Internet of Things Architecture

IoT-A

Project Deliverable D2.3 – Orchestration of distributed IoT service interactions

Project acronym:	IOT-A
Project full title:	The Internet of Things Architecture
Grant agreement no.:	257521

Doc. Ref.:	D2.3	
Responsible Beneficiary :	UNIS	
Editor(s):	Stefan Meissner (UNIS)	
List of contributors:	Stefan Debortoli (SAP), Klaus Sperner (SAP), Carsten Magerkurth(SAP), Dan Dobre (NEC), Stefan Meissner (UNIS)	
Reviewers:	SAP	
Contractual Delivery Date:	29.02.2012	
Actual Delivery Date:	15.03.2012	
Status:	Draft	
Version and date	Changes	Reviewers / Editors
V01- 03.05.2011	Table of Contents	Stefan Meissner (UNIS)
V02- 01.12.2011	Section 2 added	Stefan Debortoli (SAP)
V03- 26.01.2012	Added section on Business Process Execution Engines	Klaus Sperner (SAP)
V04- 06.02.2012	First version of Phase Model	Klaus Sperner (SAP)
V05- 17.02.2012	Section 4.2 added	Dan Dobre (NEC)
V06- 26.02.2012	Chapter 1,7,5.3 added	Carsten Magerkurth (SAP)
V07- 06.03.2012	Section 5.2 added, Self-healing diagram added	Stefan Meissner (UNIS)

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

	Dissemination Level	
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the Consortium (including the Commission Services)	
CO	Confidential, only for members of the Consortium (including the Commission Services)	

Executive Summary

IoT-A, the Internet of Things – Architecture, proposes the creation of an architectural reference model for the Internet of Things together with a definition of an initial set of key building blocks. One of the main outcomes of IoT-A is the efficient integration of Internet of Things systems into the service layer of the Future Internet. Work package 2 deals with the integration of real-world (IoT) services and resources with applications and processes of the Future Internet. This deliverable contributes towards the work package 2 objective of adaptive orchestration of IoT resources and services with enterprise services.

The service layer of the Future Internet will support domain experts to model applications that make use of IoT services. Work package 2 addresses **Application Support** by providing business process modelling techniques tailored to the needs and specifics of the Internet of Things as well as the complementary **Process Execution & Service Orchestration** methods.

In this deliverable we discuss how distributed IoT services should be orchestrated taking the IoT-specific characteristics into account. The document starts with a comparison of service orchestration and service choreography though as choreography turned out to be a promising approach that contributes very well to scalability, but due to the resource constraints of today's IoT devices this approach is not feasible yet. After that the report analyses existing service orchestration systems and service composition concepts. Furthermore IoT-specific requirements of service orchestration collected from work package 6 are presented and existing orchestration and composition concepts are evaluated against those requirements. The main contribution of this deliverable is the initial IoT-aware service orchestration concept that makes use of the Virtual Entity Resolution and IoT Service Resolution infrastructure being developed in work package 4.

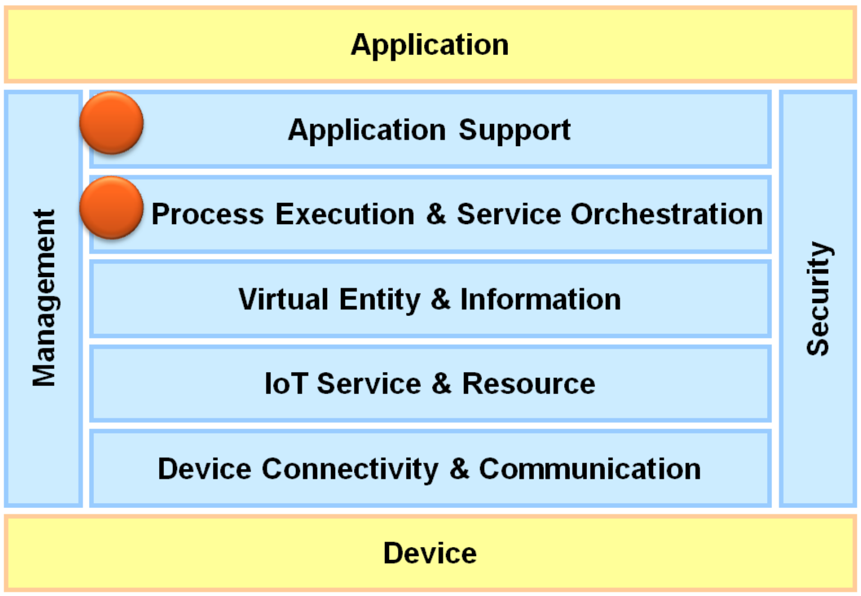

Application
Management
Application Support
Process Execution & Service Orchestration
Virtual Entity & Information
IoT Service & Resource
Device Connectivity & Communication
Security
Device

Figure 1 Functional Groups tackled in D2.3

Table of Acronyms

Acronym	Definition
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation (up to Version 1.2) Business Process Model and Notation (from Version 2.0)
D	Deliverable
IOPE	Input Output Precondition Effect
IoT	Internet of Things
IoT-A	Internet of Things – Architecture
IR	Internal Report
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OSGi	Open Services Gateway initiative
QoS	Quality of Service
REST	Representational State Transfer
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
SQR	SENSEI's Semantic Query Resolver
SRI	Service Resolution Infrastructure
UML	Unified Modeling Language
USDL	Universal Service Description Language
WfMC	Workflow Management Coalition
WP	Work Package
WPDL	Workflow Process Definition Language

WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
XPDL	XML Process Definition Language

Table of Content

Executive Summary .... - 1 -

1. Introduction .... - 10 -

1.1 Purpose of Deliverable .... - 10 -

1.2 Structure of Document .... - 10 -

2. Orchestration vs. Choreography .... - 12 -

2.1 Purpose .... - 12 -

2.2 Introduction .... - 12 -

2.3 Business Process vs. Service .... - 12 -

2.4 Current State of the Art .... - 13 -

2.4.1 Modelling Interactions .... - 13 -

2.5 Findings based on use case “sensor based quality control” .... - 13 -

2.5.1 Introduction .... - 13 -

2.5.2 Short description .... - 13 -

2.5.3 Orchestration Approach .... - 15 -

2.5.3.1 Collaboration Process Model .... - 15 -

2.5.3.2 Description from a Process Perspective .... - 16 -

2.5.3.3 Analysis .... - 16 -

2.5.4 Choreography Approach .... - 18 -

2.5.4.1 Redesign .... - 18 -

2.5.4.2 Analysis .... - 25 -

2.5.4.3 Conclusion .... - 26 -

2.5.5 Requirements .... - 27 -

2.6 Conclusion and Recommendation .... - 28 -

3. Existing service orchestration Concepts .... - 30 -

3.1 BPMN orchestration engines .... - 30 -

3.1.1 Historical overview of business process execution .... - 30 -

3.1.2 Current business process execution engines .... - 31 -

3.1.2.1 General Principle of Business Process Execution.................................... - 32 -

3.1.2.2 Activiti .... - 32 -

3.1.2.3 Bonita Open Solution .... - 34 -

3.1.2.4 jBPM .... - 35 -

3.2 Web service composition systems.. - 38 -

3.3 Related Research Initiatives .... - 41 -

4. Requirements of the IoT-aware Service Orchestration Concept...................... - 45 -

4.1 Methodology .... - 45 -

4.2 IoT-specific Characteristics of Service Orchestration .... - 45 -

4.2.1 Functional IoT-Aware Service Composition Aspects .... - 45 -

4.2.2 Non-functional Aspects of IoT-Aware Service Composition........................ - 46 -

4.2.2.1 Performance.. - 47 -

4.2.2.2 Dependability .... - 48 -

4.2.2.3 Security and Trust .... - 48 -

4.2.2.4 Cost and Payment .... - 48 -

4.2.3 Service Choreography Aspects .... - 48 -

4.3 Evaluation of Existing Service Orchestration Concepts and Tools .... - 49 -

4.4 Conclusion... - 50 -

5. IoT-aware Service Orchestration Concept .... - 51 -

5.1 Service Orchestration Phase Model .... - 51 -

5.1.1 Modelling .... - 52 -

5.1.2 Resolution .... - 54 -

5.1.3 Binding... - 54 -

5.1.4 Execution.. - 54 -

5.2 Service Composition .... - 55 -

5.3 Service Execution.. - 59 -

5.3.1 Context Management Framework .... - 59 -

5.3.2 Real World Integration Platform .... - 60 -

5.3.3 Integration .... - 60 -

5.4 IoT-Aware Aspects of Service Orchestration .... - 61 -

5.4.1 Self-configuration in Service Orchestration .... - 61 -

5.4.2 Self-optimization in Service Orchestration .... - 62 -

5.4.3 Self-healing in Service Orchestration .... - 62 -

5.4.4 Self-protection in Service Orchestration .... - 63 -

5.5 Conclusions .... - 64 -

6. Conclusions and Outlook .... - 65 -

Glossary .... - 66 -

References .... - 72 -

Index of Figures

Figure 1 Functional Groups tackled in D2.3 .... - 1 -

Figure 2 Extended sensor based quality control (Hagedorn et al. 2011) .... - 15 -

