capable, and the communications must also have QoS

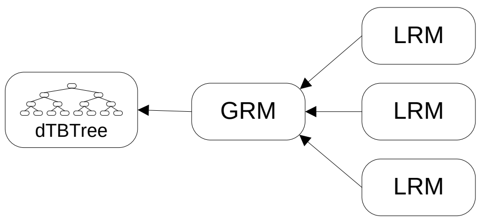


Figure 4.3: The resources management of RT-Llama is partially centralized and partially distributed. [75]

functionality. Moreover, the *business process composition infrastructure* must take user's scheduling requirements into account, and the *distribution middleware* must support the reservation of resources in advance. In addition, any unpredictability caused by *client infrastructure* must be known. However, the current work concentrates only on two of them: business process composition infrastructure and distribution middleware. Real-time scheduling is provided by real-time Java and Solaris 10 operating system. [74, p. 461]

Even though RT-Llama is real-time oriented, not all the services are required to be real-time capable. A service can be deployed to be either *reserved* or *unreserved* depending on if it can only be requested after a reservation or not. [74, p. 462]

To reserve resources required by a business process, two different strategies can be used: *concurrent* or *sequential*. In the concurrent strategy, the resources are reserved before the business process begins, and the start time and the end time of each service call are selected in advance. In contrast, the sequential strategy reserves each service one after another. It is greedy, attempting to maximize the remaining time to execute a business process. Both the strategies have advantages. The concurrent strategy is typically faster. However, sequential strategy has a higher success rate as indicated by simulations made in the study. [74, p. 464–466]

Panahi et al. have also released another paper related to the RT-Llama framework. It concentrates on the management of resource reservations that have been implemented using a binary tree data structure ("dTBTree"). dTBTree is located in a node called *Global Resource Manager* (GRM) and its information is updated periodically by *Local Resource Managers* (LRM) that each host maintains (figure 4.3). Another advance compared to the previous RT-Llama is the implementation of a "pre-screening" mechanism. Its purpose is to avoid choosing services whose utilization rate is high as it would raise the probability to miss deadlines. [75]

The study also presents simulation results that demonstrate how the success ratio, the efficiency and the effectiveness of the reservation system changes when different parameters are varied. The parameters include the utilization threshold used in pre-screening, system workload, dTBTree size and the update frequency of dTBTree. A high pre-screening threshold value results in a higher success ratio, but it also lowers reservation efficiency. Moreover, a higher workload causes more failures. The results also indicate that there is no significant difference in success rate or efficiency when the dTBTree is updated less frequently. Finally, it is shown that when pre-screening threshold is high and dTBTree size is small, the effectiveness of the system is low. [75]

4.2.5 RI-MACS

Cucinotta et al. have released a study that concentrates on a real-time SOA targeted especially for industrial automation. Their point of view is technical, suggesting not only common real-time system requirements but also evaluating real-life technologies. According to the paper, there are two key concerns in a distributed real-time system: communications based on QoS agreements and ”temporal isolation” of tasks on processors (that is, real-time aware scheduling).

The study represents an architecture called RI-MACS. It has two application programming interfaces (API): a *common API* and a *custom API.* The common API uses DPWS based communication, and as DPWS has no real-time support, there is a separate real-time communication channel on the lower protocol levels. The custom API is used to integrate any legacy devices, and it can meet hard real-time requirements. RI-MACS uses WS-Agreement [4] to negotiate QoS levels. When a client wants to use a service, it makes a QoS offer that is submitted to the provider of the service. Then, the offer is either accepted or rejected. [25]

To enable CPU reservations, RI-MACS uses a real-time capable Linux operating system which runs a web server with a real-time module. Experiments of DPWS calls were made on the server both with and without the real-time module in use. ”Heavy” load was generated on the server after which two different image processing services with maximal response times of 300 milliseconds and nine seconds were called. The results indicate that when the real-time module was in use, deadlines were not missed in either of the experiment. In contrast, deadlines were missed frequently when the real-time module was not in use. [25]

4.2.6 Middleware With Partial JMS API

Garces-Erice has suggested a middleware that implements the JMS (Java Messaging Service) API partially. JMS was chosen as its API is ”mature” and ”well known to programmers in the enterprise environment”. The middleware is intended to enable SOA, and there are three main requirements: the middleware shall be real-time capable, it shall be lightweight to save resources and it shall be compatible with legacy infrastructures. [41]

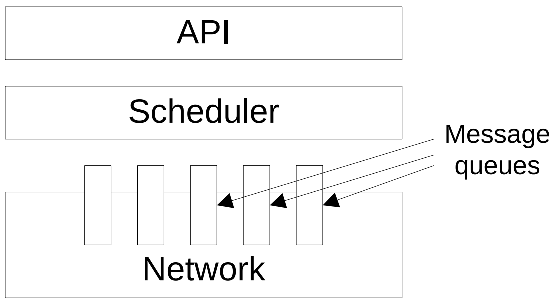


Figure 4.4: The high-level structure of the real-time capable SOA suggested by Garces-Erice. [46, p. 81]

Figure 4.4 shows the architecture of the system. The downmost layer is *network* which implements communications. It is not specified which communication protocol shall be used; thus, any legacy protocol can be preserved. The layer above the network layer is *scheduler* which implements the actual real-time functionality. It puts events into different queues according to their type. An individual queue is treated using the *FIFO* (first in first out) principle. The architecture does not require any specific scheduling algorithm to choose from which queue the next task is taken in a specific situation. The topmost layer is the *API* that is implemented using JMS. [41, pp. 80–82]

The features of JMS are rich, and not all of them are used in the architecture. JMS provides both synchronous and asynchronous messaging modes of which asynchronous is favored in the study as it enables event-based communication. JMS communication is reliable, and three different modes can be used to send message acknowledgements. Not all the features are suitable for a real-time environment, and that is why some of its functionality is not implemented in the study. [41, pp. 80–82]

To experiment the performance of the architecture, simulations were made. They showed that both latency and latency variation were low which suggests suitability for a real-time environment. [41, pp. 83–84]

4.3 Conclusions

If all the devices of an industrial automation system used a service-oriented way to communicate, it would facilitate integration and reduce the need for configuration while installing or modifying a system. However, there are two obstacles: on one hand, the computational resources of devices are restricted, and on the other hand, there is no real-time capable SOA technology. This chapter discusses the efforts made this far to overcome these problems, putting more weight on real-time issues.

Despite all the work made for a real-time SOA, no solution exists this far. The binary encoding of OPC UA is an alternative for soft real-time functions, but its indeterministic nature prevents its use in hard real-time systems. DPWS made it easier to use Web Services in devices by lowering the required amount of computational resources, but the specification does not consider real-time requirements at all. Even though there have been successful attempts to shorten the response times of DPWS, as long as the current specification is used, the only possibility to reach deterministism is to build a real-time support separately. Cucinotta et al. used a separate real-time module in a server application. Even though the results of the study are promising, the real-time module is not a complete solution for an industrial SOA. It guarantees only the behavior of the server in which it is run – the behavior of the communications channel is not controlled by it. Some promising simulations have been made with the architecture of Garces-Erice, but the technology is dependent on Java, and it is not ready to be taken into use currently. The study on Tsai et al. concentrate on dynamic service composition whereas Panahi et al. have made research on resources reservation and the scheduling of an entire business process. Again, these aspects are essential in a real-time SOA, but a complete architecture cannot be implemented with them alone.

Service-oriented architecture is still making its way to industrial automation. There has been a lot of development work on different service-oriented frameworks, and currently, there are obstacles that make it impossible to implement an industrial SOA that covers all the communications of an industrial plant. However, as development work goes further, there seems to be nothing to prevent service-orientation from being the future architectural style of industrial automation. Still, it is going to take several years before SOA is a real candidate for a complete production system.

5. SERVICE-ORIENTED LIFE CYCLE SIMULATION

In industrial plants, there are several functions for which SOA can be applied. The estimation of environmental impacts is one example. When the equipment of an industrial plant is being chosen, an important aspect is whether some devices load the environment less than the others. This chapter introduces a group of service-oriented business processes that estimate the environmental footprint of industrial devices. The environmental data used by the business processes is retrieved from a public environmental database.

5.1 Life Cycle Thinking

As the population of the world keeps growing, environmental issues are getting more and more serious. The volume of industrial products is rising, and it results in higher pollution rates. On the other hand, there is more and more competition for natural resources. As a consequence, industries are working to resolve the real environmental effects caused by their production to enable sustainable development.

Life Cycle Assessment (LCA) is a methodological framework used to estimate how a product affects the environment during its entire life cycle. That is, the cumulative emissions and the resources depletion caused by energy usage, materials usage, assembly, recycling, disposal and so on are included. LCA has a subphase called *Life Cycle Inventory* (LCI) which means the process of collecting and storing environmental data. Another subphase of LCA is called *Life Cycle Impact Assessment* (LCIA) which covers the estimation of how emissions and resources depletion affect the environment. [79]

When LCA is performed for manufacturing equipment, it is not enough to look at the emissions caused by manufacturing the devices. During the years or decades of operation, the industrial process performed by the equipment will both require inputs and produce outputs. Different devices have different environmental footprints: one device may consume less of one resource but more of another resource than other devices. There may also be different emission outputs. Environmental thinking is not only relevant at design time. When obsolete components are replaced, different choices will again have different environmental impacts.

Table 5.1: Some sources of life cycle inventory data.

Source	**Origin**	**Information**
EAA [28] (European Aluminium Association)	Europe	Provides a document with aluminum life cycle information.
EcoInvent [29] (Swiss Centre for Life Cycle Inventories)	Switzerland	Generic LCI database. Access has to be paid for.
ELCD II [30] (European Reference Life Cycle Database)	Europe	Generic life cycle database in XML. Developed and maintained by the European Commission [99].
FEFCO [35] (European Federation of Corrugated Board Manufacturers)	Europe	Life cycle reports of corrugated board production.
GaBi [39] (Ganzheitliche Bilanzierung)	Germany	Life cycle simulation software. A free demo version available for everyone; data sets may not be republished. Uses data taken from BUWAL, EcoInvent and PlasticsEurope databases.
PlasticsEurope [76] (Association of Plastics Manufacturers in Europe)	Europe	Documents that contain information about the amount of energy required for producing plastics and their raw materials.
VTT Lipasto [93]	Finland	Information of emissions caused by traffic.
WorldSteel [100]	International	No data publicly available, but LCI documents are sent by request at no cost.

Several organizations provide LCI databases, and some of them offer their data at no cost while the others need to be paid for (table 5.1). EAA, FEFCO, PlasticsEurope, VTT Lipasto and WorldSteel concentrate on only one industry. EcoInvent is generic, but its data is not available for free. GaBi is an application that utilizes LCI data gathered by some LCI database providers, but it is not possible to utilize its data in another application. Only one of the LCI databases under investigation, ELCD, is offered free of charge in XML format. These factors together with the fact that ELCD does not concentrate on only one industry make it a good alternative for building applications on it.

5.2 ELCD and ILCD

The format of ELCD II is called ILCD (the International Reference Life Cycle Data System). [30] ILCD format was developed as there was no format providing appropriate LCA documentation, enabling the export and the import of data and being transferable to LCA tools. ILCD is based on ISO/TS 14048 standard as well as several LCA data formats that have existed before it. [99]

Several contributions have been made to facilitate the use of ILCD and ELCD. The entire ELCD can be downloaded, and data sets can be added to or removed from the downloaded database with an application called ILCD Editor. Moreover, all the XML schemata and the XSLT documents of ILCD are also downloadable. [58]

ILCD is not only a data format, but there is also a five-book publication set called *ILCD Handbook*. It is a ”series of technical documents that provide detailed guidance on – – Life Cycle Assessment (LCA)”. Even though a general framework of LCA is already given in standards ISO 14040 and ISO 14044, some of their definitions are not exact. As a consequence, the results given by LCA may vary depending on the chosen practices. The avoidance of this problem is one of the goals of ILCD Handbook. Moreover, ILCD Handbook aims to improve the acceptance of LCA among stakeholders and to be suitable for ”everyday decision-making” among them. [47]

ILCD contains several types of XML data sets, but only four of them are relevant in this study: *process data sets*, *flow data sets*, *flow property data sets* and *unit group data sets*. Figure 5.1 shows the most relevant parts of them, using the production of electricity in Finland and some of its outputs as an example. In this study, the most relevant parts of all are the *exchanges* defined in *process data sets*: they represent the inputs and the outputs of a process, including emissions. An important note is that the physical quantity of exchanges is not limited to mass; for example, the radiation caused by a radioactive emission is also given. As no physical unit is directly associated to an exchange, there is no way to resolve or to assume the unit using the information provided by a *process data set*. By retrieving the related *flow data set* and its *flow property data set*, the path finally leads to a *unit group data set* in which the reference unit is defined. [30] This way of associating units indirectly makes it more complex to utilize the database as a total of three documents has to be retrieved when a unit is resolved.

Code 5.1 shows how an exchange definition extracted from ELCD. The relevant parts in this study are the description of the exchange, its direction, its mean amount and the reference to the related *flow data set*.

A *process data set* may contain hundreds of exchanges, but the contents are not limited to them. There are references to the sources of the data, information related to the geographical location to which the data set applies, process classification, descriptions in different languages, information on set data reviewers and so on. The information of some *process data sets* is aggregated from global data while the others concentrate on a specific area. The number of exchanges varies, and different *process data sets* also have different reference years. To give an overview, table 5.2 compares the properties of some process data sets. [30]

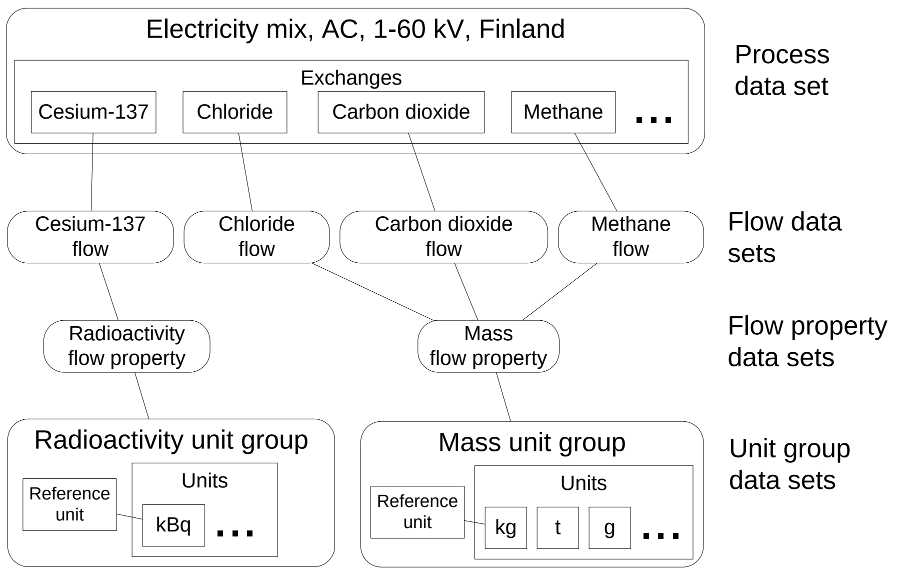


Figure 5.1: Examples of the ELCD data sets that are relevant in this study and their most relevant parts. The example process is the production of electricity in Finland in which all the production types (nuclear power, water power, wind power and so on) are mixed. Based on [30].

```
<exchange dataSetInternalID="152">
  <referenceToFlowDataSet
    refObjectId="fe0acd60-3ddc-11dd-af54-0050c2490048"
    version="02.01.000" type="flow data set"
    uri="../flows/fe0acd60-3ddc-11dd-af54-0050c2490048_
      02.01.000.xml">
    <common:shortDescription xml:lang="en">
      carbon dioxide (Emissions to air)
    </common:shortDescription>
  </referenceToFlowDataSet>
  <exchangeDirection>Output</exchangeDirection>
  <meanAmount>0.473253874538804</meanAmount>
  <resultingAmount>0.473253874538804</resultingAmount>
  <dataSourceType>Mixed primary / secondary</dataSourceType>
  <dataDerivationTypeStatus>
    Unknown derivation
  </dataDerivationTypeStatus>
</exchange>
```

Code 5.1: An example of an exchange element in a process data set document. [30]

Table 5.2: Properties of some *process data sets* provided by ELCD. [30]

Process description	Location	Year	File size	Exchanges
Electricity Mix; AC; consumption mix, at consumer; 1kV-60kV	Finland	2002	302 kB	402
Lorry transport; Euro 0, 1, 2, 3, 4 mix; 22 t total weight, 17.3 t max payload	Europe	2007	117 kB	15
Pre-cast concrete; minimum reinforcement; production mix, at plant; concrete type C20/25, without consideration of casings	Europe	2006	299 kB	407
Steel hot rolled section; blast furnace and electric arc furnace route; production mix, at plant	Global	2000	48 kB	38

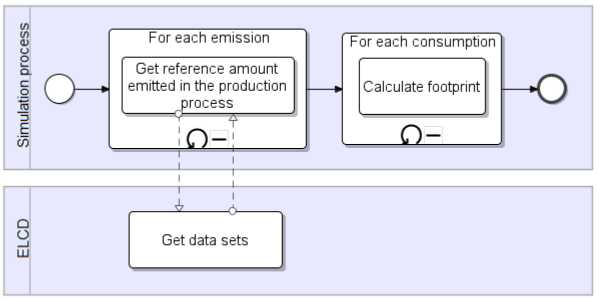


Figure 5.2: The life cycle simulation business process described on a high level.

5.3 Conceptual Framework for Footprint Estimator

5.3.1 Requirements

The data provided by ELCD makes it possible to build a service-oriented business process that calculates footprints caused by devices. Figure 5.2 shows the idea of the life cycle simulation business process on a high level. First, the user launches the business process by giving the information that is required to calculate the footprint. Then, the business process retrieves production process information from ELCD after which the footprints are calculated. Finally, the results are shown to the user.

To calculate a footprint, the first thing to do is to retrieve the related *process data set*. There may be different *process data sets* for different countries or continents, and there may also be several almost similar processes with some detailed differences. Moreover, *process data sets* are named according to their ID number that has nothing to do with the description of the process. Thus, there is no way to conclude the URL from the description of a production process; a search function is necessary to retrieve a process data set from ELCD. A search function is provided by the ELCD website, but there is no way to embed it directly into a business process. All the data sets can be downloaded from the ELCD website. So, it might be possible to write a program that would first read the descriptions of all the production processes and then build a search index for them. However, it would require a lot of such work that is out of the scope of this study. To limit the scope, it is assumed that the user already knows the URL of the relevant *process data set*. To resolve the URL, the user may use the search function on the ELCD website.