Figure 3 Sensor based quality control: conversation model .... - 19 -

Figure 4 Sensor based quality control: high-level behavioural model .... - 19 -

Figure 5 Service resolution: conversation model .... - 20 -

Figure 6 Service resolution: register augmented entity .... - 21 -

Figure 7 Service resolution: service lookup .... - 21 -

Figure 8 Sensor based quality control: choreography #1 .... - 22 -

Figure 9 Sensor based quality control: choreography #2 .... - 22 -

Figure 10 Sensor based quality control: conversation model #1 .... - 23 -

Figure 11 Sensor based quality control: conversation model #2 .... - 24 -

Figure 12 Historic evolvement of Business Process Modelling Languages (Bartonitz 2009) .... - 30 -

Figure 13 Activiti Components (Activiti 2012) .... - 33 -

Figure 14 Representation of BPMN models within Activiti .... - 33 -

Figure 15 Class Hierarchy of FlowNodeActivityBehaviour .... - 34 -

Figure 16 Bonita Open Solution Components (Bonitasoft 2012) .... - 35 -

Figure 17 Overview of jBPM Components (jBoss Community 2012) .... - 36 -

Figure 18 Representation of process models in jBPM .... - 37 -

Figure 19 Class Hierarchies of Node and NodeInstance .... - 38 -

Figure 20 Life-cycle for the dynamic composition of services (Grossmann et al. 2011) .... - 39 -

Figure 21 SENSEI Architecture (Martin Strohbach et al. 2010) .... - 42 -

Figure 22 SENSEI Entity Binding schema (Martin Strohbach et al. 2010) .... - 42 -

Figure 23 Re-planning of Service Compositions (Villalonga et al. 2010) .... - 43 -

Figure 24 SENSEI Service Execution Monitoring (Villalonga et al. 2010) .... - 43 -

Figure 25 Taxonomy of QoS Attributes of Services and Compositions (Rosenberg 2009) .... - 47 -

Figure 26 Sequence Diagram for Resolution, Binding, and Execution of IoT Services .... - 51 -

Figure 27 Examples for SensingTask, ActuationTask, and PhysicalObject as proposed in (Sperner, Meyer, and Magerkurth 2011) .... - 52 -

Figure 28 Resolution of IoT Service for exactly specified PhysicalObject .... - 54 -

Figure 29 Input Output Matching - 55 -

Figure 30 Input Output Matching Example - 56 -

Figure 31 Precondition Effect Matching - 56 -

Figure 32 Precondition Effect Matching Example - 57 -

Figure 33 Decomposition of Services - 58 -

Figure 34 High level architecture of the Context Management Framework consisting of coordinating Context Agents - 59 -

Figure 35 Real World Integration Platform - 60 -

Figure 36 Self-Healing Service Subscription - 63 -

Index of Tables

Table 1 - Sensor based quality control: advantages of orchestration approach ....- 17 -

Table 2 - Sensor based quality control: disadvantages of orchestration approach ....- 17 -

Table 3 - Sensor based quality control: evaluation of IoT-characteristics.......................................- 17 -

Table 4 - Sensor based quality control: choreography messages ....- 24 -

Table 5 - Sensor based quality control: advantages of choreography approach.............................- 25 -

Table 6 - Sensor based quality control: disadvantages of choreography approach ....- 25 -

Table 7 - Sensor based quality control: evaluation of IoT characteristics in choreography..............- 26 -

Table 8 - Sensor based quality control: service interaction patterns ....- 26 -

Table 9 - List of requirements...- 28 -

Table 10 Comparison of dynamic composition approaches (Grossmann et al. 2011) ....- 40 -

Table 11 Comparison of Web Service Composition Approaches (Dessislava and Dimov 2011).....- 41 -

Table 12 Functional Requirements ....- 46 -

Table 13 Evaluation of Service Orchestration Concepts..- 49 -

1. Introduction

This deliverable focuses on service orchestration. Within the context of the IoT-A project, it fulfils a crucial interfacing function, as it bridges the gap between the different work packages 4 and 2, thus building upon the results of IoT service resolution as it is contributed by work package 4 and at the same time utilising this service resolution functionality in order to dynamically create and adapt higher level orchestrated services that are bound by the IoT-A process execution which work package 2 contributes to the project.

Correspondingly, this deliverable also fulfils an interfacing function within work package 2. In the context of the first batch of deliverables that are due M18, it logically connects the first deliverable D2.1 (Resource Description Specification) with D2.2 (Concepts for Modeling IoT-Aware Processes) by applying the resources and services specifications defined in D2.1 to the concept of service orchestration, so that the process layer as it is discussed in D2.2 can utilise and interpret the service specifications in order to bind the most appropriate orchestrated IoT-aware services to the respective process steps and activities. Therefore, regardless of the direction or perspective we take, service orchestration is located in between the domains of process execution and simple service execution and effectively links them together. Without this link both worlds would remain separate and one of the fundamental problems of the Internet of things, namely the rudimentary conceptual connection to enterprise systems would remain unsolved.

1.1 Purpose of Deliverable

The primary aim of this deliverable is to develop and discuss a service orchestration concept for the IoT-A project that is appropriate for the realisation within a Future Internet service layer, as it must integrate the tradition of process orchestration concepts that come from the enterprise world as well as the idiosyncrasies of IoT. Apart from the close interconnection with process execution, the service orchestration concept must also do justice to the scalability issues typically found in IoT systems, so that the concept has to take decentralised autonomous configuration, organisation, management, and repairing capabilities into account. If these self-* properties are reflected, a deployment in real world large scale systems becomes possible. For the realisation of the limited use cases implemented within the course of the project, the interconnection with the process execution layer, as it is discussed in D2.2, has a stronger focus.

Apart from these two aspects, the deliverable also dares to take a more fundamental look at the problems of making different individual services work together in a sensible way. Insofar, the deliverable also explores the different paradigms or perspectives of realising this. Consequently, the deliverable examines very intensively the advantages, disadvantages, and implications of the service orchestration paradigm compared to the service choreography paradigm. While it may not always be feasible to realise service choreography with the constraints and limitations of today's technologies, it is definitely worthwhile to conceptually take the potential benefits and implications of service choreography into account, when anticipating future IoT service architectures. Therefore, a final purpose of this deliverable is also to have a more fundamental and radical look at service orchestration or choreography in order to contribute to a future research agenda in that field.

1.2 Structure of Document

The deliverable starts off in chapter 2 with a fundamental discussion of the different means of combining individual services, namely the question of service orchestration versus choreography. It especially considers the entire life cycle of augmented entities that may exist before and after the typical use case situations that we envision in work package 7, and the applicability of certain IoT characteristics such as distributed execution, distributed data, and scalability, but also takes a realistic and pragmatic look on the potential means of implementation with today’s technologies.

In the next chapter 3, the state of the art on service orchestration concepts is discussed and evaluated in the light of IoT aware services. An important underlying consideration is the integration into existing service orchestration approaches, in order to ensure the external relevance of the IoT-A concept. In that respect, the focus of our discussion aims at two different threads. The first one is the traditional business process management, for which service orchestration has been a central topic ever since. Correspondingly, we also address the other big trend in service orchestration, namely the web technologies that were developed with the proliferation of the Internet.