To perform calculation, the life cycle simulation business process takes the following input:

- the name of the *process data set file* related to the material that is consumed
- the time the consumption will take (for example, the hours a device operates during its life)
- the interesting amounts of consumption per time unit (for example, per hour when comparing different devices with different consumptions)
- the emissions of interest

The following output will be given:

- the emitted amount of each substance for each consumption given in the input

It was chosen to use Intalio BPMS to implement the business process. There were several reasons for it: there was previous experience on creating service-oriented business processes and processing complex XML documents with it. Moreover, Intalio BPMS can be downloaded and used free of charge, and it is also being continuously developed.

5.3.2 Footprint Calculation

Several different methods could be used to calculate the environmental footprint of a device. For example, points of view could be how high the toxicity of a substance is or how much the substance accelerates the greenhouse effect. However, the purpose of this study is to only demonstrate the integration of a database to service-oriented business processes. Thus, only the output amounts of emissions are calculated.

To keep footprint calculation simple, several assumptions are made. Let us assume that the resources consumption of the production of the device itself is minimal compared to the emissions caused by the material it consumes during its life[1]. For example, if a valve consumes electricity, its life is so long that the footprint of the production of the valve itself is very low compared to the amount of electricity it

[1]It is not necessarily so: for example, manufacturing a car or a mining truck requires a lot of resources.

will use[2]. Furthermore, it is assumed that none of the emissions can be collected in production processes. A thing to note is that a substance may be both an input and an output: for example, some reaction could both consume carbon dioxide from air and emit it to air which would reduce the net emitted amount. However, such consumptions are assumed so small that they do not have to be considered.

To calculate the footprint of one emission caused by the consumption of one material, the total operating time of the device during its life has to be known. It is given by equation 5.1 which is the product of device operation per day in hours (O_d), device operation per year in days (O_y) and the total life of the device in years(L). The unit of the result will be one hour.

$$O_T = O_d O_y L \tag{5.1}$$

When multiplying several numbers, it is important to consider the risk of an overflow. As the total operating time of a device can be decades and it is given in hours, the resulting number can be big. For example, if a device operates 24 hours every day and its life is 40 years, the resulting number of hours is $24 \times 365.25 \times 40 = 350640$. However, it is not a big multiplier when floating point numbers are used in calculation. Thus, the risk of overflows is considered minimal.

Now that the total operating time of the device is known, the next thing to calculate is the emission that the device causes per hour (equation 5.2). C is the amount of the material consumed per hour and E_R is the emitted amount when the reference amount C_R of the material is consumed.

$$E = \frac{CE_R}{C_R} \tag{5.2}$$

The total footprint of the device is calculated as the product of O_T and E as shown in equation 5.3.

$$F = O_T E = O_d O_y L \frac{CE_R}{C_R} \tag{5.3}$$

For example, with the following values, the footprint will be calculated as in equation 5.4.

- The device operates 24 hours per day.
- The device is used every day (365.25 days per year, leap years considered).
- The life of the device is 15 years.
- The device consumes 6.5 megajoules (MJ) of electricity per hour.
- The reference amount of electricity production is 3.6 MJ.

[2]The consumption of electricity or other resources can typically be found in the device specifications given by the manufacturer.

- The emitted amount of the substance of interest is 0.0057 kilograms (kg) per reference amount (3.6 MJ) in electricity production.

$$F = 24\ {}^{\mathrm{h}}/_{\mathrm{d}} \times 365.25\ {}^{\mathrm{d}}/_{\mathrm{a}} \times 15\ \mathrm{a} \times \frac{6.5\ {}^{\mathrm{MJ}}/_{\mathrm{h}} \times 0.0057\ \mathrm{kg}}{3.6\ \mathrm{MJ}} = 1353.25\ \mathrm{kg} \qquad (5.4)$$

Equation 5.3 calculates the footprint of only one device, one material and one emission. To calculate the footprint for several substances or for several different resources consumed, the equation is simply used once for each value to be calculated.

5.4 Footprint Estimator Architecture

5.4.1 Architecture – the First Attempt

When software research is made with a little experience in the related domain, it is typical that the first solution that comes into mind may not be the best one. A solution that seems straightforward and easy to implement in the beginning may prove to be a bad choice for one reason or another. The solution is typically supposed to meet several requirements that may be simplicity, maintainability, performance and so on. There is a danger that a simple design results in high resources consumption and a low performance. That is what happened with the first architecture of the footprint estimator business process.

Figure 5.3 describes the first architecture of the solution. It is a tree that consists of two types of nodes: the squares represent data sets provided by ELCD whereas the circles represent business processes that process the information of the data sets. The purposes of the business processes and the resources in the figure are explained in the following paragraphs. As the design method used was bottom-up, the business processes will be explained beginning from the lowest level of the hierarchy.

GetSubstanceExchanges extracts the substance exchanges of a production process that are relevant according to given parameters. As the parameters, it takes the name of the relevant *process data set* file, the exchange direction and the substance of interest. As an example, let us look at retrieving the emitted amount of chloride when the reference amount of electricity is produced in Finland. The parameters shall be the name of the *process data set* file of electricity production in Finland, "chloride" as the substance parameter and "output" as the exchange direction. Then, the business process extracts all the chloride outputs, calculates their sum and returns them.

As *process data sets* do not contain any information on the physical units of exchanges, they have to be resolved separately. To keep *GetSubstanceExchanges* simple, a separate business process, **GetFlowProperties**, is implemented. As the

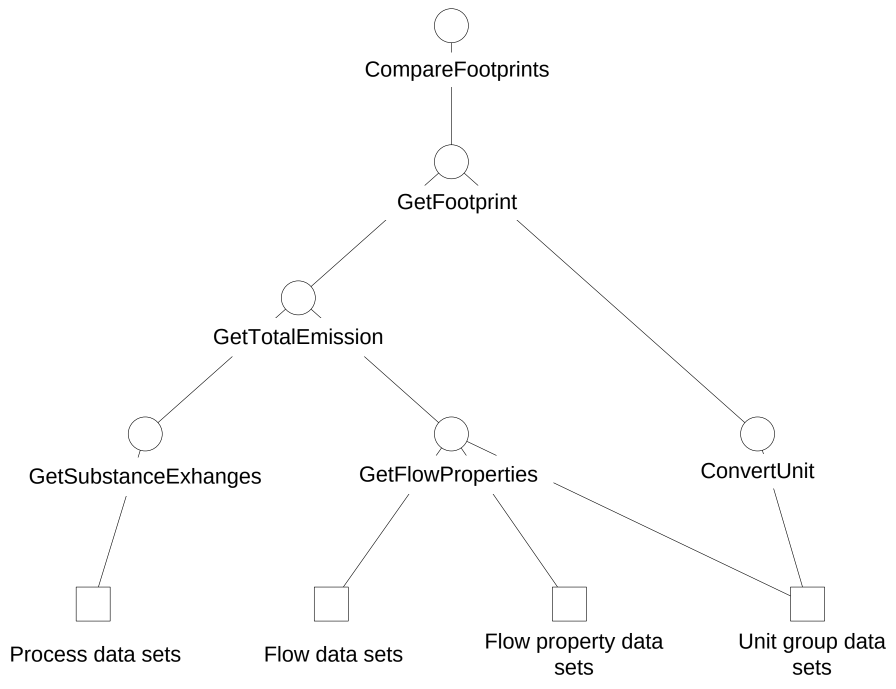


Figure 5.3: The architecture of the first version of the life cycle simulation business process.

input, *GetFlowProperties* takes the name of the related *flow data set* file. Then, it finds out the reference unit of the related substance flow by retrieving the related *flow data set*, *flow properties data set* and *unit group data set.* In addition to the unit of the flow, the name of the *unit group data set* file is also returned as it may be required later for unit conversions.

The task of **GetTotalEmission** is to compose *GetSubstanceExchanges* and *Get-FlowProperties.* It retrieves the exchanges of the emissions that are relevant to its parameters and combines that information with the corresponding physical unit. No individual exchanges are returned; instead, only the total sum of all the outputs is returned. As it is required on the higher levels of the hierarchy, the reference unit defined in the *process data set* file is also returned. Like *GetFlowProperties*, *GetTotalEmission* also returns the name of the relevant *unit group data set.*

ConvertUnit calculates a conversion from one physical unit to another. The parameters it takes are the name of the relevant *unit group data set* file, the original unit, the target unit and the amount to be converted. *ConvertUnit* can be used if the reference unit used in a process data set does not correspond to the amount of consumption that has been given when calculating a footprint. For example, if electricity consumption is given in kWh (kilowatt hours), *ConvertUnit* can convert it into MJ (megajoule).

GetFootprint uses *GetTotalEmission* and *ConvertUnit* business processes. Its purpose is to calculate the footprint of one emission when one material is consumed. It takes the name of the related *process data set file*, the emission of interest and the

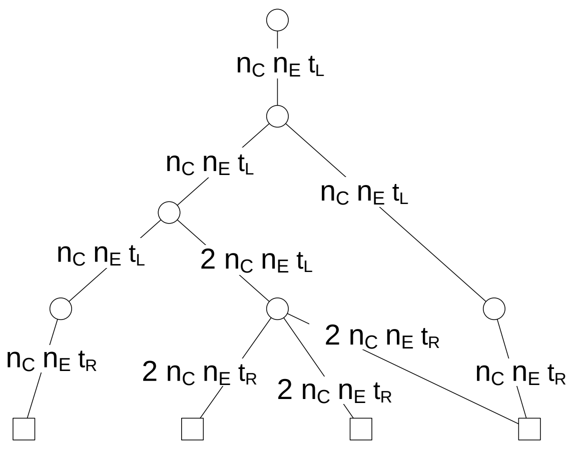


Figure 5.4: The total time spent on requesting each service in the hierarchy.

information related to consumption duration as the parameters. The business process calls *GetTotalEmission* to resolve the amount of emission and its physical unit when the reference amount of the related material is consumed. Then, it converts the given consumption unit to the reference unit of the production process. Finally, the footprint is calculated.

CompareFootprints takes several emissions of interest, several consumptions and the name of the related *process data set* file as its parameters. Then, it forms all the possible pairs of given consumptions and emissions and calls *GetFootprint* for each of them. Despite its name, the business process does not perform any comparison on emission amounts, but such a comparison is easy to perform using the return value.

The business processes in the resulting hierarchy are simple and easy to understand, but there is one trade-off, namely performance. *CompareFootprints* forms all the possible pairs of given emissions of interest and given consumptions. Thus, the resulting amount of requests to *GetFootprint* is the product of the numbers of the emissions of interest and the consumptions. The business processes below it in the hierarchy execute each service call once or twice depending on the service. If all the business processes are run on the same machine, their interaction will be local. Five of the services are such. Five more connections remain, and all of them are between the business processes and the ELCD server which means they are remote. Let n_C be the number of consumptions, n_E be the number of the emissions of interest, t_L be the time required for a local service call and t_R be the time required for a remote service call. When *CompareFootprints* is called, the time spent calling each resource will be as shown in figure 5.4. If the time required to run a business process is considered small compared to the time required for resource calls, the resulting total response time can be calculated as shown in equation 5.5.

$$t = n_C n_E (6t_L + 8t_R) \tag{5.5}$$

There is probably a significant difference between the request times to a local resource and to a remote resource. Despite that, to simplify calculation, both the request times are marked with t_S. Equation 5.6 results.

$$t = n_C n_E (6t_S + 8t_S) = 14 n_C n_E t_S \tag{5.6}$$

Let us examine a request that contains four consumptions (n_C) and four emissions of interest (n_E), which is not a high number. The resulting multiplier of t_S is $14 \times 4 \times 4 = 224$, which is a big number of service requests considering that only four consumptions and four emissions are requested. It is clear that such an architecture cannot be efficient. However, these calculations were made only after implementing the architecture. In experiments, *CompareFootprints* was called with two consumptions and four emissions of interest (that is, *CompareFootprints* calls the hierarchy below it eight times). The response time of the business process was dozens of seconds but it succeeded. When the number of consumptions was raised to three (resulting in 12 calls to the hierarchy), the client application (soapUI) gave a request timeout after 50 seconds. The experiments showed what was noticed earlier theoretically: the architecture had to be redesigned.

There were several reasons for the architectural failure. The main reason was that it was not expected that service interactions would take such a long time. In addition, a principle that is good for a non-distributed object-oriented application may not be good in service-oriented design. In object-oriented programming, a class definition is often good if it is simple because it makes the design of the class easy and straightforward. In this service-oriented application, the low-level business processes in the hierarchy were designed simple, but it degraded the performance because it requires more service calls. Performance is always present in a distributed application. Thus, a sensible approach would be to minimize the number of service calls and to try to design each service so that it repeats tasks instead of forcing a service consumer repeat requests. Making a service able to accomplish a higher number of similar tasks at a time does not make it any worse from the consumer point of view: it can still perform the task only once when such a request comes. However, it makes the design more difficult as the *granularity* of business processes has to be raised.

5.4.2 Optimized Architecture

After one architecture with a poor performance had been implemented, the goal of the new architecture was to minimize the number of service requests. In the

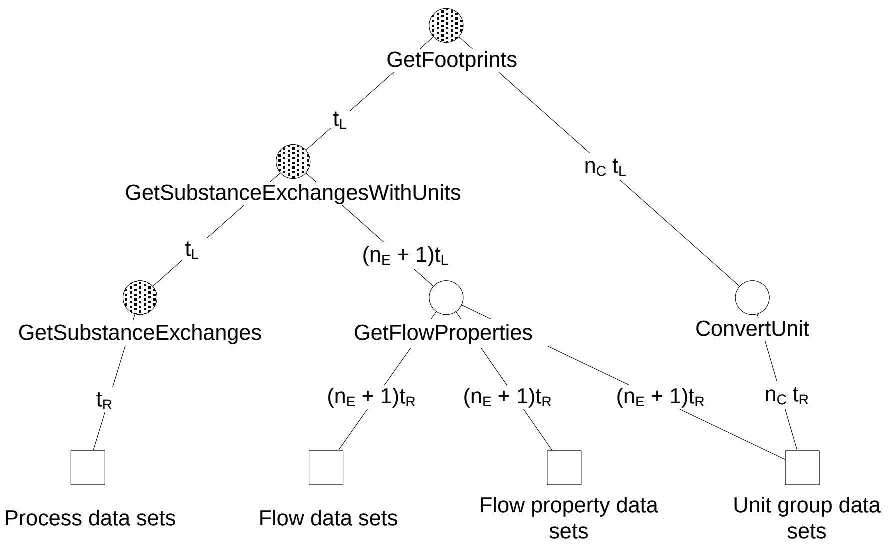


Figure 5.5: The total time spent on requesting each service in the optimized architecture. A dotted circle indicates that a service is new whereas the others were reused from the first version.

first inefficient version, the philosophy was to keep the granularity of the low-level business processes low and make the business process at the top of the hierarchy the only complex one. To improve the performance, it is necessary to distribute the complexity to lower levels.

An important point is that as all the calculation is related to one *process data set*, it is enough to retrieve the reference values of the emissions defined in it only once. Then, the reference values can be multiplied according to each amount of consumption given by the user. As the entire business process tree from the top of the hierarchy to a *process data set* was called $n_C \times n_E$ times (once for each consumption–emission pair) in the first architecture, the new architecture would call the tree only once. As *process data sets* are typically large, the change is assumed to improve performance significantly. Moreover, the change does not only affect the time spent on processing *process data sets* but it also reduces the number of service calls in the entire hierarchy.

Figure 5.5 shows the optimized architecture and time spent requesting each service in one hierarchy call. **GetSubstanceExchanges** was redesigned to extract the relevant exchanges of several substances in one request and it is now called only once per each call to the hierarchy. Similarly, the new **GetSubstanceExchangesWith-Units** that replaced *GetTotalEmission* now processes several emissions in one request. Furthermore, the changes in *GetSubstanceExchanges* and *GetSubstanceEx-changesWithUnits* make it possible to process several emissions simultaneously on the level above them. As a result, the new **GetFootprints** can replace both *Com-pareFootprints* and *GetFootprint*.

Not all the business processes were rebuilt as *GetFlowProperties* and *ConvertUnit* were reused from the first architecture. As *GetFlowProperties* can only retrieve the unit of one exchange, it has to be called once for each emission anyway (it will be called $n_E + 1$ times as it also retrieves the reference unit of the product that results from the process). Similarly, *ConvertUnit* is called once for each consumption amount. The calls to *ConvertUnit* could be further optimized so that if there are several consumption values with the same unit, it would be called only once for all of them. However, it would raise the complexity of *GetFootprints*.

The three new business processes that replace four previous ones affect the number of requests in the hierarchy significantly. If the time used for business process execution is again assumed small compared to the time of communications, the total time required to execute *GetFootprints* is given by equation 5.7.

$$t = (n_C + n_E + 3)t_L + (n_C + 3n_E + 4)t_R \tag{5.7}$$

If the equation is simplified by using the same variable for the times required by local requests and remote requests (t_S), equation 5.8 results.

$$t = (n_C + n_E + 3)t_S + (n_C + 3n_E + 4)t_S = (2n_C + 4n_E + 7)t_S \tag{5.8}$$

Table 5.3 compares the response times of the requests made to the original architecture and to the optimized architecture when the response times are calculated using equations 5.6 and 5.8. It can be stated that the more consumptions and emissions are given, the better the optimized architecture is compared to the original one. However, the equations do not consider that the response times of different resources may be different. Especially a request of a *process data set* takes probably longer than a request to other resources because *process data sets* are the biggest of all the documents requested remotely. It is remarkable because the original architecture calls a *process data set* $n_C \times n_E$ times whereas the optimized architecture calls it only once. Thus, it is expected that the ratio between the real response times of the original and the optimized architecture is even higher. No detailed experiments were made to compare the two architectures, but it was seen that an example input that would result in a timeout in the original architecture was completed in less than 20 seconds in the optimized one.