Chapter 4 then addresses the concrete requirements towards the service orchestration concept. We try to bring together the findings from the evaluation of existing service orchestration concepts and tools with the specific requirements from the Internet of things. This combination aims at doing justice both to the specificity of IoT systems and the IoT-A project in general, as well as to the constraints and implications of the existing architectures that exist in the real world, outside the domain of the Internet of things.

Chapter 5 then integrates the results of the preceding three chapters in order to present the actual IoT-aware service orchestration concept developed in work package 2. The chapter starts with the service orchestration model, which outlines the phases of modelling, resolution, binding, and execution of services. Subsequently the composition of atomic services to form higher order services is discussed. After a detailed presentation of dedicated frameworks for the execution of IoT-aware services, the IoT-aware aspects of service orchestration are analysed in detail, in order to justify the applicability of the model to large-scale Internet of Things systems.

The concept discussed in chapter 5 and in the broader context of this deliverable is then finally set in the context of the other activities in the project within the concluding remarks of chapter 6.

2. Orchestration vs. Choreography

2.1 Purpose

This chapter should outline the differences between orchestration and choreography of services and processes. A use case helps to illustrate the various IoT specific aspects and serves as a basis for the structured research approach. The goal is to come up with a recommendation for choosing the appropriate coordination paradigm in the IoT.

2.2 Introduction

An analysis of different IoT use cases of the retail sector shows that the paradigm of business process and service orchestration (Peltz 2003), viz. having one central and powerful backend system in place, which coordinates and integrates the whole process and its participants via (web-)services and message exchange, is still mostly used for modelling and designing new IoT applications. This observation leads to the problem that an augmented entity loses its remarkable IoT functionality, if no appropriate backend system with it can interact is available. By considering the following generic product life-cycle, this issue arises in every stage of the life-cycle, except for the ones in which the interaction is possible. Starting at the manufacturer, a consumer good usually moves across a logistics and distribution network to a retailer, before the customer buys and takes the product home for further usage. The last step is the disposal of the used product. In this case, the entity changes its environment at least five times across the life-cycle. Therefore, in order to enable a seamless integration of so-called *smart items* or *smart objects* with enterprise IT systems across the whole augmented entity's life-cycle, the concept of business process and service decomposition, respectively, needs to be applied (Haller, Karnouskos, and Schroth 2008). This approach is "more collaborative and allows each involved party to describe its part in the interaction", explains (Peltz 2003). This interaction style, called choreography, tracks the message sequences among multiple parties and sources rather than a specific business process that a single party executes (Peltz 2003).

Thus, requirements for enabling business process and service choreographies and the integration of an IoT enabled physical entity into the environment of all its life-cycle stages need to be elaborated.

2.3 Business Process vs. Service

The emergence of service oriented architectures (SOA) allows the composition of various services to processes (Weske 2007). The definition of a service, as stated by Steve Burbeck, says that "services are loosely-coupled computing tasks communicating over the internet that play a growing part in business-to-business interactions" (Weske, 2007, p. 58). Business processes are usually steered by a business process execution engine or a workflow management system, which have an underlying service-oriented architecture (Weske 2007). Since "web services are the current realization of service-oriented computing" (Weske, 2007, p. 315), automated tasks in business processes are therefore also implemented by web services (Barros, Dumas, and Ter Hofstede 2005). These web services can either be used in process orchestrations, but also in process choreographies, i.e. for exchanging messages between business partners over the internet. Therefore, it can be said that services are used for business process implementations. Thus, as a definition for this chapter, a process is one level of abstraction higher than a service. Consequently, when talking about choreographies, the terms process choreography and service choreography describe the automated implementations of processes, just with the focus on different layers.

2.4 Current State of the Art

2.4.1 Modelling Interactions

Dependencies and interactions of business processes or service orchestrations between enterprises are usually captured by service choreographies. This kind of business-to-business collaboration is usually realized by sending and receiving messages of individual process orchestrations (Weske 2007). Introducing collaboration rules helps the enterprises to establish a well-defined B2B communication. Furthermore, implementation costs can be reduced because interaction behaviour does not have to be defined with each and every business partner, since industry-wide standards serve as references for the specific types of interactions (Weske 2007). Service choreographies specify these collaboration rules. Various industry domains already defined their own standards. Examples include RosettaNet for the supply chain domain, SWIFTNet for financial services and Health Level Seven for health care services (Weske 2007). However, all these standards lack of flexibility to be extended in order to define new business-to-business collaborations, especially supporting IoT applications.

As there exist control flow patterns for process orchestrations (Russell et al. 2006), there also exist service choreography patterns which serve as granular, reoccurring types of interactions, as well as a benchmark of various modelling languages (Barros, Dumas, and Ter Hofstede 2005). Basic examples for service choreography patterns are *send*, *receive*, *send / receive*, etc. A comprehensive list can be found in the work of (Barros, Dumas, and Ter Hofstede 2005) and (Weske 2007, p. 249).

Only a few standardized languages for process choreography modelling exist. The most famous ones are the academic language Let's Dance (Zaha et al. 2006) and BPMN (Object Management Group 2011) in its latest version 2.0. Other languages are for example BPEL^{light}, BPEL^{gold}, BPEL4Chor, WSFL, WS-CDL, iBPMN, BPSS/UMM, and SCA (Kopp, Leymann, and Wagner 2011). However, by assessing the languages based on the previously mentioned service choreography patterns, neither one of them support all discovered patterns (Weske 2007; Barros, Dumas, and Ter Hofstede 2005). Therefore, more research is required in this field. For this project, the selected choreography modelling language is BPMN 2.0 due to its flexibility for extensions and possible future compatibility to process orchestration models, which also can be modelled with BPMN 2.0. A comprehensive argumentation for choosing BPMN 2.0 can be found in the deliverable D2.2 of WP2 in the IoT-A project and in the work of (Meyer et al. 2011).

2.5 Findings based on use case “sensor based quality control”

2.5.1 Introduction

IoT technologies can be applied in numerous business domains to improve customer satisfaction, to reduce costs across the whole supply chain, and to increase the transparency of all business objects. Due to the almost infinite number of use cases which can occur among all domains, the example use case was extracted from deliverable D7.1 of WP7 (Hagedorn et al. 2011).

2.5.2 Short description

This use case shows how sensor networks monitor perishable goods in a store. The sensor infrastructure measurements are used for estimating the quality of a rare and expensive form of Chinese orchids. Depending on the luminance, humidity, and temperature of the environment, the estimated future quality of the orchids is determined and prices are reduced, even before a perceivable degradation of quality occurs. By applying this sensor based quality control and combining it with dynamic pricing, it is ensured that the goods are sold before quality degradation is likely to occur (Hagedorn et al. 2011).

In order to extend the existing functionality with another similar feature, the triggering of the orchid’s disposal (either shelf removal in the store) has been added. This disposal is triggered if the orchid’s calculated quality factor drops below a certain threshold.