The architecture could be optimized even more. In practice, all the emissions processed during the work had either kilogram (for mass) or kilobecquerel (for radiation) as the unit. As all the radioactive emissions had an isotope number at the end of their names (for example, "cesium-137"), it could be assumed that all of them have kilobecquerel as the unit. The rest would then be assumed to have kilogram as the unit. This way, it would be needless to retrieve *flow data sets*, *flow property*

Table 5.3: A comparison of the theoretical factors of t_C in the original and in the optimized hierarchy.

n_C	n_E	$14n_Cn_E$	$2n_C + 4n_E + 7$	Ratio
1	1	14	13	1.08
2	2	56	19	2.95
4	4	224	31	7.23
8	8	896	55	16.3
8	16	1792	87	20.6
16	16	3584	103	34.8

data sets or *unit group data sets* – actually, the only dataset required to resolve the reference emissions would be one *process data set. Unit group data sets* are used for unit conversions as well – however, they could be saved to a local cache to avoid the need to retrieve them for each conversion. Assumptions like this raise the risk of errors while running the business processes. In the optimal situation, the structure of ELCD was such that the units were given in process data sets. It would both make it easier to create applications with a good performance and eliminate the need for speculations whether assumptions can be made or not.

As the entire ELCD is available for download as one package, it could also be stored and used in a local machine, which would eliminate the need for remote resource calls. The disadvantage of this solution is that whenever data sets are updated, the most recent data would not be available. However, the data sets on the local machine could be updated regularly (for example, once a week).

5.5 Footprint Estimator Implementation

5.5.1 Business Process Design

In this subsection, the implementation details of the *GetFootprints* business process hierarchy are presented. Then, a brief comparison is made on the complexity of the original architecture and the optimized architecture.

The BPMN diagrams of the business processes are presented among the text, but the input and the output XML schemata of each business process are presented in appendix A. In each of the BPMN diagrams, an additional tree diagram is presented to indicate in which part of the business process hierarchy the related business process is located. All the XML schemata are presented as UML class diagrams. By default, each class box in a diagram represents a complex type. However, if the class box contains text "from xsd", it shall be interpreted as an XML base type. Furthermore, all the "member variables" in the class boxes shall be interpreted as

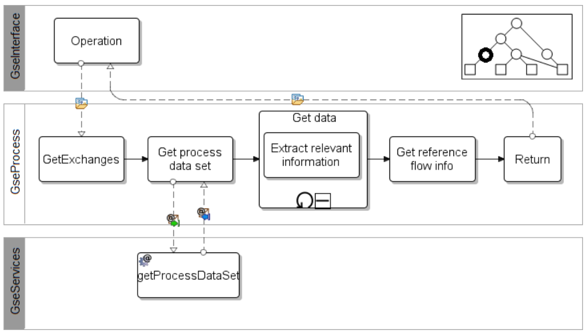


Figure 5.6: The BPMN diagram of GetSubstanceExchanges business process.

base type elements. No XML attributes have been used in the XML schemata defined for this study.

In each business process, it is assumed that the path of each data set type remains the same. That is, instead of taking the entire URL of each data set file as the parameter when retrieving information, only the name of the file is taken. There is a risk that the business processes get broken if the absolute location of ELCD or the data sets is changed. However, if it happens, it does not require a lot of work to change the paths defined for the data sets and to redeploy the business processes to an Intalio server.

The XML schema of **GetSubstanceExchanges** is in figure A.1. As the input, the business process takes the name of the related *process data set* file and the name of the emissions whose information will be extracted. The output contains the following information: the geographical location of the process, the name of the *unit group data set* file related to the reference product flow of the process. Furthermore, the emitted mean amount and the name of the related *flow data set* file are presented for each emission. As the units of substance exchanges cannot be directly seen in a *process data set*, the unit field of the output has to be left empty. However, the field is included in the output as it enables the reuse of the entire XML schema when the units are retrieved later.

The flow of *GetSubstanceExchanges* is presented in figure 5.6. After the business process has begun in the "GetExchanges" task, the relevant *process data set* is retrieved. Then, the relevant information is extracted from the *process data set* by executing "Extract relevant information" task for each emission name that has been given in the input. The extraction requires a total of three XSL transformations. The first one calculates the total output of the emission, the second one extracts file name of the related *flow data set* and the third one assigns the extracted information into the output. After the emissions information has been assigned, the relevant process information is extracted in the task "Get reference flow info" – again, an XSL transformation is used. Finally, the business process ends in the "Return" task.

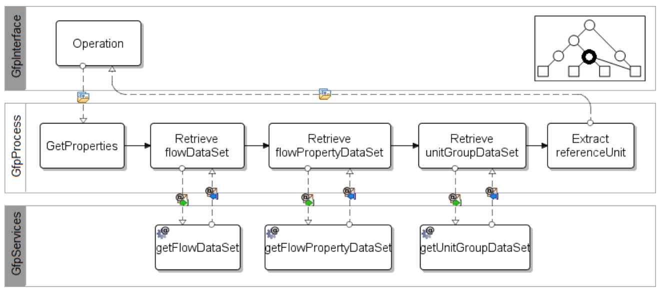


Figure 5.7: The BPMN diagram of GetFlowProperties business process.

GetFlowProperties has the XML schema shown in figure A.2. The only input is the file name of the related *flow data set*. The output contains the name of the related material, the name of the unit in which the material is measured, the physical quantity related to the unit and finally the file name of the related *unit group data set*.

Figure 5.7 illustrates the flow of *GetFlowProperties*. As the primary goal of the business process is to retrieve the unit of a flow, the required data sets are retrieved one by one. In the last task, the unit information is extracted after which the business process returns its output.

GetSubstanceExchangesWithUnits does not have an XML schema of it own as it reuses the schema of *GetSubstanceExchanges* (figure A.1). The output differs in that *GetSubstanceExchangesWithUnits* adds the units of the emissions to the output.

The flow of *GetSubstanceExchangesWithUnits* is presented in figure 5.8. First, the relevant emissions are retrieved by calling *GetSubstanceExchanges* business process in the ”Get exchanges” task. Then, they are assigned to the return value. The reference information of the production process is retrieved in ”Get reference flow properties” task by calling *GetFlowProperties* business process after which the values are assigned. Then, the units of the emissions are retrieved by calling *GetFlowProperties* for all of them after which each unit can be assigned – in the assignment, an XSL transformation is used. Finally, the output of the business process is returned.

ConvertUnit has the input and the output presented in figure A.3. In the input, the original amount, its unit and the target unit to which the amount shall be converted are given. The business process has no means to find the right data set file according to given units. Thus, it requires also the name of the related *unit group data set* file. As the output, the business process simply returns the given amount converted to the new unit.

ConvertUnit has a simple flow as described in figure 5.9. First, ”Get unit group” task retrieves the related *unit group data set*. Then, using the contents of the data

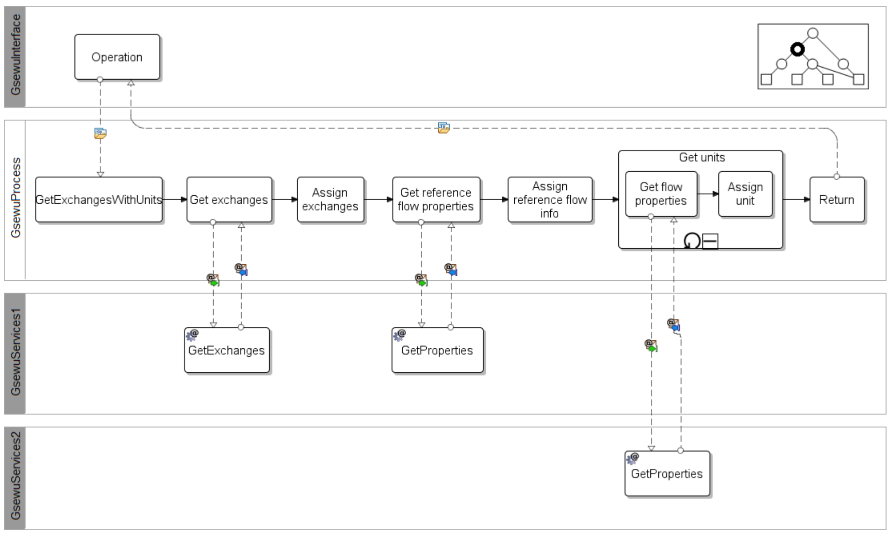


Figure 5.8: The BPMN diagram of GetSubstanceExchangesWithUnits business process.

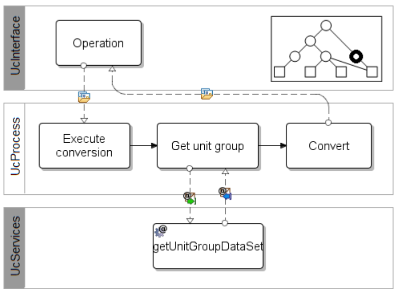


Figure 5.9: The BPMN diagram of ConvertUnit business process.

set, the unit conversion is performed using an XSL transformation after which the converted value is returned.

The XML schema of **GetFootprints** in shown in figure A.4. The input contains the file name of the related *process data set*, the total operating time to be used when calculating total emissions, the emissions of interest and the consumption amounts and their units to be compared. In the output, the name of the process and the location of it are provided. After them, there is an entry for each consumption amount to be compared that contains the information how much of the emissions of interest will be produced with that consumption. *StringList* and *substanceExchangesInfos* types are reused from the XML schema of *GetSubstanceExchanges* business process.

The flow of *GetFootprints* is presented in figure 5.10. First, the relevant emission outputs are retrieved using *GetExchangesWithUnits* business process. Then, "Create consumption entries" task executes an XSL transformation to create an entry into the output for each consumption amount given. Once the consumption entries have been created, entries for the emission amounts will be created into them using XSLT. Then, "Calculate conversions" task converts each of the given consumptions into the

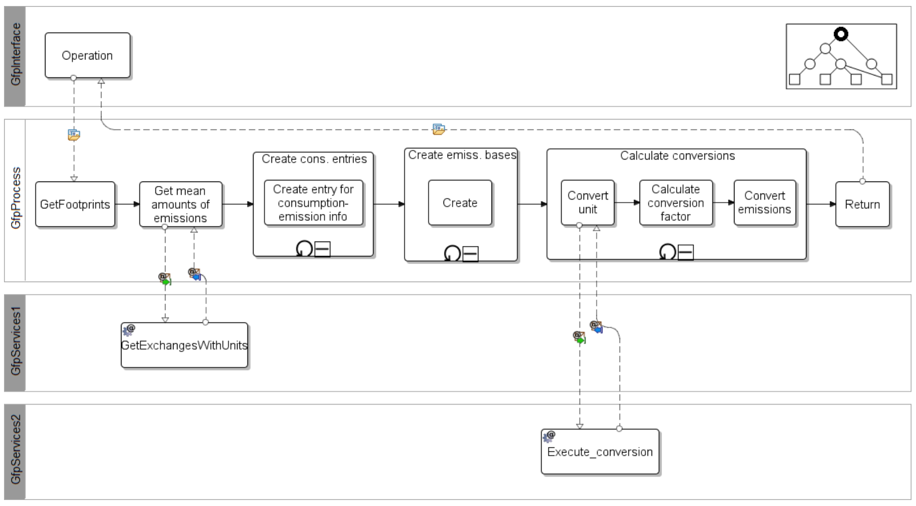


Figure 5.10: The BPMN diagram of GetFootprints business process.

Table 5.4: Comparison of some properties of the business processes in the original and in the optimized architecture.

Property	**Original arch.**	**Optimized arch.**
Business processes	6	5
XSLT documents	4	9
Total loops in business processes	3	5

reference unit of the process. Finally, the resulting emission amounts are calculated for each consumption and assigned using XSLT. Then, the output is ready to be returned.

As shown in the XML schema of *GetFootprints* (figure A.4), the output is supposed to include the name of the related *flow data set* file for each emission. However, the information of each emission is assigned using XSLT, and Intalio BPMS only allows the use of three external parameters in an XSL transformation. Thus, the name of the *flow data set* file cannot be passed because entries ”substance”, ”meanAmount” and ”unit” already take the places. There is no need for *flow data set* files information on this level of the business process hierarchy anyway. However, as *substanceExchangeInfo* type is originally defined in the schema of *GetSubstance-Exchanges*, it cannot be removed.

Table 5.4 compares some properties of the original architecture and the optimized architecture. The number of business processes is even lower in the optimized architecture. However, as the number of XSLT documents and the number of loops indicate, the optimized architecture is more complex than the original one. The higher granularity in the low-level business processes results in a better performance. Another consequence is that more data has to be passed in one service call to the low-level business processes. It results in more complex data processing.

5.5.2 Data Extraction from ELCD

As ILCD format uses XML, it is supposed to be easy to process the contents ELCD. Despite that, the complexity of ILCD is a challenge. Considering all the limitations and the possibilities related to the structure of the XML documents would result in a lot of work. The purpose of this study is to demonstrate one possible way of using ELCD. Thus, the details of the document structures defined in the XML schemata are not considered. Instead, the structures of a few data sets are used as reference structures to be used in data extraction.

There are several details about the ILCD XML schemata that permit a document with insufficient contents. For example, in a *process data set*, the *exchangeDirection* element in an exchange definition is optional. A missing exchange direction would make it impossible to identify whether an exchange is an emission or not. So, it has to be assumed that the exchange direction is always given. Another issue are the references to *flow data set* files. As far as is known, all the exchanges have a reference to a *flow data set* that provides additional information about the substance flow. Such references are given using the complex type *GlobalReferenceType* that may contain an element called *shortDescription*. The role of *shortDescription* is essential in exchange definitions as it provides the name of the related substance. However, according to the XML schemata, it is not obligatory. A reason to not require it in the complex type definition could be reusability: if the requirements of *GlobalReferenceType* are less strict, its reuse potential is higher among different XML document types. However, the consequence of this loose complex type definition is that a *process data set* would be valid even if its exchanges had no substance defined at all. Thus, it may be questioned whether it is a good design choice.

The assumptions are not limited to XML structures but their contents are also involved. It is assumed that the descriptions of exchanges in *process data sets* are given as below. For example, a description could be ”chloride (Emissions to air)”.

```
substanceName + " (" + emissionType + ")"
```

The pattern for the assumption was chosen after a few randomly chosen *process data sets* had been investigated. As ELCD uses such a pattern in exchange descriptions, an attempt to find a description that matches exactly a substance name would never find anything (for example, a search for ”chloride”). Moreover, an exact match can neither be used in a substring search because one substance name can be a substring of another (for example, ”hydrogen” is a substring of ”hydrogen arsenide” and ”hydrogen fluoride”). As emission types are not considered in this context – that is, emissions to air, water and soil are equally calculated – they can be dropped away from the search pattern. Thus, it is assumed that a search for descriptions that *begin with* the pattern below returns all the relevant exchanges.

```
substanceName + " ("
```

Having to use such a pattern to find substrings raises a question about the practices of good database design. As there is a general principle that all the data items should be atomic, there should be no need to divide data units in parts like here. Even though it would make the structure of XML documents more complex, it would also make them easier to process.

To further simplify the data extraction, it is assumed that the exchanges of one substance name are given using the same physical unit. There are at least two physical quantities that are used: mass and radioactivity. A substance has always a mass, but it may also contain isotopes that are radioactive (for example, ”cobalt” and its radioactive isotope ”cobalt-58”). It is assumed that an emission of radioactivity is always given with the isotope number after the substance name. Thus, if ”cobalt” is requested instead of ”cobalt-58” or another isotope, exchanges with radioactivity as the quantity are never received. Another assumption related to physical units is that the same unit is always used for the same quantity (for example, the only unit used for mass is ”kg”). This assumption is not considered a potential problem as each unit group has exactly one reference unit. The result of this assumption is that when there are several exchanges of the same substance in a process, they can be summed directly without any unit conversions. It not only makes the implementation of business processes easier but it also causes less network load as fewer data sets are retrieved.

To conclude, several assumptions were made to extract data from ILCD. Some of them are not likely to cause problems. However, the assumptions related to exchange description strings could be problematic. If there are any exchanges whose description patterns are different to the assumed pattern, the result could be either that not all relevant exchanges are recognized or that irrelevant exchanges are returned. If any physical units related assumptions failed, it would result in incorrect calculations. However, even though they raise the risk of failures, current assumptions are acceptable for this study as the purpose is only to demonstrate the use of the ELCD.

5.5.3 HTTP Request Challenge

The data sets of ELCD are retrieved as XML-over-HTTP, which is supposed to be a trivial task in Intalio BPMS. There is no difficulty creating a REST service binding in Intalio Designer, but the attempt to use such a binding to the ELCD database did not succeed. As the corresponding request did not cause problems when it was executed by soapUI, a protocol analyzer was taken to use to investigate the differences of the requests made by Intalio server and soapUI. The protocol analyzer captured the following HTTP request submitted by Intalio server:

```
GET http://lca.jrc.ec.europa.eu/lcainfohub/datasets/elcd/
processes/6dd69400-9e1d-4376-a6f3-260877acd194_02.01.000.xml
HTTP/1.1
Accept-Encoding: gzip,deflate
User-Agent: Jakarta Commons-HttpClient/3.1
Host: lca.jrc.ec.europa.eu
```

The following request was submitted by soapUI:

```
GET /lcainfohub/datasets/elcd/processes/6dd69400-9e1d-4376-
a6f3-260877acd194_02.01.000.xml HTTP/1.1
Accept-Encoding: gzip,deflate
User-Agent: Jakarta Commons-HttpClient/3.1
Host: lca.jrc.ec.europa.eu
```

The only difference between the requests is that Intalio uses an *absolute URI* as the *request URI* (the URI after ”GET”) whereas soapUI uses an *absolute path.* To make sure that the failures were really caused by this detail, a simple software was written in Java to try HTTP requests with different headers. The results were as expected. Using an absolute path in a request returned the requested file. In contrast, an absolute URI returned a HTML page that would redirect the client to another page which indicates that the server did not understand the request.

As HTTP is a widely used standard, there should be no compatibility problems in such a trivial thing as the request URI. An interesting question is which one acts the wrong way, Intalio server as the client or the ELCD server. In HTTP 1.1 standard, the request URI is defined as follows [80]:

```
Request-URI = "*" | absoluteURI | abs_path | authority
```

The standard gives following request line examples when using (1) an *absoluteURI* or (2) an *abs_path* [80]:

```
(1) GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1
(2) GET /pub/WWW/TheProject.html HTTP/1.1
```

As said in the standard, both representations *must* be accepted by all HTTP 1.1 servers. However, it is also said that a client ”will” only use an *absoluteURI* when it calls a proxy server. [80] On the one hand, the server used by ELCD (Microsoft IIS 6.0 according to HTTP response headers) does not accept an *absoluteURI* in the request even though it should. On the other hand, Intalio server uses *absoluteURI*

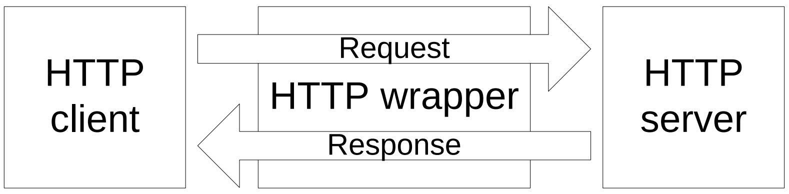


Figure 5.11: The idea of a HTTP wrapper.

in the request even though it is not expected according to the standard. That is, neither the ELCD server nor the Intalio server as the HTTP client complies to HTTP 1.1 standard. In this case, compatibility would be reached if even either of them were compliant to the standard.

Even though Intalio has a feature to create business process specific custom headers to HTTP requests, there is no way to configure the format of the request URI. Thus, Intalio and Microsoft IIS are simply incompatible.

However, there is a straightforward solution of submitting HTTP requests indirectly by using a wrapper. The basic idea of a wrapper is simple: it operates between the actual client and the actual server and rebuilds the requests submitted by the client (figure 5.11). The wrapper used in this particular solution has been implemented in Java, which means that it can be run in any Java capable web server. As Intalio server has such a capability, no separate web server is required. Thus, to use the resources of ELCD, the wrapper is first deployed to the Intalio server being used to run the business processes. Then, when creating any business process that uses ELCD, all the requests are routed through the wrapper instead of directly calling the ELCD server.

5.5.4 XSLT Usage

XSLT is an essential technology in the implementation of *GetFootprints* business process as almost all relevant data is extracted with it from the XML documents of ELCD. XSLT offers means to process large XML documents and it has a built-in support in Intalio BPMS. Another technology that could be used is XQuery which is also supported by Intalio BPMS. However, as neither an example nor a tutorial of XQuery usage in Intalio BPMS was found, it was decided to use XSLT instead.

Despite the XSLT support of Intalio BPMS, it is not a good environment for XSLT development. Testing any change made to an XSLT document requires the entire business process to be redeployed to an Intalio server which slows down working. There are different XSLT processors available: for example, several WWW browsers such as Opera have a built-in XSLT support. There is also an open-source alternative Saxon which is also used by Intalio BPMS during business process execution. The standalone version of Saxon is used from the command line whereas

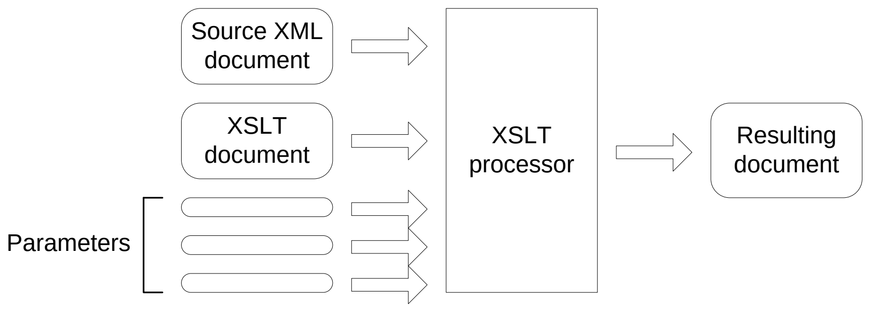


Figure 5.12: The way of executing XSL transformations in the business processes of this study.

Opera has a graphical user interface. Opera is more intuitive: it has the means to show the result of the transformation and highlight different parts of the resulting XML document. However, Opera does not support all the functions used in the most recent version of XSLT. In addition, as Saxon is used by Intalio BPMS as well, any XSLT document developed with it are supposed to work in a business process as well. As the command line usage of Saxon is not difficult, it was decided to use it instead of trying different WWW browsers or other XSLT processors.

Figure 5.12 illustrates the way of using XSLT in an Intalio BPMS business process. A transformation has two obligatory inputs: the source XML document to be transformed and the XSLT document that defines the transformation. Optionally, the transformation can also take external parameters. All the inputs are given to the XSLT processor that executes the transformation and returns a document as the output. Depending on the XSLT document, the output is not necessarily XML. In some cases of this study, the resulting document is a single numeric value or a string that has been extracted from the source XML document according to given parameters.

XSLT may not always be an optimal way to extract data from XML documents. When calculating the total emission of a substance in the *GetSubstanceExchanges* business process, it was assumed that all the emission flows of the same substance have the same quantity and the same physical unit. It simplifies the business process as the unit is only retrieved for one emission; only one *flow data set* file name needs to be returned. However, with XSLT, it is impossible to retrieve only one matching item from an XML document if several matches are found. A solution for this problem is to return a string that contains all the returned file names. Then, as the expected length of a *flow data set* file name is known beforehand, a substring with that length is taken from the beginning of the string. It works, but it is anything but robust.

5.5.5 Experiences of BPMN

Looking at figures 5.8 and 5.10, it can be questioned whether a good understandability can always be reached when defining an executable business process in BPMN. When a BPMN diagram is drawn purely from the design point of view and the execution details of the related business process are not considered, the power of expression of the diagram is the only goal. Only the tasks that are important to make the flow understandable are required to be drawn. However, it is different with executable processes.

When an executable business process is created, more tasks are required. The process has to be described on such a level of detail that a respective executable can be created. Even if it were be possible to convert an intuitive diagram to an executable format, it may require some of the tasks to contain very complex actions. In Intalio BPMS for instance, complex assignments are performed with XSL transformations. If there are several XSL transformations in one task, it may be difficult to process. An approach that is both easier to implement and that lowers the danger of errors is to split the task. However, the result may be that there are several tasks whose only intention is to describe assignments and other actions that actually belong to the WS-BPEL level. So, the actual flow of the diagram may be difficult to understand in the way it is meant by the developers of BPMN. Thus, to reach both a good understandability and an easy implementation, it may be useful to create two diagrams. The goal of one is to be understandable to anyone, and the other that is created from the execution point of view.

5.5.6 Using Intalio BPMS

In principle, composing a business process with Intalio Designer is straightforward and easy. For example, to call a Web Service in a business process, the related WSDL description and the XML schemata are imported. Then, any operation of the Web Service can be dragged and dropped into the BPMN diagram of the business process. The graphical and intuitive Mapper view is used to assign variables and XML fields to each other; it is also used to assign the output and the input values of Web Service calls. However, especially when complex XML documents are processed, problems may occur. There are also certain actions that will cause problems while creating or running a business process that may not be obvious to a user. As Intalio BPMS consists of several parts made by different organizations, it is sometimes difficult to know whether an error has been occurred in Intalio Designer, Intalio server or another component such as Apache ODE.

The most significant problem about Intalio BPMS is that whenever a business process is modified, there is a risk that it gets broken. After renaming files or diagram elements or after modifying the XML schemata used by a business process, Intalio Designer fails often to update the references. An experienced user may be able to fix the process manually – however, it is typical that the entire business process has to be rebuilt. To minimize the problems caused by this, it is meaningful to try to split business processes to small entities. Then, if a business process gets broken, only a restricted functionality has to be rebuilt.

If problems occur requesting a service, Intalio server does not provide many means to resolve what has caused the error. There is no way to investigate the response returned by a service if it is other than what was expected. By investigating the information provided by Intalio server, it may be possible to see the HTTP status code returned by the service, but the information provided by it is limited. Neither can the request that caused the failure be investigated. It would be useful especially when the request has not been defined using a WSDL file but manually by the user. Due to the insufficient error information, a protocol analyzer has to be used often to find the cause of errors.

A problem was also encountered assigning values to XML sequences. WS-BPEL provides the "[ ]" modifier to choose exactly one element from a sequence. For some reason, an integer variable used for such indexing when assigning to a sequence results in a warning message of multiple selected nodes. When a business process containing such an assignment is run in an Intalio server, the execution ends to an error message. For some reason, the same expression works with no problems when a value is being copied *from* a sequence. Later, a solution was discovered: even though using an integer variable, a type conversion has to be made for it using *number()* function. However, this kind of requirement may not be expected by a user. As there was no working solution during the development of the footprint estimator, such assignments were performed with XSL transformations, which is a complex way to do it.

XSLT is a powerful way to modify XML documents, but there is a limitation in Intalio BPMS that no more than three external parameters can be used in an XSL transformation. To assign contents using an XSL transformation, the most straightforward way is to pass it as external parameters. However, due to the limitation, only three variables can be assigned at a time. If the document being processed is complex, it is a serious limitation. Naturally, it can be overcome by performing several or more complex XSL transformations, but its efficiency is questionable.

An issue that may surprise a new user is that Intalio server will confuse business processes if their namespaces are the same and if their pools have the same names. One could expect that it is enough to give a different name to different business processes. If several business processes with conflicting namespaces and conflicting pool names are deployed, only the one deployed last will work. The problem can be overcome by either using individual pool names or an individual namespace in each business process. Even though it can be considered a good design principle to use individual namespaces, Intalio Designer does not encourage to it; neither a warning is given when such namespace conflict occurs during deployment. Moreover, the examples and the tutorials provided by Intalio Community website typically use the default namespace suggested by Intalio Designer (”http://www.example.com/”).

There is also another reason why the names of diagram elements should be considered carefully. When a business process is built, the name of the WSDL generated for the business process is formed using the name of the diagram and its executable pool. For example, if a diagram is called ”Diagram” and its executable pool is called ”Process”, the name of the WSDL file will be ”Diagram-Process.wsdl”. It may not be the most descriptive one. Moreover, the operation that will be created in the service interface will receive the name of the first task in the business process. ”Read input” may be a descriptive name for a task, but it is a bad name for an operation. It may be difficult to find a name that is descriptive for both a task and an operation.

To improve the usability of Intalio BPMS and to reduce the time required to create business processes, there should be more feedback given at the design time (that is, by Intalio Designer). Currently, several problems only occur while trying to deploy a business process to an Intalio server or while running it. If Intalio Designer recognized the errors made by a user as soon as they have been done, time would be saved. There would be less need to test business processes under development after each new task added.

Despite all the problems, it can be said that using Intalio BPMS to model executable business processes saves time compared to programming languages (such as C++ or Java). Defining conditional execution or loops with BPMN may be slower than with a textual language. However, the possibility to define service interaction with drag and drop is quick compared to defining everything textually. An advantage is also that the BPMN diagram does not only define the flow but it can also be used for documentation as it is.

5.6 Client Development

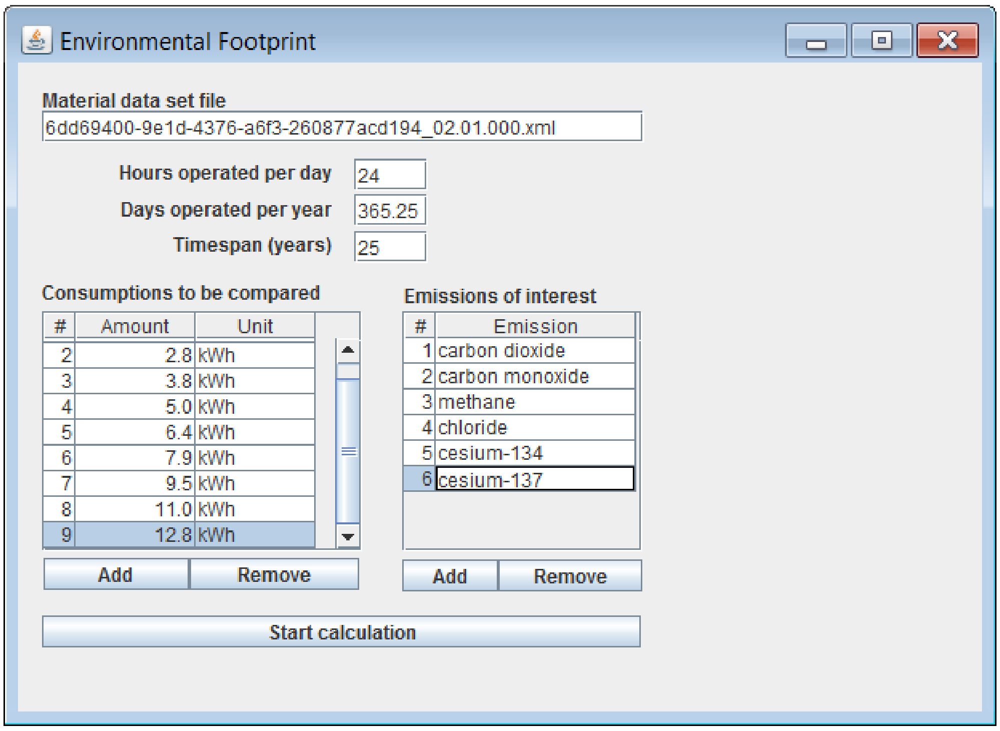
Different client solutions are possible for the footprint estimator business process. The only requirement is that the client can call the Web Service interface of *GetFootprints* business process. The most probable use would be to integrate the business process to a larger solution. The client could be, for instance, an enterprise information system or a business process created with Intalio BPMS. However, in this study, a simple standalone client is created. The programming language to be used is Java, and the necessary functionality to call the Web Service is created with JAX-WS (Java API for XML Web Services) library. With JAX-WS, exchanging XML messages with Web Services is easy. It offers the means to convert Java objects into XML contents and vice versa.

At first, it was attempted to create the client with NetBeans IDE. There were no problems using JAX-WS to generate the classes performing the Web Service interaction. However, when it was tried to compile the client, an error message about the version of JAX-WS occurred: a newer version of JAX-WS was required. JAX-WS was updated successfully, but trying to compile the client, another error message occurred. At this point, it was decided to try if another development environment, Eclipse, could offer a JAX-WS usage with fewer problems.

Client generation with Eclipse proved simpler than with NetBeans IDE. Even though JAX-WS was still used, less code was generated by Eclipse for the Web Service interaction. What is advantageous is that Eclipse does not require a web server software installed to create a Web Service client. In NetBeans IDE, Web Service clients are created inside web projects which require always a web server.

The original objective of creating a Java applet to be run in a web browser proved problematic. After the Web Service interaction code had been generated with JAX-WS, there were no difficulties developing the graphical user interface. The functionality of the applet was also correct when it was tried in Eclipse. However, problems arose while attempting to run the applet in an HTML page in a web browser. The client applet was exported as a JAR package from Eclipse, and an HTML page containing the applet was created. The applet appeared in the web browser as expected, but it received no response from the Web Service. After a few attempts, the logs of Intalio server were investigated. They indicated that there had been no request to the service at all, which referred to that the applet could not submit requests for some reason. Both Opera and Mozilla Firefox were tried as the browsers with the same result. As there were no problems running the applet in Eclipse, it was expected that the security features of the web browsers blocked the functionality of the applet. Creating a security signature for the JAR file did not help even though it resulted in a security warning in the browser every time the applet was started. Investigations to the source code indicated that the execution halted when an object of a class generated by JAX-WS was instantiated. However, as there was no solution at this point, it was decided to create a different client application.

A normal Java application was created instead of an applet. The source code of the applet was almost completely reused in the application. As a result, only minimal work was required. The application was exported as an executable JAR file, and it worked since the first attempt. The JAR file was not signed, but it did not cause any problems.



Environmental Footprint

Material data set file: 6dd69400-9e1d-4376-a6f3-260877acd194_02.01.000.xml

Hours operated per day: 24

Days operated per year: 365.25

Timespan (years): 25

Consumptions to be compared

#	Amount	Unit
2	2.8	kWh
3	3.8	kWh
4	5.0	kWh
5	6.4	kWh
6	7.9	kWh
7	9.5	kWh
8	11.0	kWh
9	12.8	kWh

Emissions of interest

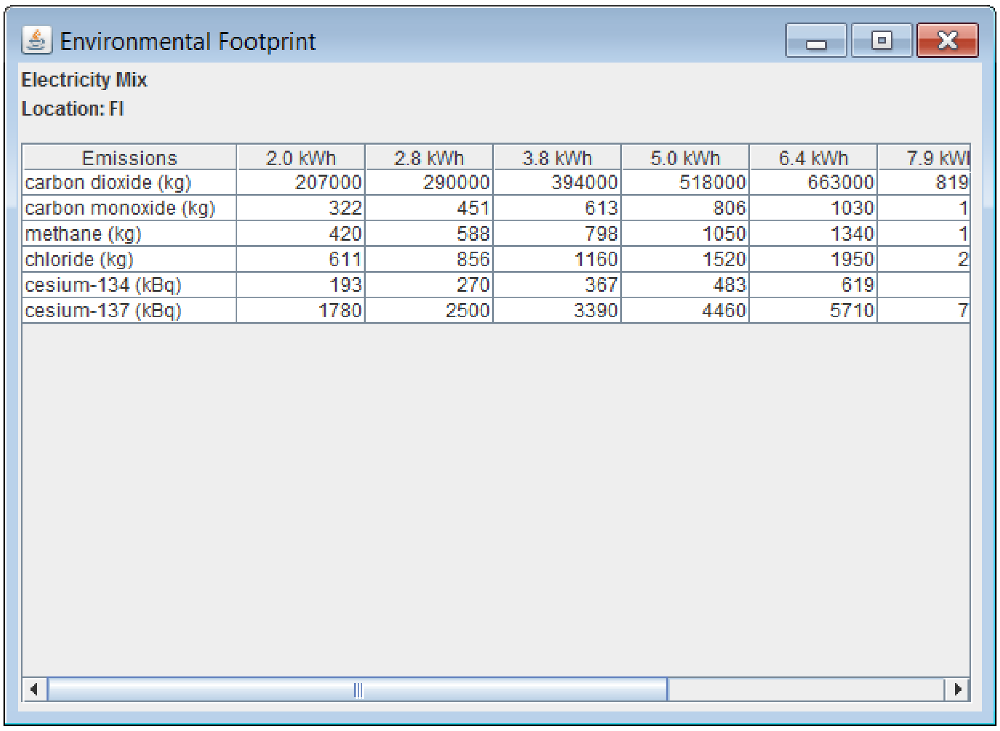
#	Emission
1	carbon dioxide
2	carbon monoxide
3	methane
4	chloride
5	cesium-134
6	cesium-137

Figure 5.13: The start view of the client.