2.5.3 Orchestration Approach

2.5.3.1 Collaboration Process Model

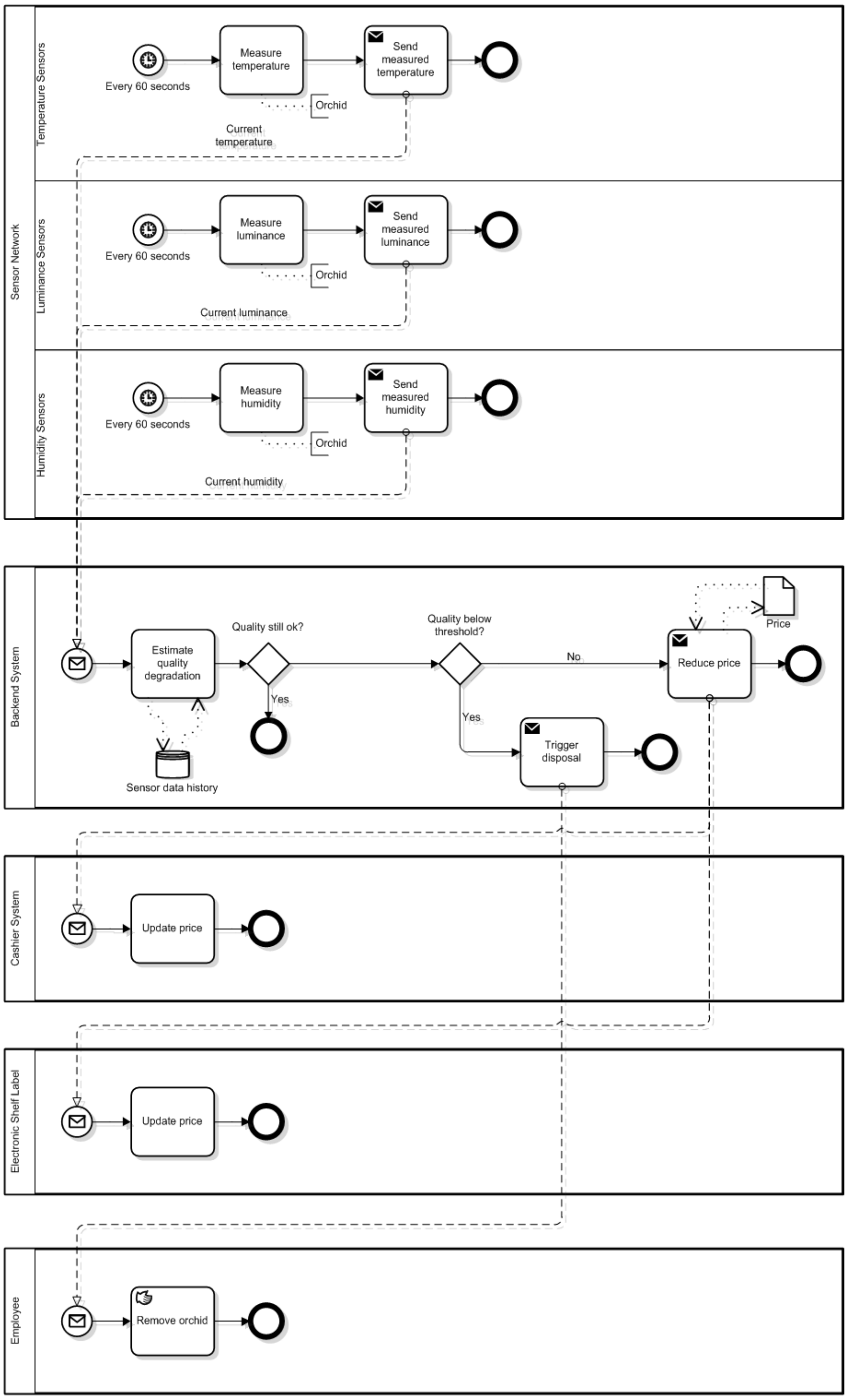


Figure 2 Extended sensor based quality control (Hagedorn et al. 2011)

2.5.3.2 Description from a Process Perspective

Smart sensors connected to the store's backend system continuously monitor certain environmental parameters, which are crucial for the quality of the fragile orchids, as e.g. the humidity level must be in a certain range, so that neither dryness nor inundation, neither warm nor chill, nor non-optimal luminance harms the quality of the orchids. As it is difficult to maintain optimal conditions in a supermarket, the sensors would only send alerts to the backend system on severely inappropriate conditions, but the average conditions over time are still measured to calculate the estimated point in time for a perceivable degradation in quality (Hagedorn et al. 2011).

As the store management focuses on delivering the highest quality possible, the backend system is configured to automatically reduce the price of perishable goods already before a degradation of quality can be perceived by the consumer. Based on environmental measurements from humidity, luminance, and temperature sensors, the platform automatically reduces the price at the point of sale (POS) in order to foster the sales of the orchids, if a threshold of quality degrading environmental parameters is reached. The backend system utilizes a component called "Campaign Generator Agent" that allows for overriding price information from the retail ERP system to account for local adaptations in prices due to e.g. quality degradations or shelf space issues. Data from the backend system is then propagated to the cashier system and also wirelessly to electronic shelf labels (ESLs) that function as complementary IoT actuators to the sensors that measure the environmental conditions of the orchids (Hagedorn et al. 2011).

If the calculated quality factor drops below a certain threshold, the backend system triggers the disposal of the orchid by sending a message to an employee's mobile phone, which then tells him or her to manually remove the product from the shelf.

2.5.3.3 Analysis

From a business and industry perspective, the use case demonstrates two important retail related concepts: Dynamic pricing and quality control of perishable goods. Dynamic pricing as a real-time tool for price optimization strategies has always been crucial for profit maximization. In contrast to the state of the art, dynamic pricing in the featured use case is not performed on static information such as best before end dates in the transaction data of the backend ERP system, but it is based on real time IoT data gathered from a sensor infrastructure (Hagedorn et al. 2011).

As about 20% of perishable goods never reach the consumer, but are disposed of before, either in the store or in the supply chain, the utilization of IoT sensors is also an interesting concept to implement quality control of perishables and thus reduce waste and increase profits at the same time (Hagedorn et al. 2011).

However, due to various laws with regards to health and safety, prices cannot drop down to zero if the product is unusable or uneatable. Therefore, faulty products need to be removed from the shelves urgently. This use case can be applied to any perishable product and assures a guaranteed minimum quality to the customer, based on product-specific, real-world, and real-time data.

By taking a look at the collaboration model, a strong dependency of the backend system can be seen since most of the application logic is running on it. The following tables evaluate the advantages and disadvantages of this centralized approach, based on an analysis of key process participants.

Process participants	*Advantages*
Sensor network	Pre-configured sensors are cheap and simple devices. Pre-configured sensors have small physical dimensions due to its technical simplicity.
Backend system	Calculations and thresholds are kept very flexible and can be changed when necessary by updating the according functions.

	More functionalities and events, which base on the sensor data, can be added when necessary.

Table 1 - Sensor based quality control: advantages of orchestration approach

Process participants	*Disadvantages*
Sensor network	The permanent transmission of sensor data leads to constantly high network traffic.
Backend system	A backend system is required in order to do simple quality measurements and event triggering. Therefore, a support of the characteristic IoT functions in all stages of the augmented entity's life-cycle cannot be guaranteed. This is because not every stage provides an appropriate backend system, which supports the same quality calculation and event triggering functions, including the definition of the same threshold values. The backend system needs to do lots of calculations, depending on the number of associated monitored products. This leads to limited scalability since all the processing burden lies on the central orchestration engine. These frequent calculations lead to a high CPU and memory load of the backend system, which results in less computation resources for other activities which need to be performed at the same time.

Table 2 - Sensor based quality control: disadvantages of orchestration approach

The initial impression that the backend system plays a central role in this setup was now confirmed after this analysis. (Meyer et al. 2011) also discovered in their work "towards modelling real-world aware business processes" that current approaches "focus on modelling and executing planned processes in a constant enterprise environment" (p. 2). The following table analyses selected Internet of Things characteristics (cf. Meyer et al. 2011), which occur in the sensor based quality control use case, and which have been identified as disadvantages in the previous analysis.

IoT characteristics (cf. Meyer et al. 2011)	*Findings*
Distributed execution	In this use case, the backend system prevents to fully utilize the distributed execution of the business process. The advantage of the interconnection of things and the distributed computation power is not leveraged in this setup.
Distributed data	In this use case, data is only collected from distributed augmented entities, but stored in a central database. This again fosters the dependency on the backend system.
Scalability	The excessive use of the network bandwidth and the computation capacity of the backend system limit the overall scalability. The more augmented entities are added to the system, the more network traffic is generated which will reach its maximum eventually.

Table 3 - Sensor based quality control: evaluation of IoT-characteristics

Summarized it can be said that the presented use case does not completely take advantage of the Internet of Things features in the way it is currently modelled. The previous analysis of the IoT use case has revealed a strong dependency between the augmented entities and a central backend system, which is executing most of the application logic in this setup. The use case was designed with the intention of having a central process execution engine running as part of the backend system, which is controlling the overall composition. The various augmented entities will just have an embedded piece of software installed, which provides communication-interfaces via web-services to the process execution engine. Even though the difference between process orchestration and process choreography is not always very clear and sharp (Santos and Madeira 2006; Peltz 2003), this intended centralized steering and controlling behaviour is an indicator for having a process orchestration.