The initial view of the client application is showed in figure 5.13. It is used to give the input information required for footprint estimation: the name of the relevant *process data set* file, operating duration information, the amounts of consumption to be compared and the emissions whose footprints shall be calculated. After the information has been given, the user submits the dialog.

When the footprints have been calculated, the results view (figure 5.14) is shown. It shows the description of the process as given in the *process data set* file and the location of the process. These fields together make it possible to check whether the information has been retrieved from the right *process data set* file. Under them, there is a table that contains the requested emissions as rows and the given consumptions as columns. Each cell shows the footprint of one emission when a specific amount of the resource is consumed.

Finally, the client worked as wanted, but there were two major challenges that were not overcome. First, it was decided to give up using NetBeans IDE and to use Eclipse instead. Then, running the client as an applet inside a web page did not succeed after which a normal application was implemented. These challenges show that it is anything but trivial to implement a JAX-WS client that operates inside a web page. However, there were no problems creating a normal Java application with Eclipse.



Electricity Mix
Location: FI

Emissions	2.0 kWh	2.8 kWh	3.8 kWh	5.0 kWh	6.4 kWh	7.9 kWl
carbon dioxide (kg)	207000	290000	394000	518000	663000	819
carbon monoxide (kg)	322	451	613	806	1030	1
methane (kg)	420	588	798	1050	1340	1
chloride (kg)	611	856	1160	1520	1950	2
cesium-134 (kBq)	193	270	367	483	619	
cesium-137 (kBq)	1780	2500	3390	4460	5710	7

Figure 5.14: The results view of the client.

5.7 Conclusions

In this chapter, a business process that estimates environmental footprints of devices was introduced. An environmental database is used to retrieve the emission data of production processes after which the reference amounts of different emissions is multiplied with the lifetime usage of a device. Several organizations have collected information about the inputs and the outputs of different production processes. ELCD, an environmental database that provides its data free of charge in XML format, was used in this study. All the business processes have been created with Intalio BPMS.

The architecture of the footprint estimator is a hierarchy of business processes. The environmental data being used is located on the downmost level of the hierarchy. In the first version of the architecture, most complexity was located on the top level. As the business processes on the bottom levels were simple, they had to be called repeatedly, which deteriorated overall performance. Then, the hierarchy was optimized by placing more complexity to the bottom levels, and a better performance was reached. Further optimizations are possible: for example, a data cache could be used.

Five business processes were created on different levels of the hierarchy. A lower number of business processes could have been possible if the structure of ELCD was simpler; the structure of ELCD requires three XML data sets to be retrieved to resolve the reference unit of an emission. Four of the business processes retrieve or process the reference data of ELCD. The topmost business process performs the multiplication of reference amounts with the lifetime consumption of the devices being observed.

Even though XML-over-HTTP data utilization is supposed to be easy, problems were encountered integrating ELCD to business processes. It was discovered that the ELCD XML data sets contain non-atomic fields, and some necessary fields have been declared optional. Moreover, ELCD server cannot be integrated to Intalio BPMS directly due to HTTP incompatibility issues. An HTTP wrapper service was used to overcome the problem.

XSLT and BPMN are essential technologies in the business processes. Despite XSLT can process complex XML data, it was noticed that it may not be an optimal technology for XML database processing. BPMN is both intuitive and expressive. However, modeling executable processes may require modeling of low-level actions which may decrease understandability.

Several problems were encountered creating business processes with Intalio BPMS. Business processes do not tolerate modifications well, and namespace collisions occur easily. Moreover, it is often difficult to find the cause of an error: several errors cannot be found at design time, and there are few possibilities to investigate service return values. However, it can be said that business process modeling with Intalio BPMS saves time compared to traditional textual programming.

Creating an example client for the business process in Java was problematic. After encountering errors with one development environment, another was chosen after which the client was compiled successfully. However, the client was never used as a Java applet inside a web page as planned because it was unable to connect the service while running in a web browser. Finally, the client was changed to a Java application after which it worked with no problems.

The business processes created in this chapter enable the comparison of the environmental footprints of different devices. Although the client developed in this study is a simple Java application, more complex applications could be developed. As the entire business process hierarchy has a single Web Service interface, integration is straightforward to any application. In the examples, only one resource (such as electricity) is observed. However, a client application could execute the business process several times for different resources consumed. Now that it has been indicated that it is possible to integrate ELCD to an executable business process, new business processes with new features could also be created.

6. SERVICE-ORIENTED CONDITION MONITORING

Another industrial SOA application created in this study is the condition monitoring business process. Condition monitoring is an essential function in modern industrial plants: the availability of up-to-date condition information reduces the probability of unexpected failures. It does not only reduce the number of unwanted stoppages but it may also improve the safety of employees and reduce the danger of polluting the environment. The service-oriented business process introduced in this chapter retrieves condition information from devices using service-oriented business processes and DPWS. A plant information model is also used. All the business processes are created using the same technologies as in the previous chapter.

6.1 Architecture

The idea of the condition monitoring business process is simple as shown in figure 6.1. First, the user of the business process starts the process. Then, the process queries for device condition information and finally shows it to the user. Figure 6.2 shows the architecture of the condition monitoring application. It consists of business processes (represented by circles), devices with DPWS support (represented by squares) and the plant model (represented by a rectangle). The roles of different services in the hierarchy are explained in the following paragraphs.

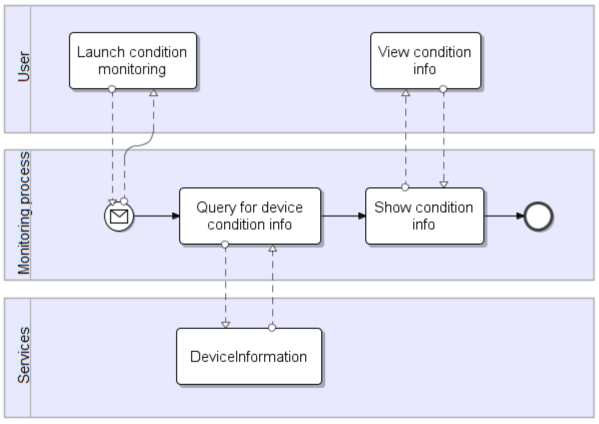


Figure 6.1: The condition monitoring business process on a high level.

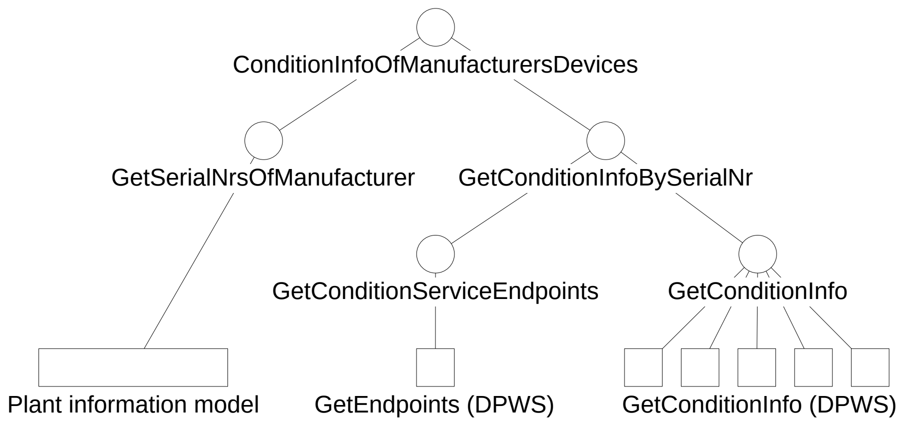

ConditionInfoOfManufacturersDevices
GetSerialNrsOfManufacturer
GetConditionInfoBySerialNr
GetConditionServiceEndpoints
GetConditionInfo
Plant information model
GetEndpoints (DPWS)
GetConditionInfo (DPWS)

Figure 6.2: The architecture of the condition monitoring business process.

The purpose of a plant model is to manage the data related to a factory plant. During a plant lifecycle, several parties produce or use plant information that can be in different human languages and different data formats. Two problems arise. On one hand, data conversions to different formats are required – sometimes they have to be performed even manually. On the other hand, it is typical that the plant data is obsolete as it has not been maintained to correspond the system reconfigurations made during the lifecycle of the plant. [66, pp. 1–2] Plant models intend to overcome this problem. Several standards have been developed in the area, and they deal with different aspects of plant models. The reason for having more than one standard is that different industry domains have different needs. In addition, there have also been different opinions about which is essential in a plant model specification. [84, p. 1] In this scenario, only a simple functionality is required from the plant model: it shall provide the required information to resolve the serial numbers of the devices of a given manufacturer. The plant model being used has an XML-over-HTTP interface, which makes its integration straightforward.

Two types of DPWS services are required for the architecture. The actual condition information is provided by **GetConditionInfo** DPWS services that are provided by devices. The endpoints of *GetConditionInfo* services are provided by **GetEndpoints** DPWS service. There are several reasons for using a separate service to provide service endpoints. As new devices may come and old devices may be replaced, it is expected that device endpoints are not known during the design of the condition monitoring business process. On the other hand, less manual work is required as there is no need to configure the *GetConditionInfo* service endpoints manually (there may be countless devices in the network).The only static endpoint in the network will be that of *GetEndpoints*. The disadvantage of this solution is that there is a single point of failure in the network: if *GetEndpoints* is offline, the condition information cannot be retrieved. However, to avoid such situations, there could be more devices providing the *GetEndpoints* service, bringing redundancy.

There are five business processes in the architecture. On the top of the hierarchy, **ConditionInfoOfManufacturersDevices** uses two other business processes. First, it retrieves the serial numbers of the devices delivered by a manufacturer after which the condition information of devices is retrieved using these serial numbers. The serial numbers are provided by **GetSerialNrsOfManufacturer** business process which calls the plant information model to gather the information. The actual condition information is provided by **GetConditionInfoBySerialNr** which uses two other business processes. **GetConditionServiceEndpoints** uses the *GetEndpoints* DPWS service to retrieve the endpoints of the services. After the endpoints are known, **GetConditionInfo** business process is called to retrieve the condition information from devices.

6.2 Implementation

6.2.1 Business Process Design

This subsection presents the design of the business processes in the condition monitoring business process hierarchy. The flows of the business processes are illustrated using BPMN diagrams: Their input and output XML schemata are presented as UML class diagrams in appendix B. In each BPMN diagram, an additional tree figure indicates the place of the related business process in the hierarchy. No attributes are used in the XML schemata; that is, all the "classes" and their members shall be interpreted as XML elements. However, if annotation "from xsd" is presented, it means that the related node is a text node whose type is an XML base type.

GetSerialNrsOfManufacturer business process retrieves the serial numbers of the devices of one manufacturer. Its XML schema for input and output data is shown in figure B.1. The flow of the business process is shown in figure 6.3. First, an XSL transformation is executed to assign the manufacturer name to a device search request after which the search is performed. For each matching device, the search function returns the related identifier used in the plant information model. Then, the identifiers are used in a loop construct that retrieves the information of each device. For each device data set, two XSL transformations are required: one to extract the serial number of a device and another to assign it to the return value of the business process.

The functionality of **GetConditionServiceEndpoints** business process is nothing more than that of *GetEndpoints* DPWS service; however, an easier integration with other business processes is provided. As the input, *GetConditionServiceEndpoints* takes a list of serial numbers of the devices whose condition service endpoints

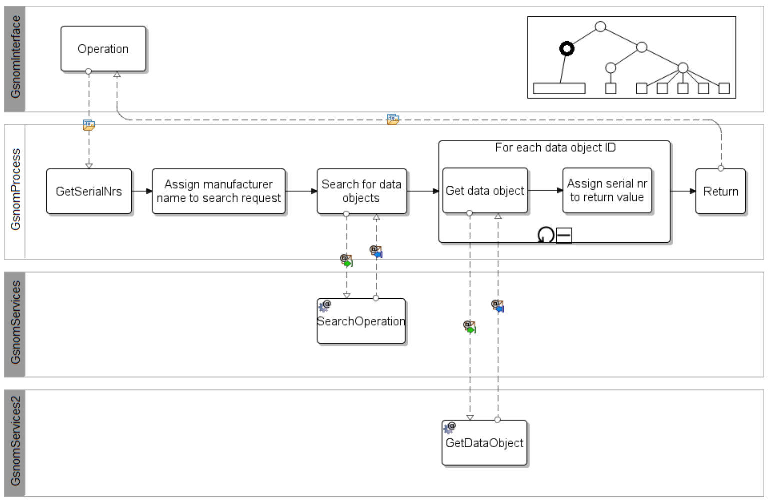


Figure 6.3: The BPMN diagram of the GetSerialNrsOfManufacturer business process.

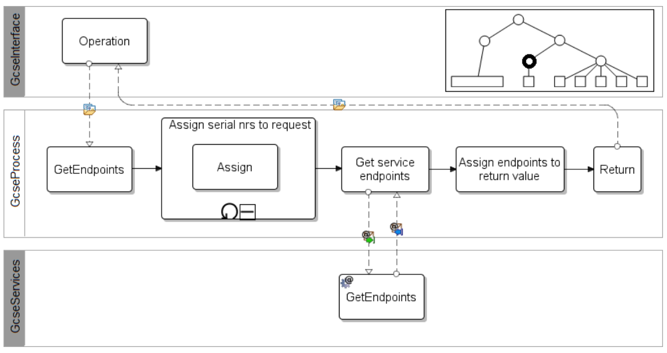


Figure 6.4: The BPMN diagram of the GetConditionServiceEndpoints business process.

shall be retrieved (figure B.2). As the output, it returns the endpoints of the condition services provided by those devices. If any device is unreachable, its serial number will be given in the list of unreachable devices. The flow of the business process is shown in figure 6.4. First, *GetConditionServiceEndpoints* assigns the serial numbers from its request to the request to be submitted to *GetEndpoints* DPWS service. Then, *GetEndpoints* is called after which the returned service endpoints are assigned to the return value of the business process using XSLT.

GetConditionInfo business process retrieves the condition information from given condition service endpoints. It reuses some parts of the XML schema of *GetConditionServiceEndpoints* as shown in figure B.3: in its input, it takes the similar endpoint list of DPWS condition services. In the input, it is also chosen whether only the condition information of those devices that have problems shall be returned[1]. In the output, it returns the condition information retrieved from given endpoints. The flow of the business process is shown in figure 6.5. For each

[1]Integer is used as the type of this element as problems were encountered using boolean values in Intalio BPMS.

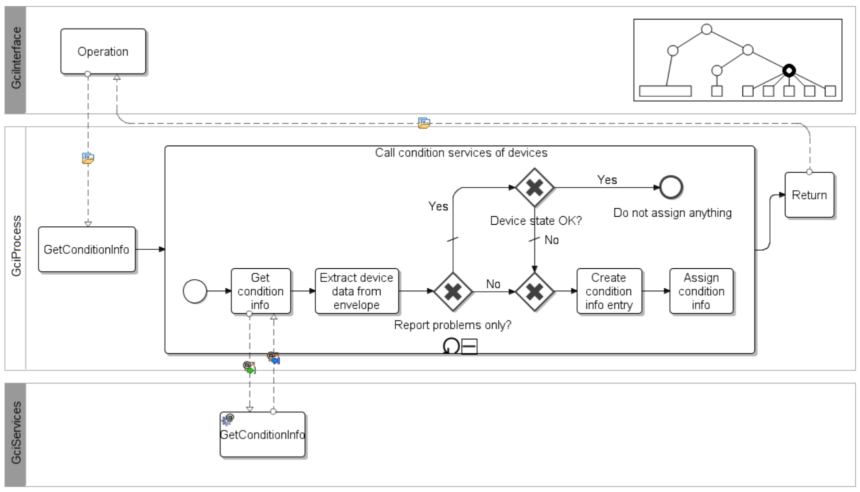


Figure 6.5: The BPMN diagram of the GetConditionInfo business process.

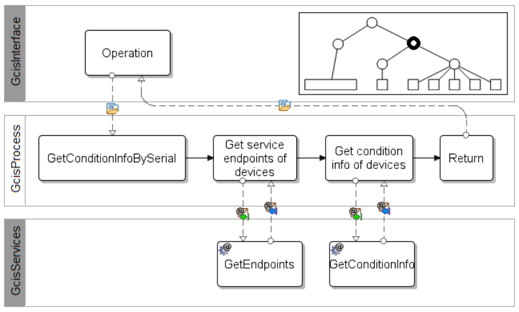


Figure 6.6: The BPMN diagram of the GetConditionInfoBySerialNr business process.

endpoint given, it retrieves the condition information. Device condition information is extracted from the device return value using XSLT. Depending on the request, either the condition information of all the devices or only those with problems are assigned to the return value. Before assigning device information to the return value, XSLT has to be used to add a new entry for it.

GetConditionInfoBySerialNr business process uses *GetConditionServiceEndpoints* and *GetConditionInfo* business processes to retrieve condition information according to given serial numbers. The XML schema of the business process is presented in figure B.4. All the elements inside the input and the output have been reused from the XML schemata of *GetConditionServiceEndpoints* and *GetConditionInfo*. The input contains a list of device serial numbers, and the output contains the condition information of devices as well as unreachable devices. Figure 6.6 shows the flow of the business process. First, *GetConditionServiceEndpoints* business process is called to retrieve condition service endpoints using device serial numbers. Then, the endpoints returned by it are used to retrieve device condition information by calling *GetConditionInfo* business process.

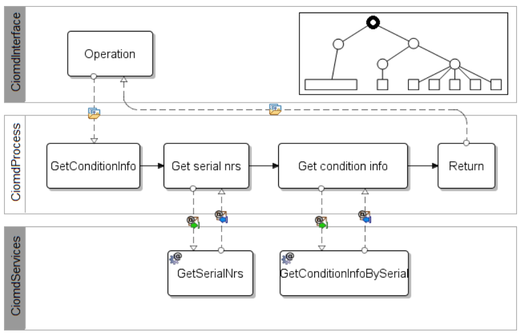


Figure 6.7: The BPMN diagram of the ConditionInfoOfManufacturersDevices business process.