By considering the augmented entity life-cycle, the lack of support of the IoT functionalities outside the retail store environment, where no appropriate backend system is available, is another indicator for the presence of a process orchestration.

Since process choreographies are more collaborative and less centralized in nature (Santos and Madeira 2006), a process redesign by applying the choreography pattern (Weske 2007) might enable the IoT functionalities across the whole augmented entity life-cycle. This paradigm shift was also suggested by (Haller, Karnouskos, and Schroth 2008) by his concept of *business process decomposition*. Even the IoT-A domain model (cf. Walewski et al. 2011) already considered a possible interaction between physical entities in its design, since the user is described as “a human person or some kind of active digital entity (e.g., a service, an application, or a software agent)” (Walewski et al. 2011), p. 22.

Chapter 2.5.4 provides an alternative process model, which takes the neglected IoT characteristics into account.

2.5.4 Choreography Approach

2.5.4.1 Redesign

Based on the findings in the previous chapter, a redesign of the process model tries to incorporate the choreography pattern into the existing IoT use cases. The underlying development process is based on the research of (Weske 2007), as well as applying the latest version of the modelling language BPMN 2.0. Furthermore, the assumption of having decentralized storage on each augmented entity is made.

The first step in designing a process choreography is creating a high-level structure design, in which the participant roles as well as their communication structures are defined (Weske 2007). The latest version of BPMN 2.0 introduced a new diagram type for that – the so called conversation diagrams (OMG BPMN 2011). In Figure 2 the following participants can be identified in the orchestration setup: a sensor network, a backend system, a cashier system, an electronic shelf label, as well as an employee.

With the intention of having most of the application logic as well as the product data stored on the augmented entity, a new process participant needs to be created, which contains the already existing sensor network, but also adds a resource where the application logic will be running on. Assuming that we are in the setting of a supermarket, multiple orchids may be present, as well as many employees will be working there simultaneously. Another assumption is that price updates of augmented entities always need the backend system as conversation partner. Based on these facts, the following high-level structural model, or conversation model can be created.

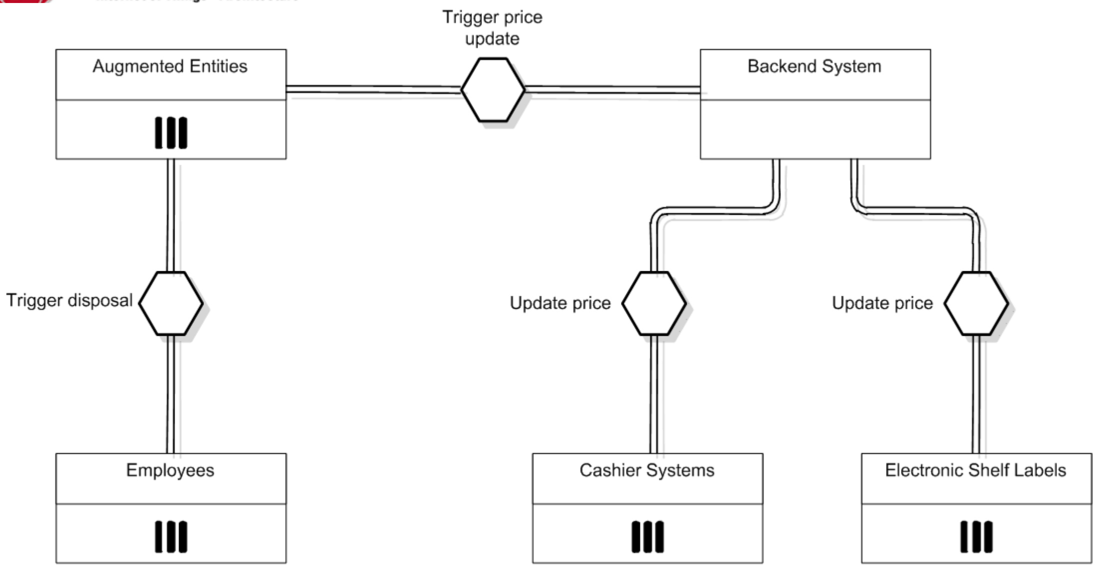


Figure 3 Sensor based quality control: conversation model

As we can see in Figure 3, multiple augmented entities (orchids in the super market use case), consisting out of a sensor network as well as a resource in the form of a process execution engine, communicate with one single backend system for price updates and with multiple employees for shelf removal notifications. Even though the cashier systems and the electronic shelf labels are part of the whole use case, the communication to these participants from the augmented entity point of view is always done through the backend system. Therefore, they are not considered anymore in the following choreography process models.

The second step is the creation of a high level behaviour design, using milestones that are achieved during the collaboration (Weske 2007). By considering the mobile nature of augmented entities in the Internet of Things (cf. IoT characteristics by Meyer et al., 2011) and also the consideration of the entities life-cycles, it is not always guaranteed that an appropriate communication partner is available. Modelled communication partners can unexpectedly become unavailable at execution time, whereas devices, which have not been available at modelling time, might become available at runtime (Meyer et al. 2011). Therefore, some lookup mechanisms are required. The following diagram shows the high-level behavioural model, represented by a series of milestones.

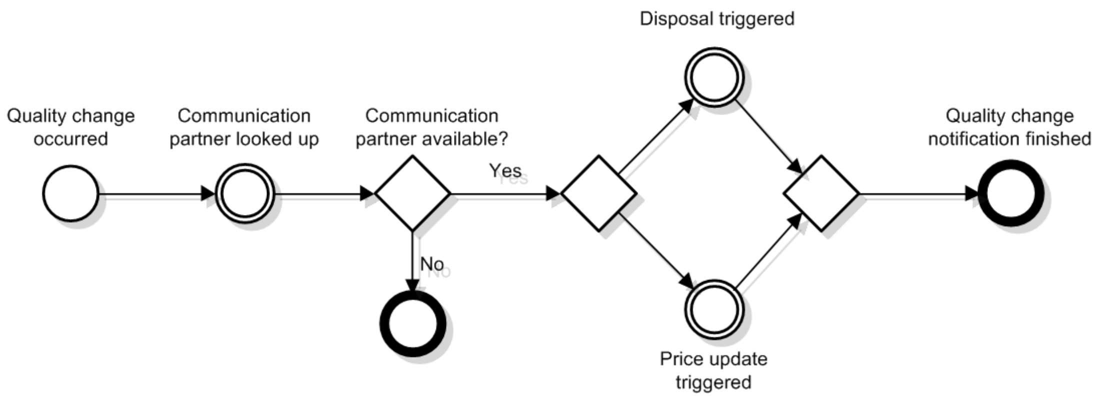

Disposal triggered
Quality change occurred
Communication partner looked up
Communication partner available?
Yes
No
Quality change notification finished
Price update triggered

Figure 4 Sensor based quality control: high-level behavioural model

Standard BPMN elements were used to express the high-level behavioural model. Each milestone indicates a state during the collaboration and has a business meaning. The milestones have dependencies on previous milestones. For example, a quality change must occur before an adequate communication partner is being looked up, or a communication partner needs to be available before a message can be sent.

As the next step, the actual choreographies need to be defined as interactions between the participants, which represent the transitions between the milestones of the behavioural model. Choreographies should be kept small in order to keep the complexity on an appropriate level (Weske 2007). The end event of the single choreography models (e.g. Figure 7) are always the intermediate event of the high-level behavioural model (e.g. "communication partner looked up").

Communication partner looked up. Due to the mobile nature of the augmented entities in the Internet of Things, an appropriate communication partner cannot always be guaranteed. Therefore, a lookup mechanism must be provided which is applicable for all IoT use cases since this functionality is independent from the resulting application, but affects all mobile augmented entities. Since the Internet of Things utilizes existing technologies and protocols like TCP/IP for example, it is assumed that all augmented entities are globally connected with each other over the internet. Therefore, the missing part is now the lookup of a communication partner which is geographically and logically the right partner to communicate. WP4 of the IoT-A project is facing this challenge which enables the lookup and resolution of services of appropriate communication partners. The system will be named “service resolution infrastructure” (SRI) for the remainder of this chapter. However, since this lookup functionality is not use case specific but rather a general Internet of Things technique, a high-level process will be sketched for the sensor based quality control use case and serves as a reference for further lookups. Therefore, all other use case process models also do not include these lookups. An SRI needs to provide three main functionalities: discovery, lookup, and monitoring. Lookup refers to finding an object that can fulfil a given request or task, including resolving its name and address in order to interact with it. Discovery includes lookup but the associated framework may or may not include a fixed repository. Thus, it may include peer-to-peer device interactions, where devices announce their capabilities on a known network channel. Monitoring refers to keeping the dynamic links between resources and entities up-to-date (cf. D4.1:(de las Heras et al. 2012)). The following conversation model shows a possible high-level interaction and lookup process between mobile augmented entities and their requested services.