Located on the top of the hierarchy, **ConditionInfoOfManufacturersDevices** business process uses *GetSerialNrsOfManufacturer* and *GetConditionInfoBySerialNr* business processes. The XML schema of *ConditionInfoOfManufacturersDevices* is shown in figure B.5. As the input, it takes a manufacturer name and the flag whether only devices with problems are returned; both of them are passed to the business processes it uses. The output is the return value of *GetConditionInfoBySerialNr* as it is. The flow of *ConditionInfoOfManufacturersDevices* is simple, as shown in figure 6.7: it contains only two business process calls. However, even this business process requires an XSL transformation: it is used to convert the serial numbers list of *GetSerialNrsOfManufacturer* so that it can be passed to *GetConditionInfoBySerialNr*. This is due to the design of the XML schemata. Even though the contents of these two serial number lists are identical (a list of strings), their XML schema types have been defined separately. As the result, a direct assignment is not possible. From this point of view, a better choice would have been to use the same type for both.

6.2.2 DPWS Network

The DPWS communications stack used to create the DPWS services in this study is JMEDS (Java Multi Edition DPWS Stack) provided by WS4D (Web Service for Devices) [94]. It was chosen after its evaluation in [52] showed its suitability for creating DPWS applications.

Figure 6.8 shows the structure of the DPWS device network. The master device shall always be online and it shall preserve its host name and port. It searches for DPWS devices with the *GetConditionInfo* service periodically. When such devices are found, their endpoint information is saved and a subscription is made to receive information about device state changes. If a device leaves the network, the master device receives a notification of it and removes the endpoint information of the

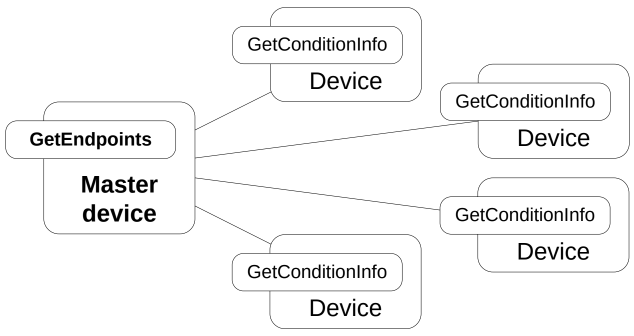


Figure 6.8: The structure of the DPWS device network.

device. The endpoint information of a device can be retrieved from the master device by the serial number of the device[2].

6.2.3 Integrating DPWS to Business Processes

Even though JMEDS generates a WSDL document for its services automatically, their integration to business processes is not straightforward. No endpoint definitions are provided in the WSDL documents. As there is no way to configure endpoints outside a WSDL document in Intalio BPMS, they should be added manually to WSDL files. Another limitation is that the endpoints defined in WSDL files are static. If there are several DPWS devices offering the same service, each of them requires its own WSDL definition as endpoints are defined static in WSDL. Due to these limitations, another way than WSDL import has to be used to integrate DPWS devices to Intalio BPMS business processes.

DPWS uses SOAP-over-HTTP as the communication method, and SOAP envelopes are XML documents. Thus, an alternative way to communicate with a DPWS service is to treat it as an XML-over-HTTP service. Such a binding can be defined in Intalio BPMS by creating a REST connector. Then, the type of the XML content to be submitted and received is set as the SOAP envelope. As the XML schema of SOAP envelope does not define what it shall contain, no data assignments can be performed directly in Intalio BPMS. However, the default contents of the envelope to be submitted can be edited manually. Then, the return value can be processed using an XSL transformation.

The solution looks simple, but challenges were encountered implementing it. Experiments showed that the HTTP requests generated by Intalio server are not compatible with JMEDS. Thus, it was decided to use the same HTTP wrapper that was used as in the life cycle simulator business process. Using the wrapper, HTTP GET

[2]Serial numbers are assumed unique regardless of the manufacturer of the device even though this may not be the case with real devices. However, the master device could be developed further to additionally use manufacturer names in identification.

```
POST /ConditionService HTTP/1.1
User-Agent: AOT Http Wrapper
Host: 130.230.xxx.xxx:50012
Content-Length: 475
Content-Type: application/xml; charset=ISO-8859-1

POST /ConditionService/ HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: application/soap+xml;charset=UTF-8
User-Agent: Jakarta Commons-HttpClient/3.1
Host: 130.230.xxx.xxx:50012
Content-Length: 475
```

Code 6.1: The HTTP POST request headers sent by the HTTP wrapper (first) and soapUI (second).

requests to a DPWS device worked as expected[3]. However, when trying to invoke a DPWS service with a SOAP message using HTTP POST method, problems arose even though the same request worked with no problems when executed by soapUI. A protocol analyzer was taken into use to compare the request headers sent by the HTTP wrapper and soapUI (code 6.1).

There were a few differences between the HTTP requests of HTTP wrapper and soapUI, and content type definition seemed the most probable cause of the problem. The existing functionality of the wrapper could not be modified because it would have broken the functionality offered to current consumers; thus, a new resource was added to the wrapper. Both the content types to be consumed and produced by the wrapper resource and the content type the wrapper would use requesting the ”wrapped” resource were set to ”application/soap+xml”. Moreover, character encoding was set to UTF-8 instead of ISO 8859-1. After the new wrapper resource had been implemented, the wrapper was deployed to Apache Tomcat web server.

Experiments made on the new wrapper resource did not have the desired result. Tomcat refused from executing the request: it complained that the input format was not appropriate. It referred to that even though the HTTP wrapper was set to consume SOAP messages, the Java library (JAX-RS[4]) used to create the wrapper does not allow it. It is a reasonable restriction in the sense that a service with a SOAP-over-HTTP interface is actually an operation-oriented Web Service. Such services are not supposed to be created with JAX-RS but with JAX-WS or another SOAP Web Service library.

However, the problem was overcome easily. The content type consumed and produced by the resource was set to ”application/xml”, and the content type it

[3]GET method is used, for instance, to retrieve the WSDL document of a service provided by a device.

[4]Java API for RESTful Web Services

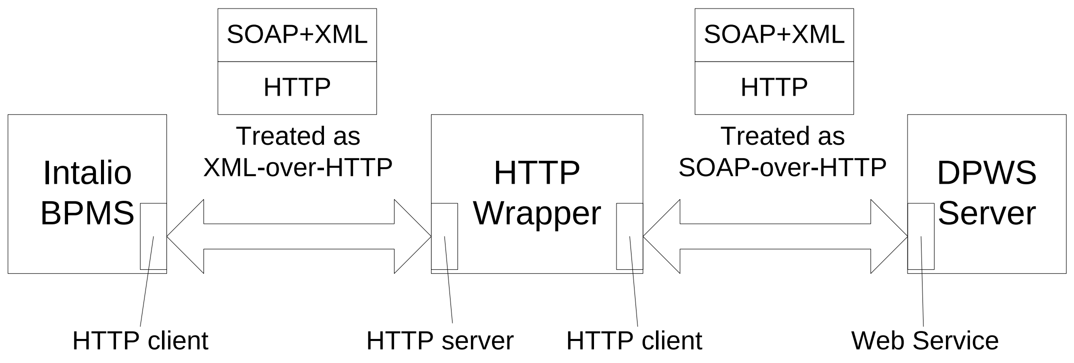


Figure 6.9: Requests are treated as any XML content between Intalio BPMS and the HTTP wrapper even though the messages contain actually SOAP.

would use in requests was kept as ”application/soap+xml”. It had the desired effect: requests to DPWS services through the wrapper succeeded. At this point, it was unclear whether character encoding affected the result. It was set back to ISO 8859-1, and there was no effect on the result of the request. However, it was decided to use UTF-8 as it provides a wider support for different characters. Figure 6.9 illustrates the content types used in the scenario: HTTP wrapper consumes and produces ”application/xml”, but in requests, it uses ”application/soap+xml”.

6.3 Conclusions

This chapter introduced a service-oriented business process that retrieves condition information from devices. DPWS was used to make devices capable of Web Service communication, and a plant information model provided additional device information. The DPWS network was implemented so that there is one device that observes other devices and provides information about their endpoints. This way, the network can be changed dynamically with no need for manual configuration.

Five business processes were created to collect and process device information. They form a hierarchy so that those on the top accomplish complex tasks by using the simple ones on the bottom levels. The downmost level of the hierarchy is formed by the information used by the business processes – that is, the DPWS devices and the plant information model.

The integration of DPWS services to business processes was problematic. The WSDL files generated by the DPWS communications stack do not contain any service endpoints. That is, manual endpoint configuration is required, but Intalio BPMS does not support it. The solution was to treat DPWS services as XML-over-HTTP services that have SOAP envelope as the XML schema type used in communication. The information returned by DPWS was extracted from envelopes using XSLT. Another problem was that the HTTP requests sent by Intalio BPMS were not compatible with the DPWS stack. The problem was solved by using a HTTP wrapper.

Despite the problems encountered integrating DPWS, it can be said that the use of Web Services and XML facilitates integration. To make integration even easier, Intalio BPMS could have better possibilities to configure service endpoints and HTTP requests manually. The DPWS stack could also accept a wider range of different HTTP requests.

7. SUMMARY

This chapter has two sections. First, the results of the work are summarized. Then, future work is discussed.

7.1 Results

Several advantages could be reached if all industrial automation devices were capable of communicating in a service-oriented way. It would make system assembly and configuration more flexible, which would save time and money. However, no generic solution is currently available for two reasons. First, current service-oriented technologies require too much computational resources to be run in current devices. Second, no technology can provide both service-orientation and hard real-time support. Research has been made to overcome these problems, but especially a strict real-time capable service-oriented architecture cannot yet be implemented with current technologies.

However, several functions of operations and maintenance can be implemented using the current service-oriented technologies. As a demonstration, two service-oriented business processes were created in this study. The first business process estimates the environmental footprints of devices whereas the second one retrieves condition monitoring information from devices. The technologies used in communications are Web Services and HTTP. All the data is represented as XML. To model business processes, a standard graphical notation called BPMN is used. The modeling environment used in the study converts BPMN diagrams to WS-BPEL which is an executable business process description language.

The environmental footprint estimator business process demonstrates the integration of a public environmental database to service-oriented business processes. The database provides its data in XML format which is a standardized way to present structured data. Despite the complexity of the database, data processing is straightforward as XML is easy to integrate to business processes. The business process modeling environment that was used has the means to process complex XML documents. HTTP incompatibility problems were encountered between the business process execution environment and the database servers. However, they were solved by using a HTTP wrapper to manipulate requests. The business process modeling tool suffers also from technical problems. Despite them, it was ascertained that business process modeling with the chosen method saves time. Using BPMN to model the interaction between business process participants graphically is both intuitive and quick. Using a textual programming language for the same purpose would require more work. It was also noticed that it is essential to minimize the number of any service requests in architectural design. As a result of the optimizations that reduced the number of service calls, the performance of the footprint estimator improved significantly.

The condition monitoring business process uses similar technologies as the footprint estimator, but the data is retrieved from industrial devices. To enable integration using Web Services, the devices use a technology profile called DPWS. DPWS lowers the resources consumption of Web Services, improving their suitability for devices. Despite DPWS, integration to business processes cannot be performed directly due to HTTP incompatibility problems. However, the problems can be overcome by using a HTTP wrapper. Another issue was that the interface definitions provided by the DPWS framework used in the study do not define service endpoints. It hindered the use of the interface definitions in business processes. The integration was performed by defining endpoints manually. In addition, more work was required on message processing as the automatic functions of the business process modeling environment could not be used.

7.2 Future Work

More research could be performed related to the environmental footprint estimator. The retrieval of data from the environmental database is slow as it has to be performed over WWW. Time would be saved if the business processes were optimized even more to minimize the number of redundant documents retrieved. There could also be a local cache of documents. The cache could be such that all the contents of the database would be downloaded into it regularly. Another option would be an ”on-request” type cache. Whenever a new document was requested, it would be stored in the cache. If the document being requested existed in the cache already, it could be returned from there. To keep the cache up-to-date, the documents could have a maximal age after which they would be refreshed.

The condition monitoring business process could also be developed further. This far, only small networks have been tested. It would be important to receive information about network behavior when there are hundreds or thousands of devices. It would also be interesting to investigate the performance of DPWS on different devices.

Moreover, both business processes could be integrated to enterprise information systems. The environmental footprint estimator would suit well to decision support systems. The condition monitoring business process would be useful as a part of supervisory systems. Furthermore, the business processes could be used together as a part of an OEE (Overall Equipment Effectiveness) system. Whenever there is a need for device replacement, the condition monitoring business process indicates it. Then, while choosing a new device to replace the old one, the footprint estimator business process provides data for device comparison.

The business processes could also be integrated to a business collaboration system. Whenever the condition monitoring business process indicates a technical problem, the collaboration system is used to submit a work request to the maintenance service provider. Similarly, work requests are sent when a new device has been chosen using the results given by the footprint estimator. At least a device supplier and a device installer are needed to perform the installation. Such a collaboration system would facilitate the communication between organizations, reducing the need for manual work.

REFERENCES

[1] ActiveVOS [WWW]. [Referenced 14.5.2010]. Available: http://activevos.com/

[2] Aggarwal, A., Amend, M., Astier, S., Barros, A., Bartel, R., Benitez, M., Bock, C., Brown, G., Brunt, J., Bulles, J., Chapman, M., Cummins, F., Day, R., Elaasar, M., Frankel, D., Gagné, D., Hall, J., Hille-Doering, R., Ings, D., Irassar, P., Kieselbach, O., Kloppmann, M., Koehler, J., Kraft, F.M., van Lessen, T., Leymann, F., Lonjon, A., Malhotra, S., Menge, F., Mischkinsky, J., Moberg, D., Moffat, A., Mueller, R., Nijssen, S., Ploesser, K., Rivett, P., Rowley, M., Ruecker, B., Rutt, T., Samoojh, S., Shapiro, R., Saxena, V., Schanel, S., Scheithauer, A., Silver, B., Srinivasan, M., Toulme, A., Trickovic, I., Voelzer, H., Weber, F., Westerinen, A. & White, S.A. Business Process Model and Notation (BPMN) Version 2.0 Beta 2 [WWW]. Object Management Group. 5.6.2010 [Referenced 31.8.2010]. Available: http://www.omg.org/spec/BPMN/2.0/Beta2/PDF/

[3] Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P. & Yiu, A. Web Services Business Process Execution Language Version 2.0 [WWW]. OASIS. 11.4.2007 [Referenced 12.5.2010]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[4] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano J., Tuecke S. & Xu, M. Web Services Agreement Specification (WS-Agreement) [WWW]. Open Grid Forum. 14.3.2007 [Referenced 18.9.2010]. Available: http://www.ogf.org/documents/GFD.107.pdf

[5] Bangemann, T., Diedrich, C., Colombo, A.W. & Karnouskos, S. SOCRADES – Service Oriented Architecture in der Automatisierungstechnik [WWW]. 3.6.2008 [Referenced 20.5.2010]. Available:
http://www.socrades.eu/Documents/objects/file1259600136.4

[6] Barker, A., Walton, C.D. & Robertson, D. Choreographing Web Services. IEEE Transactions on Service Computing [electronic journal]. 2(2009)2, pp. 152–166. 17.7.2009 [Referenced 30.8.2010]. Available:
http://ieeexplore.ieee.org/

[7] Barreto, C., Bullard, V., Erl, T., Evdemon, J., Jordan, D., Kand, K., König, D., Moser, S., Stout, R., Ten-Hove, R., Trickovic, I., van der Rijn, D. & Yiu, A. Web Services Business Process Execution Language Version 2.0 Primer

[WWW]. OASIS. 9.5.2007 [Referenced 14.5.2010]. Available: http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm

[8] Barry, D.K. Web Services and Service-Oriented Architectures. San Francisco, California, USA 2003, Morgan Kaufmann. 245 p.

[9] Bean, J. SOA and Web Services Interface Design Principles, Techniques and Standards. Burlington, Massachusetts, USA 2009, Morgan Kaufmann. 384 p.

[10] Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J. & Siméon, J. XML Path Language (XPath) 2.0 [WWW]. World Wide Web Consortium. 23.1.2007 [Referenced 11.11.2010]. Available: http://www.w3.org/TR/2007/REC-xpath20-20070123/

[11] Biron, P.V. & Malhotra, A. XML Schema Part 2: Datatypes Second Edition [WWW]. World Wide Web Consortium. 2.5.2001, 28.10.2004 [Referenced 7.5.2010]. Available: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[12] BizAgi BPMS [WWW]. [Referenced 31.8.2010]. Available: http://www.bizagi.com/

[13] Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J. & Siméon, J. XQuery 1.0: An XML Query Language [WWW]. World Wide Web Consortium. 23.1.2007 [Referenced 8.11.2010]. Available: http://www.w3.org/TR/2007/REC-xquery-20070123/

[14] Bohn, H., Bobek, A. & Golatowski, F. SIRENA – Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICN/ICONS/MCL 2006), Morne, Mauritius 23.–29.4.2006. 2006, IEEE. P. 43.

[15] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. Web Services Architecture [WWW]. World Wide Web Consortium. 11.2.2004 [Referenced 7.5.2010]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[16] Box, D., Cabrera, L.F., Critchley, C., Curbera, F., Ferguson, D., Graham, S., Hull, D., Kakivaya, G., Lewis, A., Lovering, B., Niblett, P., Orchard, D., Samdarshi, S., Schlimmer, J., Sedukhin, I., Shewchuk, J., Weerawarana,

S. & Wortendyke, D. Web Services Eventing (WS-Eventing) [WWW]. World Wide Web Consortium. 15.3.2006 [Referenced: 10.5.2010]. Available: http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/

[17] Bray, T., Hollander, D., Layman, A., Tobin, R. & Thompson, H.S. 2009. Namespaces in XML 1.0 (Third Edition) [WWW]. World Wide Web Consortium. 14.1.1999, 8.12.2009 [Referenced 5.5.2010]. Available: http://www.w3.org/TR/2009/REC-xml-names-20091208/

[18] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F. & Cowan, J. Extensible Markup Language (XML) 1.1 (Second Edition) [WWW]. World Wide Web Consortium. 16.8.2006, 29.9.2006 [Referenced 4.5.2010]. Available: http://www.w3.org/TR/2006/REC-xml11-20060816/

[19] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., & Yergeau, F. Extensible Markup Language (XML) 1.0 (Fifth Edition) [WWW]. World Wide Web Consortium. 10.2.1998, 26.11.2008 [Referenced 4.5.2010]. Available: http://www.w3.org/TR/2008/REC-xml-20081126/

[20] Business Process Model and Notation (BPMN) [WWW]. Object Management Group. [Referenced 17.5.2010]. Available: http://www.omg.org/spec/BPMN/

[21] Chinnici, R., Haas, H., Lewis, A.A., Moreau, J.J., Orchard, D. & Weerawarana, S. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts [WWW]. World Wide Web Consortium. 26.6.2007 [Referenced 5.5.2010]. Available: http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/

[22] Chinnici, R., Moreau, J.J., Ryman, A. & Weerawarana, S. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language [WWW]. World Wide Web Consortium. 26.6.2007 [Referenced 5.5.2010]. Available: http://www.w3.org/TR/2007/REC-wsdl20-20070626/

[23] Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. Web Services Description Language (WSDL) 1.1 [WWW]. World Wide Web Consortium. 15.3.2001 [Referenced 15.5.2010]. Available: http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[24] Cohen, F. FastSOA. San Francisco, California, USA 2006, Morgan Kaufmann. 296 p.