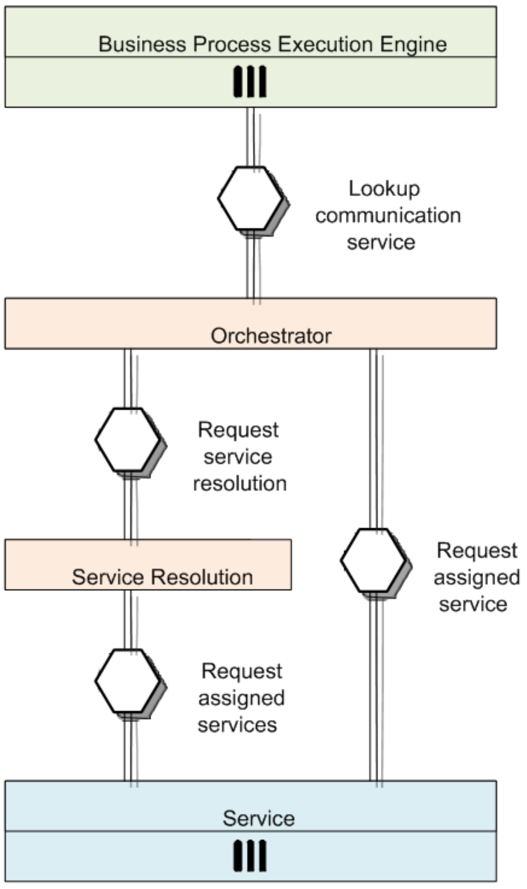

Business Process Execution Engine
Lookup communication service
Orchestrator
Request service resolution
Service Resolution
Request assigned service
Request assigned services
Service

Figure 5 Service resolution: conversation model

Figure 5 shows the conversation model for the service resolution. The business process execution engine is running inside the virtual entity, which is requiring and requesting a compatible service during process run-time. A global system, the service resolution infrastructure, receives the request and tries to locate and report an appropriate service. The basic functionality of such a system can be compared to existing DNS servers, which translate the for humans meaningful domain names into numerical identifiers, which are associated to network devices. However, the Internet of Things requires more functionalities than the existing DNS, e.g. location, availability, security, and privacy issues amongst others. The composition of services based on existing services is done by the Orchestrator as it is shown in Figure 5. An example is the measurement of a building's average temperature, based on the data of multiple temperature sensors, which are composed to an average temperature service. From an augmented entity point of view, the "lookup communication service" conversation needs to be further specified in order to enable the definition of IoT modelling requirements.

In order to make an augmented entity detectable by the service resolution infrastructure and to be available for service consumption by other augmented entities, the availability and capabilities need to be broadcasted to the SRI upon its initial connection to the internet. This registration needs to be done by every participating system within the Internet of Things. The choreography model for the registration process is shown in Figure 6.

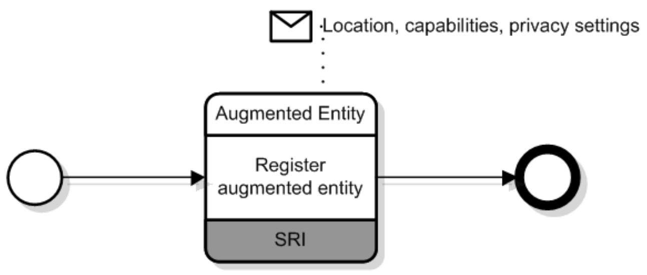

Location, capabilities, privacy settings
Augmented Entity
Register augmented entity
SRI

Figure 6 Service resolution: register augmented entity

Once all available services are registered with the SRI, the services can be looked up and consumed by other augmented entities. Since the services will not be known during the design of the business process, they have to be resolved and bound at run-time. This service lookup process could look like as it is shown in Figure 7.

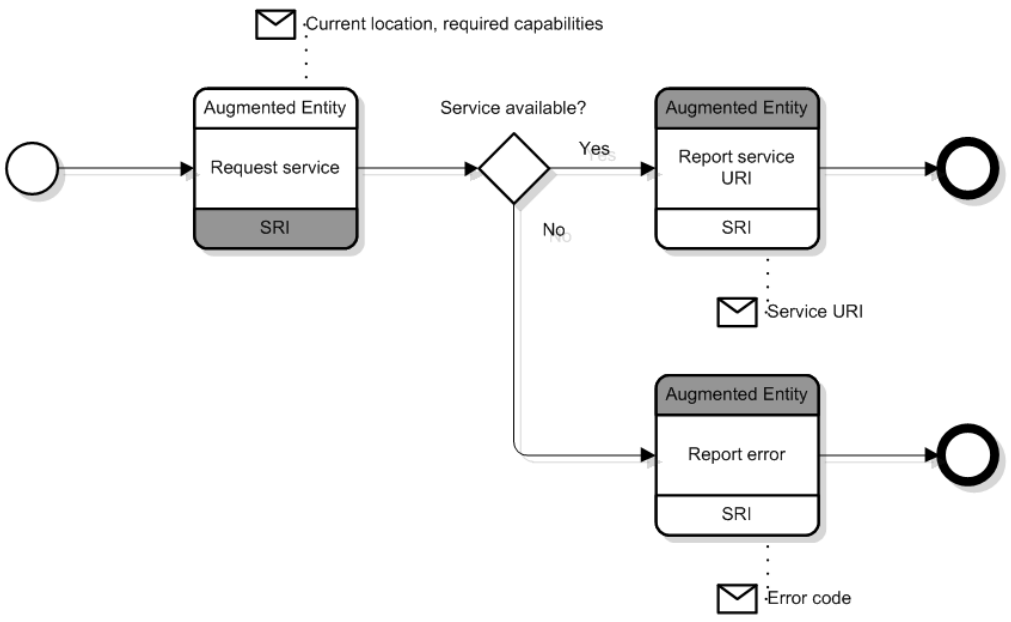


Figure 7 Service resolution: service lookup

Disposal triggered. In order to reach the milestone "disposal triggered", the augmented entity needs to send a disposal message to an employee. Since an employee is a human being, he or she cannot receive digital messages without any electronic devices. Because various studies predict that every person will have at least one internet enabled smart phone in the future[1], using these devices as a means of machine-to-human communication seems to be legitimate. Therefore, we define that employees, or responsible people across the whole product life-cycle in general, use their mobile phones to receive notifications about status changes of augmented entities. Figure 8 shows that only a single message is needed to notify the responsible employee, which has already been identified in the previous step.

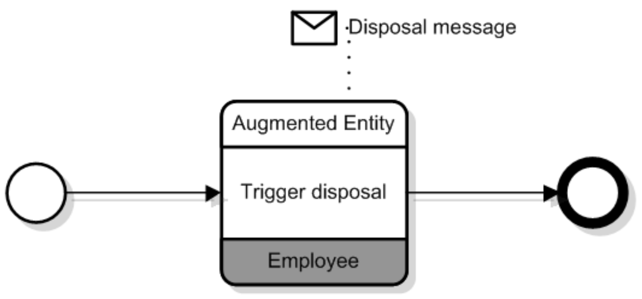


Figure 8 Sensor based quality control: choreography #1

Price update triggered. For reaching the state "price update triggered", a message transmitting the quality factor of the augmented entity (in this use case, it is the orchid) needs to be sent to the backend system. This system triggers then all further updates of the cashier systems and electronic shelf labels automatically. Therefore, the choreography as shown in Figure 9 can be kept very simplistic.