[25] Cucinotta, T., Mancina, A., Anastasi, G.F., Lipari, G., Mangeruca, L., Checcozzo, R. & Rusinà, F. A Real-Time Service-Oriented Architecture for Industrial Automation. IEEE Transactions on Industrial Informatics [electronic

journal]. 5(2009)3, pp. 267–277. 7.8.2009 [Referenced 20.5.2010]. Available: http://ieeexplore.ieee.org/

[26] Decker, G. & Barros, A. Interaction Modeling Using BPMN [WWW]. CiteSeerX. [Referenced 9.11.2010]. Available:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.8751&rep=rep1&type=pdf

[27] Dhesiaseelan, A. What's new in WSDL 2.0 [WWW]. O'Reilly. 20.5.2004 [Referenced 15.5.2010]. Available:
http://www.xml.com/pub/a/ws/2004/05/19/wsdl2.html

[28] EAA: European Aluminium Association. [Referenced 20.9.2010]. Available: http://www.eaa.net/

[29] EcoInvent: Swiss Centre for Life Cycle Inventories. [Referenced 20.9.2010]. Available: http://www.ecoinvent.org/

[30] ELCD core database version II. [Referenced 18.10.2010]. Available:
http://lca.jrc.ec.europa.eu/lcainfohub/datasetArea.vm

[31] Erl, T. Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River, New Jersey, USA 2005, Prentice Hall. 760 p.

[32] Erl, T. SOA Glossary [WWW]. SOA Systems Inc. [Referenced 24.4.2010]. Available: http://www.soaglossary.com/

[33] Erl, T. SOA Principles [WWW]. SOA Systems Inc. [Referenced 24.4.2010]. Available: http://www.soaprinciples.com/

[34] Erl, T. SOA: Principles of Service Design. Upper Saddle River, New Jersey, USA 2008, Prentice Hall. 573 p.

[35] FEFCO: European Federation of Corrugated Board Manufacturers. [Referenced 20.9.2010]. Available: http://www.fefco.org/

[36] Feldhorst, S., Libert, S., ten Hompel, M. & Krumm, H. Integration of a Legacy Automation System into a SOA for Devices. IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2009), Palma de Mallorca, Spain, 22.–25.7.2009. 2009, IEEE. Pp. 1–8.

[37] Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures. Dissertation. Irvine, California, USA 2000. University of California, Information and Computer Science. 162 p. Available:
http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm.

[38] Fujitsu Interstage [WWW]. [Referenced: 18.5.2010]. Available: http://www.fujitsu.com/global/services/software/interstage/

[39] GaBi Software. [Referenced 20.9.2010]. Available: http://www.gabi-software.com/

[40] Gang, C., Ye, D. & Che, R. Developing Trend of Industrial Fieldbus Control System. 8th International Conference on Electronic Measurement and Instruments 2007 (ICEMI '07), Xi'an, China 16.-18.8.2007. 2007, IEEE. Pp. 1765–1768.

[41] Garcés-Erice, L. Building an Enterprise Service Bus for Real-Time SOA: A Messaging Middleware Stack. 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009), Seattle, Washington, USA 20–24.7.2009. 2009, IEEE. Pp. 79–84.

[42] Goth, G. Critics Say Web Services Need a REST. IEEE Distributed Systems Online [elecronic journal]. 5(2004)12, pp. 1–7. 17.1.2005 [Referenced 3.5.2010]. Available: http://ieeexplore.ieee.org/

[43] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H.F., Karmarkar, A. & Lafon, Y. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) [WWW]. World Wide Web Consortium. 24.6.2003, 27.4.2007 [Referenced 3.5.2010]. Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[44] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H.F., Karmarkar, A. & Lafon, Y. SOAP Version 1.2 Part 2: Adjuncts (Second Edition) [WWW]. World Wide Web Consortium. 24.6.2003, 27.4.2007 [Referenced 3.5.2010]. Available: http://www.w3.org/TR/2007/REC-soap12-part2-20070427/

[45] Hadley, M. Web Application Description Language. World Wide Web Consortium. 31.8.2009 [Referenced 1.9.2010]. Available: http://www.w3.org/Submission/2009/SUBM-wadl-20090831/

[46] High, R., Kinder, S. & Graham, S. IBM's SOA Foundation: An Architectural Introduction and Overview [WWW]. IBM. 15.12.2005 [Referenced 18.9.2010]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf

[47] ILCD Handbook flyer [WWW]. European Commission Joint Research Center (JRC). [Referenced 18.10.2010]. Available: http://lct.jrc.ec.europa.eu/pdf-directory/ILCDHandbook.pdf

[48] Intalio BPMS [WWW]. [Referenced 16.8.2010]. Available: http://www.intalio.com/bpms

[49] Introduction to UDDI: Important Features and Functional Concepts [WWW]. OASIS. 2004 [Referenced 6.5.2010]. Available: http://uddi.xml.org/files/uddi-tech-wp.pdf

[50] Iwasa, K., Durand, J., Rutt, T., Peel, M., Kunisetty, S. & Bunting, D. Web Services Reliable Messaging TC WS-Reliability 1.1 [WWW]. OASIS. 15.11.2004 [Referenced 8.5.2010]. Available: http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf

[51] Jammes, F. & Smit, H. Service-Oriented Paradigms in Industrial Automation. IEEE Transactions on Industrial Informatics [electronic journal]. 1(2005)1, pp. 62–70. 4.4.2005 [Referenced 19.5.2010]. Available: http://ieeexplore.ieee.org/

[52] Kannisto, P. DPWS in Industrial Automation: Applications and Evaluation. Tampere, Finland 9.12.2010. Project work for a university course. 28 p.

[53] Kay, M. XSL Transformations (XSLT) Version 2.0 [WWW]. World Wide Web Consortium. 23.1.2007 [Referenced 28.10.2010]. Available: http://www.w3.org/TR/2007/REC-xslt20-20070123/

[54] Komoda, N. Service Oriented Architecture (SOA) in Industrial Systems. IEEE International Conference on Industrial Informatics (INDIN 2006), Singapore 16.–18.8.2006. 2006, IEEE. Pp. 1–5.

[55] Kuikka, S. ACI-32040 Automaation ohjelmistokomponentit ja sovelluspalvelut: opetusmoniste kevät 2010. 15.2.2010, Tampere University of Technology. Handout for a university course. 206 p.

[56] Laskey, K., Estefan, J.A., McCabe, F.G. & Thornton, D. Reference Architecture Foundation for Service Oriented Architecture Version 1.0 [WWW]. OASIS. 14.10.2009 [Referenced 24.4.2010]. Available: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf

[57] Lawrence, K., Kaler, C., Nadalin, A., Monzillo, R. & Hallam-Baker, P. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) [WWW]. OASIS. 1.2.2006 [Referenced 8.5.2010]. Available:

http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[58] Life Cycle Thinking: Tools for the Assessment [WWW]. European Commission, Joint Research Centre. [Referenced 26.10.2010]. Available: http://lct.jrc.ec.europa.eu/assessment/tools

[59] Linthicum, D.S. The ROI of Your SOA [WWW]. ebizQ. 10.7.2005 [Referenced 1.5.2010]. Available: http://www.ebizq.net/topics/soa/features/6092.html

[60] MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F. & Metz, R. Reference Model for Service Oriented Architecture 1.0 [WWW]. OASIS. 12.10.2006 [Referenced 24.4.2010]. Available: http://docs.oasis-open.org/soa-rm/v1.0

[61] Maguire, J. Does SOA Have an ROI? [WWW]. Datamation. 14.3.2007 [Referenced 1.5.2010]. Available: http://itmanagement.earthweb.com/erp/article.php/3665266/Does-SOA-Have-an-ROI.htm

[62] Mintchev, A. Interoperability among Service Registry Implementations: Is UDDI Standard Enough? IEEE International Conference on Web Services (ICWS 2008), Beijing, China 23.–26.7.2008. 2008, IEEE. Pp. 724–731.

[63] Mitra, N. & Lafon, Y. SOAP Version 1.2 Part 0: Primer (Second Edition) [WWW]. World Wide Web Consortium. 27.4.2007 [Referenced 3.5.2010]. Available: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[64] zur Muehlen, M. BPM, Web Services, and Standardization [WWW]. Workflow Research. 2004 [Referenced 30.8.2010]. Available: http://www.workflow-research.de/Tutorials/AMCIS2004/MIZU-BPM-AMCIS2004-Section4.pdf

[65] zur Muehlen, M., Nickerson, J.V. & Swenson, K.D. Developing web services choreography standards – the case of REST vs. SOAP. Decision Support Systems [electronic journal]. 40(2005)1, pp. 9–29. [Referenced: 11.5.2010]. Available: http://www.sciencedirect.com/

[66] Mun, D., Lee, S., Kim, B. & Han, S. ISO 15926-based data repository and its web services for sharing lifecycle data of process plants. 2009 International Conference on Product Lifecycle Management (PLM09), Bath, United Kingdom, 6.–8.7. 2009. iCAD laboratory.

[67] Nitzsche, J., van Lessen, T. & Leymann, F. WSDL 2.0 Message Exchange Patterns: Limitations and Opportunities. Third International Conference on

Internet and Web Applications and Services (ICIW 2008), Athens, Greece 8.–13.6.2008. 2008, IEEE. Pp. 168–173.

[68] Nixon, T., Regnier, A., Driscoll, D. & Mensch, A. Devices Profile for Web Services Version 1.1 [WWW]. OASIS. 1.7.2009 [Referenced 10.6.2010]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html

[69] Nixon, T., Regnier, A., Modi, V. & Kemp, D. Web Services Dynamic Discovery (WS-Discovery) Version 1.1 [WWW]. OASIS. 1.7.2009 [Referenced 8.5.2010]. Available:
http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html

[70] Ogbuji, U. UDDI 3.0? Who really cares? [WWW]. O'Reilly. 3.2.2005 [Referenced 6.5.2010]. Available:
http://www.oreillynet.com/onlamp/blog/2005/02/uddi_30_who_really_cares.html

[71] OPC Foundation [WWW]. [Referenced 18.11.2010]. Available:
http://www.opcfoundation.org/

[72] OPC Unified Architecture Specification: Part 1: Overview and Concepts release 1.01 [WWW]. OPC Foundation. 5.2.2009 [Referenced 18.11.2010]. Available: http://www.opcfoundation.org/

[73] OPC Unified Architecture Specification: Part 6: Mappings release 1.00 [WWW]. OPC Foundation. 6.2.2009 [Referenced 18.11.2010]. Available:
http://www.opcfoundation.org/

[74] Panahi, M., Nie, W. & Lin, K.J. A Framework for Real-Time Service-Oriented Architecture. IEEE Conference on Commerce and Enterprise Computing (CEC 2009), Vienna, Austria 20–23.7.2009. 2009, IEEE. Pp. 460–467.

[75] Panahi, M., Nie, W. & Lin, K.J. The Design and Implementation of Service Reservations in Real-Time SOA. IEEE International Conference on e-Business Engineering (ICEBE 2009), Macau 21–23.10.2009. 2009, IEEE. Pp. 129–136.

[76] PlasticsEurope: The portal of EU's plastics industry. [Referenced 20.9.2010]. Available: http://www.plasticseurope.org/

[77] Poulin, J. & Himler, A. The ROI of SOA Based on Traditional Component Reuse [WWW]. LogicLibrary. 2006 [Referenced 1.5.2010]. Available: http://www.logiclibrary.com/pdf/wp/ROI_of_SOA.pdf

[78] Pöhlsen, S. & Werner, C. Robust Web Service Discovery in Large Networks. IEEE International Conference on Services Computing (SCC 2008), Honolulu, Hawaii, USA 7.–11.7.2008. 2008, IEEE. Pp. 521–524.

[79] Rebitzer, G., Ekvall, T., Frischknecht, R., Hunkeler, D., Norris, G., Rydberg, T., Schmidt, W.-P., Suh, S., Weidema, B.P. & Pennington, D.W. Life cycle assessment. Part 1: Framework, goal and scope definition, inventory analysis, and applications. Environment International [WWW]. 30(2004)5, pp. 701–720. [Referenced 27.8.2010]. Available: http://www.sciencedirect.com/

[80] RFC 2616. Hypertext Transfer Protocol – HTTP/1.1. 1999, The Internet Society. 176 p.

[81] Sadtler, C., Cotignola, D., Crabtree, B. & Michel, P. Patterns: Broker Interactions for Intra- and Inter-enterprise. 2004, IBM Redbooks. 292 p.

[82] Salmenperä, M. & Salonen, M. Architecture and initial performance indicators of new OPCUA automation protocol stack Java implementation. International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE 09), 4.–12.12.2009. 2010. Paper 21 in IETA subconference.

[83] Service-Oriented Device and Delivery [WWW]. 2006 [Referenced 10.6.2010]. Available: http://www.itea2.org/public/project_leaflets/SODA_profile_oct-06.pdf

[84] Siltanen, P. & Pärnänen, A. Information modeling for process industry: Comparison of integration standards [WWW]. Version 1.0. VTT. 29.11.2005 [Referenced 24.11.2010]. Available: http://pim.vtt.fi/semill/docs/2.pdf

[85] Silver, B. SAP NetWeaver BPM White Paper. SAP. 2009 [Referenced 1.9.2010]. Available:
http://download.sap.com/download.epd?context=DB579E1ADF205511BE1
A8E056E065794EBCFE135F6FBC70B0970594C94F47D21790A654360685C0
F532F71DF142B3EE9C5CD293A82450BF7

[86] SOA Tools BPMN Modeler [WWW]. [Referenced 20.8.2010]. Available:
http://www.eclipse.org/bpmn/

[87] SOCRADES [WWW]. [Referenced 11.6.2010]. Available:
http://www.socrades.eu/

[88] SODA [WWW]. 2008 [Referenced 10.6.2010]. Available:
http://www.soda-itea.org/

[89] Stollberg, M. & Fensel, D. Semantics for Service-Oriented Architectures. In: Griffiths, N. & Chao, K.M. Agent-Based Service-Oriented Computing. London, Great Britain 2010, Springer. Pp. 113–139.

[90] Thompson, S., Beech, D., Maloney, M. & Mendelsohn, N. XML Schema Part 1: Structures Second Edition [WWW]. World Wide Web Consortium. 2.5.2001, 28.10.2004 [Referenced 5.5.2010]. Available:
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[91] Tsai, W.T., Lee, Y.H., Cao, Z., Chen, Y. & Xiao, B. RTSOA: Real-Time Service-Oriented Architecture. Second IEEE International Symposium on Service-Oriented System Engineering (SOSE 2006), Shanghai, China 25.–26.10.2006. 2006, IEEE. Pp. 49–56.

[92] UBR Shutdown FAQ [WWW]. Microsoft. [Referenced 6.5.2010]. Available:
http://uddi.microsoft.com/about/FAQshutdown.htm

[93] VTT Lipasto. [Referenced 20.9.2010]. Available: http://lipasto.vtt.fi/

[94] Web Services for Devices [WWW]. [Referenced 16.6.2010]. Available:
http://www.ws4d.org/

[95] WebSphere Business Modeler V6.2 – Modeling elements and flows. IBM. 23.4.2009 [Referenced 1.9.2010]. Available:
http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wpi_v6/wbmodeler/6.2/Modeler-PubServer/WBPMv62_Modeler_ModelingElements/player.html

[96] Weske, M. Business Process Management: Concepts, Languages, Architectures. 2007, Springer. 368 p.

[97] What Is a Business Process Orchestration Diagram? Microsoft. [Referenced 1.9.2010]. Available:
http://msdn.microsoft.com/en-us/library/aa560068(BTS.20).aspx

[98] White, S.A., Anthony, M., Arkin, A., Astier, S., Bartel, R., Barkmeyer, E., Bock, C., Burbank, D., Carlsen, S., Chobantonov, P., Corda, U., Cummins, F., Daniel, B., Fletcher, T., Forgey, S., Frank, K., Giraud, J.L., James, B., Keeling, G., Klink, M., Lonjon, A., Martin, M., Mason, L., McCabe, F., Moberg, D., Owen, M., Rivett, P., Samoojh, S., Sanchez, J., Shapiro, R., Smith, B., Sturm, M., Suryanarayanan, B., Vanchu-Orozco, M., Williams, D. & Wuethrich, P. Business Process Model and Notation (BPMN) version 1.2 [WWW]. 4.1.2009 [Referenced 12.5.2010]. Available:
http://www.omg.org/spec/BPMN/1.2/PDF

[99] Wolf, M.A., Pennington, D., Chomkhamsri, K., Pant, R., Pretato, U. & Bersani, R. ILCD Format: Scope, Development, Compatibility [WWW]. European Commission Joint Research Center (JRC). 17.11.2008 [Referenced 18.10.2010]. Available:
http://lca.jrc.ec.europa.eu/eplca/Deliverables/news_files/ILCD_Format_JRC_ILCDWorkshops17-18_and_19Nov2008.pdf

[100] WorldSteel Association. [Referenced 20.9.2010]. Available:
http://www.worldsteel.org/

A. APPENDIX: XML SCHEMATA OF FOOTPRINT ESTIMATOR

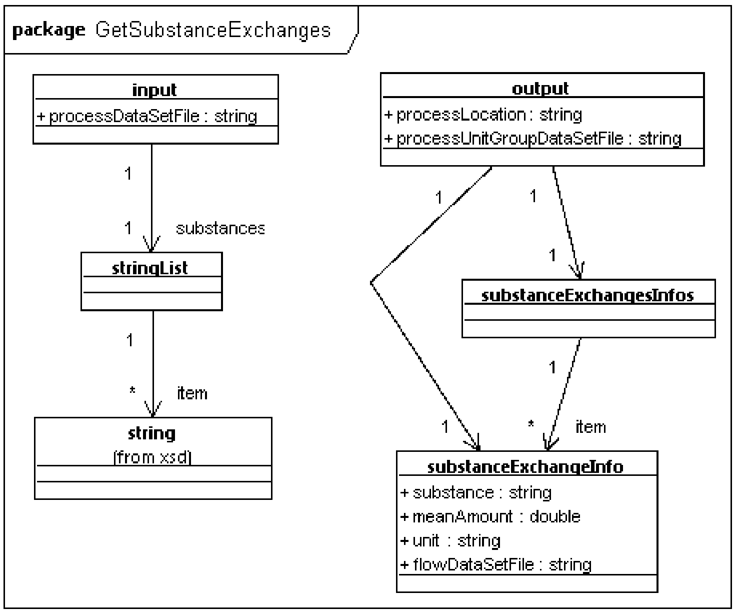


Figure A.1: The XML schema of GetSubstancesExchanges business process.