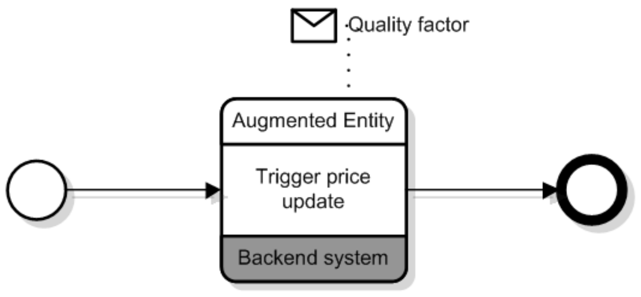


Figure 9 Sensor based quality control: choreography #2

Since all single choreographies have been defined, the message interfaces of the individual participants need to be specified. These specifications are called behavioural interfaces (Weske 2007). Behavioural interfaces, as (Weske 2007) states, "consider parts of process orchestrations that exhibit externally visible behaviour, e.g. communication activities and events that represent the sending or receiving of a message" (p. 238). The behavioural interface of the augmented entity can be seen in the conversation model as shown in Figure 10 and the interface of the employee as well as the backend system is shown in the conversation model in Figure 11. Both models neglect the fact that the service resolution infrastructure needs to be consulted in order to reduce the models complexity. However, each message exchange between collaboration partners (viz. orchid's process execution engine, backend system, and smart phones) requires a service resolution during run-time (cf. Figure 7).

[1] cf. eMarketer.com

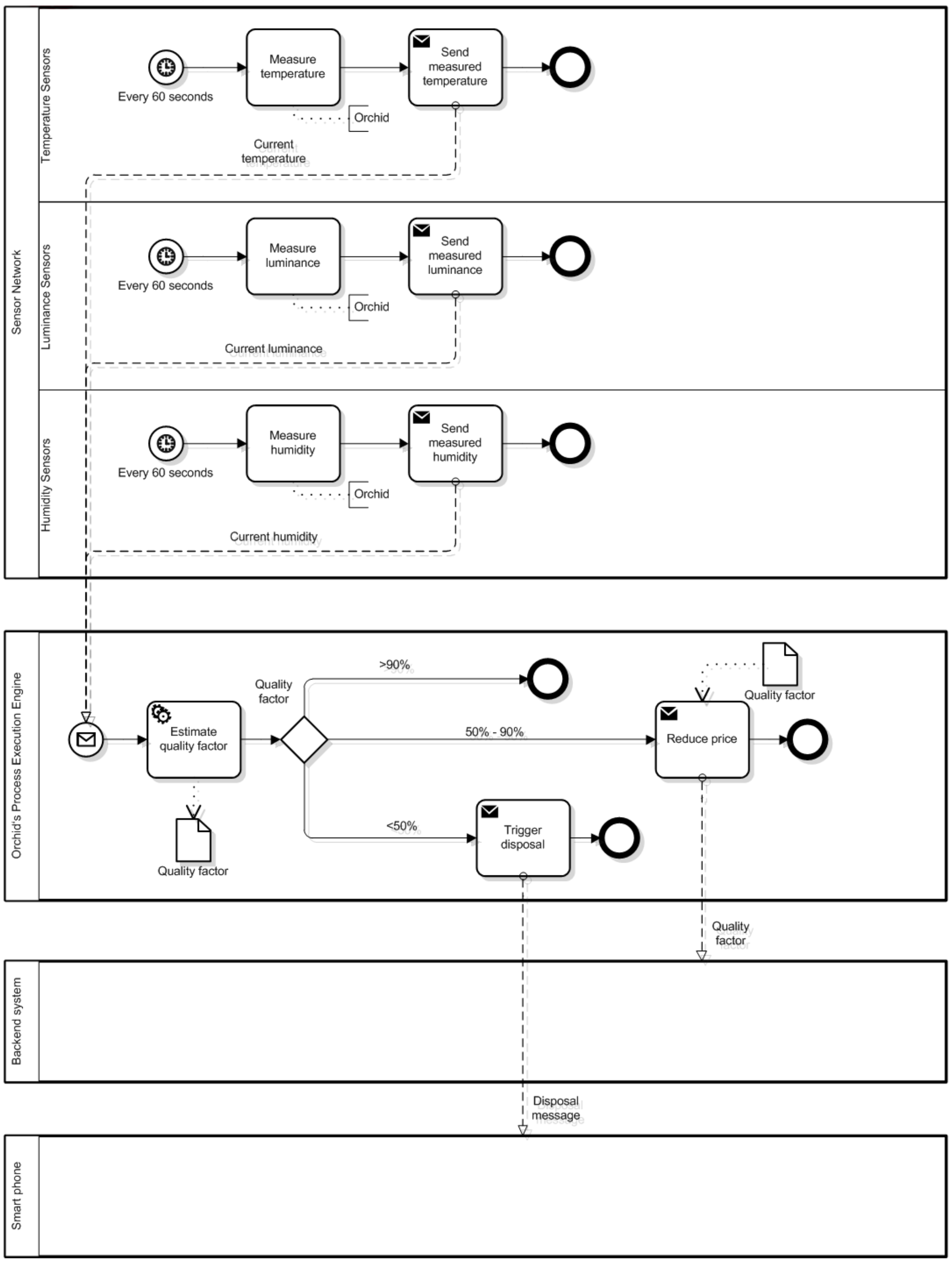


Figure 10 Sensor based quality control: conversation model #1

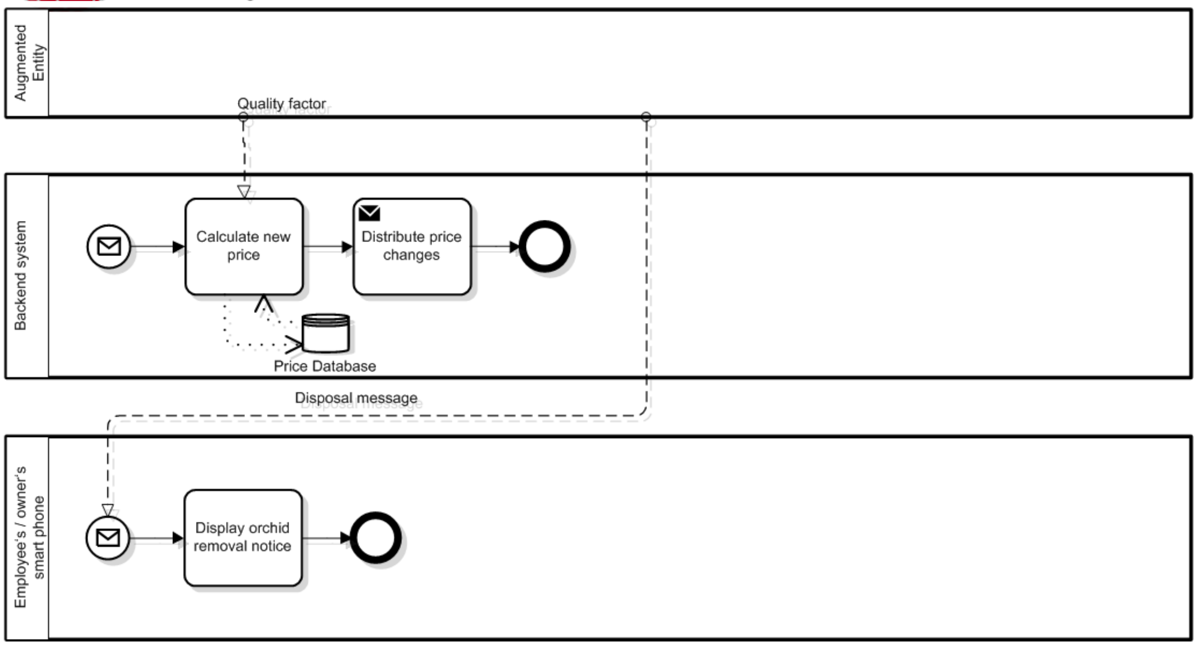


Figure 11 Sensor based quality control: conversation model #2

As it can be seen in the previous figures, besides the communication partner lookup mechanism, only two messages need to be transmitted to collaboration partners in order to enable the use case's IoT functionality across its whole life-cycle. Moreover, only one notification message needs to be sent if the life-cycle is not in the stage of the retail store where pricing information is linked to the product. In order to get an overview of the required message transmissions across the augmented entity's life-cycle, the following table has been developed.