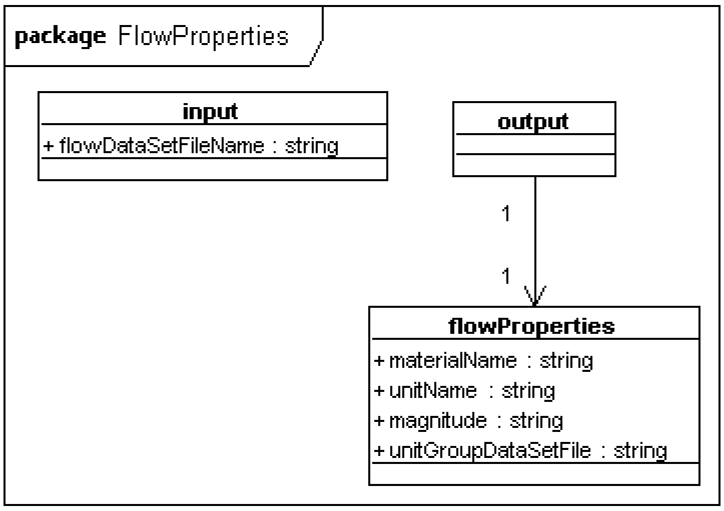


Figure A.2: The XML schema of GetFlowProperties business process.

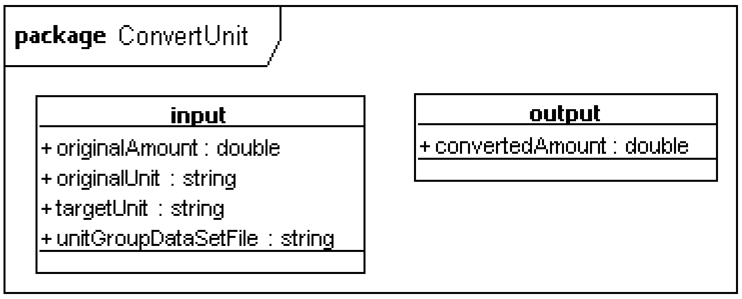

package ConvertUnit
input
+ originalAmount : double
+ originalUnit : string
+ targetUnit : string
+ unitGroupDataSetFile : string
output
+ convertedAmount : double

Figure A.3: The XML schema of ConvertUnit business process.

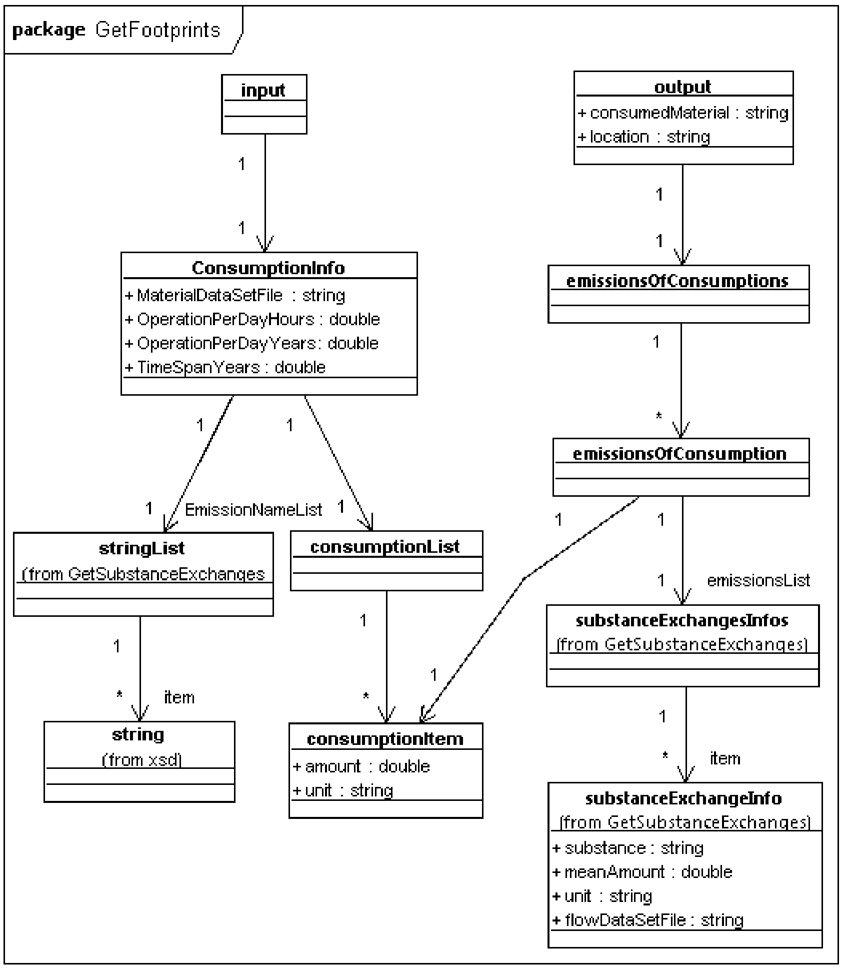


Figure A.4: The XML schema of GetFootprints business process.

B. APPENDIX: XML SCHEMATA OF CONDITION MONITORING

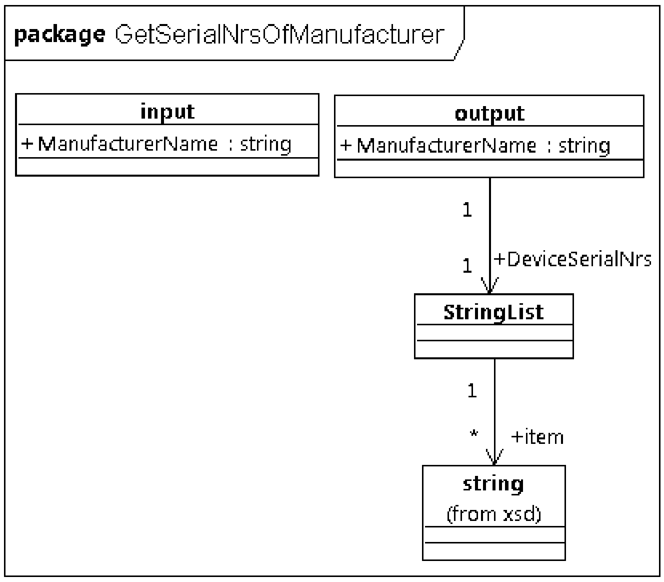

package GetSerialNrsOfManufacturer
input
+ManufacturerName : string
output
+ManufacturerName : string
1
1
+DeviceSerialNrs
StringList
1
*
+item
string
(from xsd)

Figure B.1: The XML schema of GetSerialNrsOfManufacturer business process.

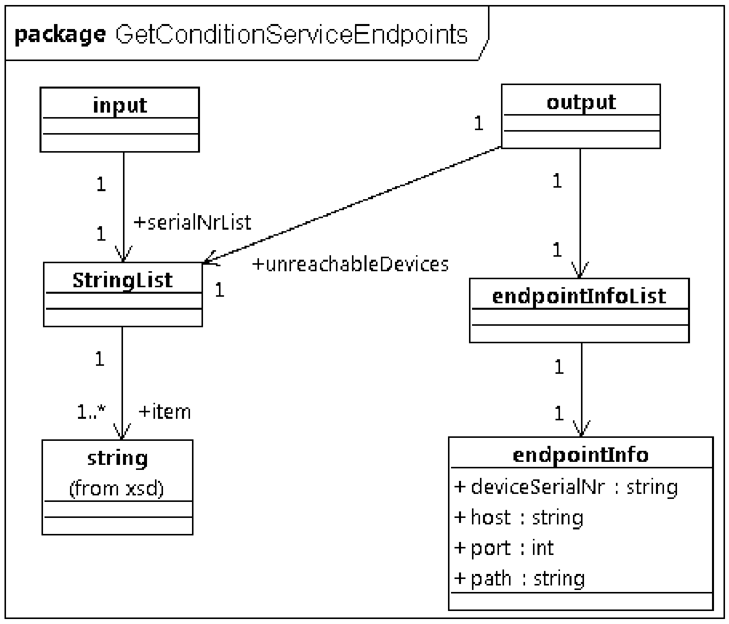


Figure B.2: The XML schema of GetConditionServiceEndpoints business process.

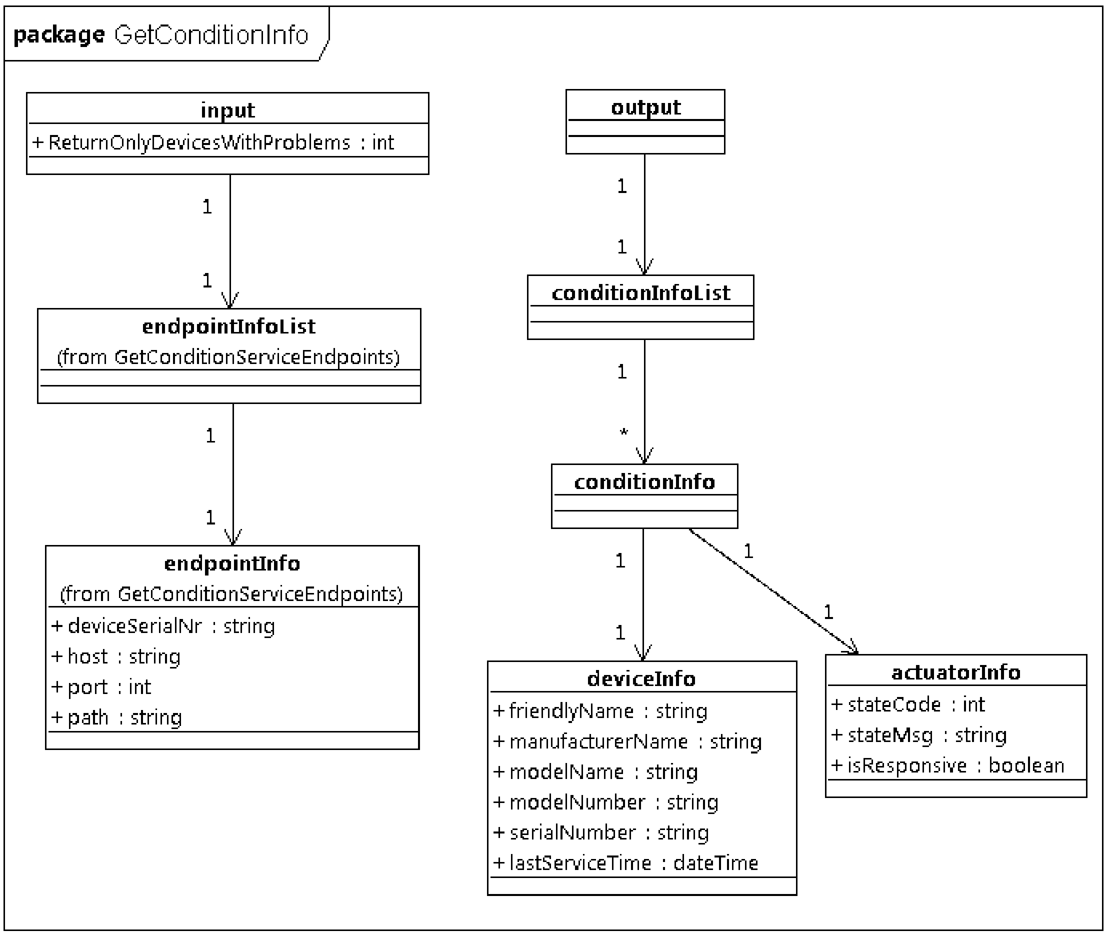


Figure B.3: The XML schema of GetConditionInfo business process.

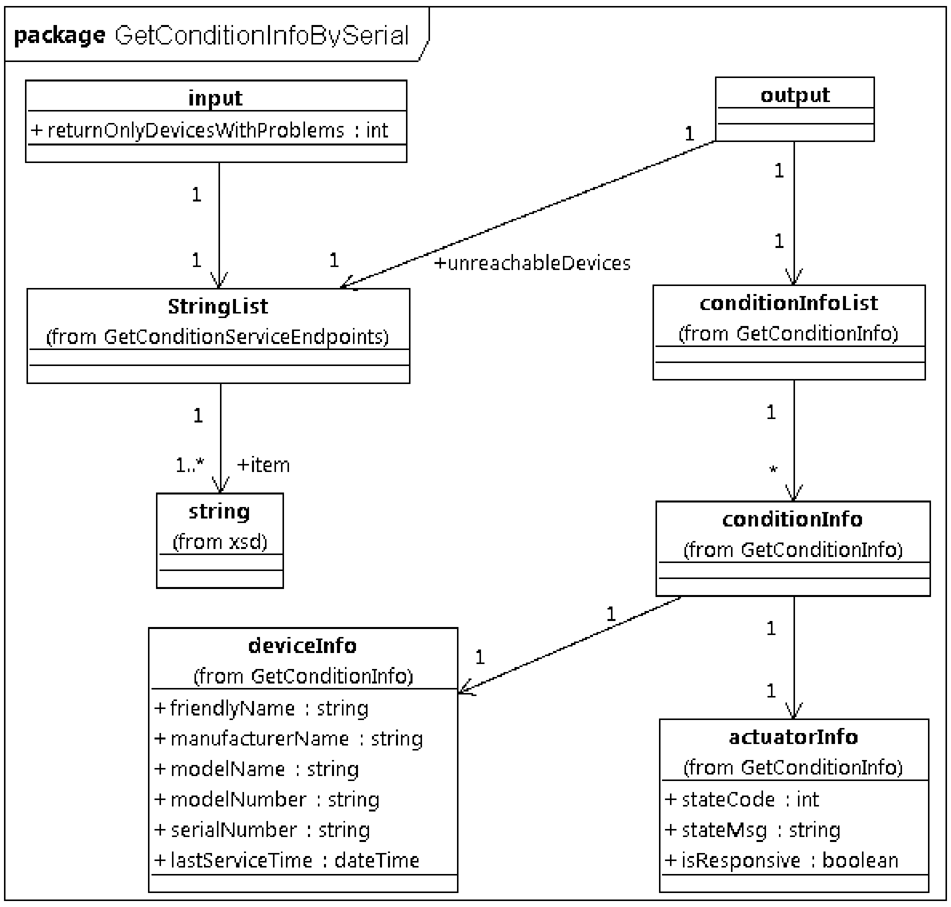


Figure B.4: The XML schema of GetConditionInfoBySerialNr business process.

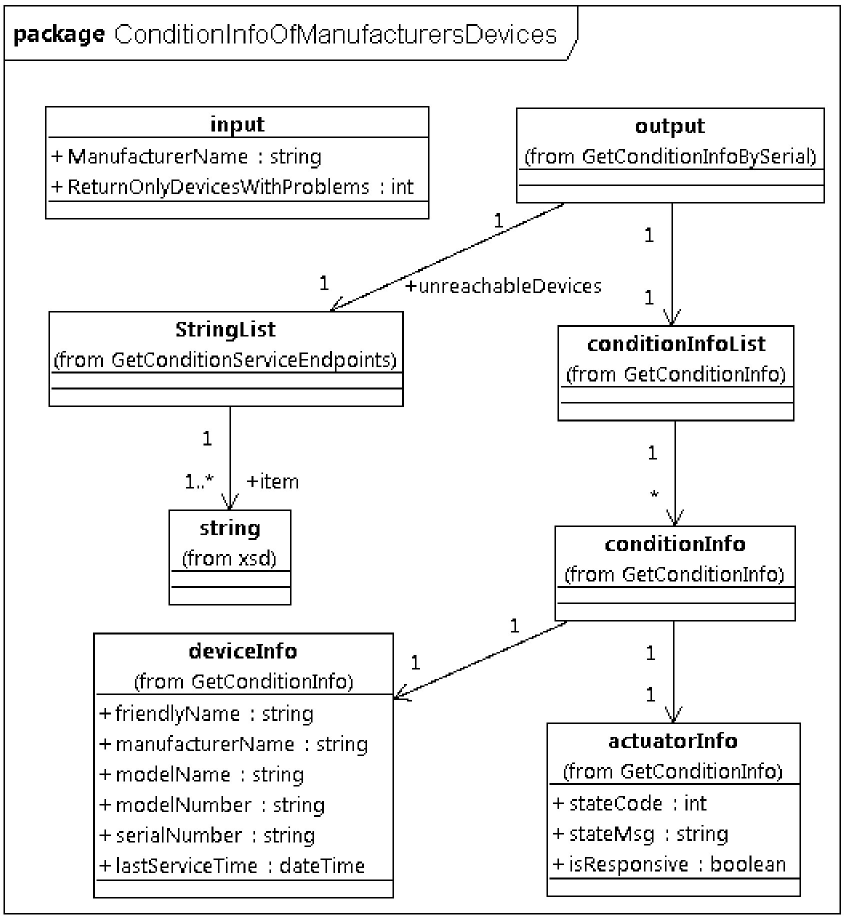


Figure B.5: The XML schema of ConditionInfoOfManufacturersDevices business process.