Life-cycle stage	*Communication partner / device*	*Message*	*Message type*
Nursery	Employees (mobile)	Disposal message	Text-notification
Logistics network	Employees (mobile)	Disposal message	Text-notification
Retail store	Employees (mobile)	Disposal message	Text-notification
	Backend system	Quality factor	Quantifiable data
Customer's home	Customer (mobile)	Disposal message	Text-notification
Recycling company	-	-	-

Table 4 - Sensor based quality control: choreography messages

Table 4 identified that only two message types are required for the sensor based quality control use case: Text-notifications and quantifiable data. Since generic text-notifications do not require any processing by computer applications, the communication interface on the mobile phone can also be kept very generic, and enables the reuse of this functionality for other use cases. The message which contains the quality factor is a use case specific notification, which requires a specific implementation. Since price information needs to be adjusted individually anyways, the lack of a standard interface is not a big issue in this application.

2.5.4.2 Analysis

By taking a look at the conversation models in chapter 2.5.4.1, the following advantages and disadvantages can be identified:

Process participants	*Advantages*
Augmented entity (includes sensor network and process execution engine)	The augmented entity can sense the environmental data and calculate the resulting quality factor independently of its current stage in the life-cycle. The network will only be used in the case of a dramatic quality change. This approach increases the scalability of this use case significantly.
Backend system	The backend system is only notified and stressed if a quality exception occurred. This reduces the load of the backend system significantly, which allows to have more augmented entities being administrated by one single system.
Employee (mobile phone)	The employee has only one device which is communicating to him or her in order to notify about status changes in the process. The assumed availability of the mobile phone represents one single point of machine-to-human communication.

Table 5 - Sensor based quality control: advantages of choreography approach

Process participants	*Disadvantages*
Augmented entity (includes sensor network and process execution engine)	Since the augmented entity now also includes a process execution engine, more computing capabilities, more memory, and more application logic needs to be implemented in order to perform the tasks. Another disadvantage results from the integration of e.g. wireless sensor networks (WSNs), which desire for long lifetimes of batteries and, thus, result in low computing power, memory capacity, and network bandwidth.
Backend system	In order to communicate quality changes of the orchid to the backend system, a propriety interface needs to be present which a) offers the reception of a quality factor message, and b) knows how to process this information in a meaningful way. Since this functionality is not always guaranteed, quality changes which result in price changes can only be propagated when an appropriate backend system is available.
Employee (mobile phone)	The employee needs to own a mobile phone which is capable of opening up some communication channels to its immediate surroundings, and it needs to be able to receive and display generic notification messages.

Table 6 - Sensor based quality control: disadvantages of choreography approach

A comparison of the advantages and disadvantages of orchestration use case analysis in chapter 2.5.3.3 and the choreography use case shows that the previously identified disadvantages, namely the high network traffic, the dependency of a backend system, and the limited scalability have been eliminated successfully. The following table analyses how the redesign of the use case affected the present IoT characteristics (cf. (Meyer et al. 2011) and chapter 2.5.3.3).

IoT characteristics (Meyer et al., 2011)	*Findings*
Distributed execution	The execution of the business process is equally distributed among all process participants. Each participant has its defined role and responsibilities.
Distributed data	The data are stored in a distributed manner. In this case, information about the augmented entity's quality is directly on the augmented entity itself, whereas pricing information is stored in the ERP system, where the price data is required.
Scalability	Since the augmented entity only uses the network when a dramatic quality change occurred, the load of the network and the backend system is kept to a minimum which enables a vast amount of participating augmented entities.

Table 7 - Sensor based quality control: evaluation of IoT characteristics in choreography

Control flow patterns have been introduced to describe the control flow in process orchestrations. Process orchestrations and process choreographies, however, have various differences which need to be considered: choreographies base on message exchange between different participants, whereas orchestrations are based on the control flow between the activities of a single process (Weske 2007). For that reason, service interaction patterns have been introduced, which aim to be able to express standardised interactions and to benchmark modelling languages. The following table analyses the presence of possible service interaction patterns.

Pattern	*Partners*	*Messages*
Send	Augmented entity & backend system	Quality factor
Receive	Backend system & augmented entity	Quality factor
Send	Augmented entity & employee	Disposal message
Receive	Employee & augmented entity	Disposal message
Service resolution	Augmented entity & backend system	Standard service resolution messages
	Augmented entity & employee	

Table 8 - Sensor based quality control: service interaction patterns

The send and receive patterns are the only standard patterns (based on the research of (Weske 2007)) which have been identified. In addition to these, the service resolution pattern also needs to be applied to find the appropriate communication partners, as it is the case in any IoT use case.

2.5.4.3 Conclusion

Wrapping up, it can be said that the shift from a business process orchestration to a business process choreography eliminates most of the identified orchestration disadvantages (e.g. no flexibility in logic of augmented entity, limited scalability, etc.) of the sensor based quality control use case. The remaining disadvantage, namely the dependency of a backend system, has been mitigated to a minimum by only using the backend system when it is really required. All other previous backend system interactions have been eliminated by the distributed execution of the process. However, the shift to a choreography approach also yielded in some new disadvantages which have to be considered when designing IoT use cases. As a consequence, the augmented entities must have more computing capabilities, more memory, and more application logic implemented in order to execute more demanding business tasks. A major challenge is, therefore, the integration of e.g. wireless sensor networks (WSNs), which desire for long lifetimes of batteries and, thus, result in low computing power, memory capacity, and network bandwidth, into its environment (Glombitza, Pfisterer, and Fischer 2010). Reasons for that are the missing technological innovations which would enable a seamless implementation and integration of service oriented architectures (SOA) based on current web service standards, as well as the execution of business processes by the means of business process execution engines on the sensor devices itself. Even though the technology for that exists, from a practical and economical point of view it would be unwise to currently do so due to high power consumptions, comparably large physical dimensions, and high monetary costs.

However, this analysis of advantages and disadvantages did not consider the fact that an omnipresent service resolution infrastructure is required. This leads to the result that in theory, the SRI is an orchestration approach for allocating services to augmented entities during run-time. Therefore, the overall IoT use case setup is a hybrid version of the choreography and orchestration approaches, whereas the orchestration based SRI is perceived as a global system, which is essential for the future internet. (I. J. G. D. Santos and Madeira 2006) also stated in their discussion about orchestration versus choreography that “real scenarios involving complex systems with multipart interactions demand both approaches” (p. 3).

2.5.5 Requirements

The previous use case analysis revealed some new requirements for enabling business process and service choreographies in the Internet of Things. The reason for that are the typical IoT characteristics themselves. Primarily the mobility factor of augmented entities leads to a list of prerequisites for enabling choreographies. Mobility, as (Meyer et al. 2011) analysed in their paper, is the characteristic that the availability of communication partners cannot be guaranteed in the Internet of Things. Usually, communication partners are not known at modelling time, but will be required at run time. In order to cope with these challenges, the following requirements need to be fulfilled:

Requirement	*Description*
More powerful augmented entities	The shift to business process and service choreographies requires the augmented entities to have more computing capabilities, more memory, and more application logic implemented in order to execute more demanding business tasks. A major challenge is, therefore, the integration of e.g. wireless sensor networks (WSNs), which desire for long lifetimes on batteries and, thus, result in low computing power, memory capacity, and network bandwidth, into its environment (Glombitza, Pfisterer, and Fischer 2010).
Service Resolution Infrastructure	Due to the mobile nature of augmented entities, the lookup of a communication partner which is geographically and logically the right partner to communicate need to be available at run-time. The SRI needs to offer the following functionalities: a) Registration of augmented entities at the time when they go online, considering properties like location, capabilities, and privacy settings. b) Receiving requests for locating and reporting appropriate communication partners as services. c) Composition of services, to a higher level service which can be used by other augmented entities.
Modelling mobile augmented entities	Service providing augmented entities and systems need a way to tell if their services are permanently available (static service), or if their availability is depending on their status, location, and privacy settings

	(mobile service). a) Examples for static services are all interfaces to the SRI and today's well-known web-services. These services are characterized by the lack of necessity to contact a service lookup and discovery provider, as the SRI is. b) Mobile services are all the features which mobile augmented entities provide to the outside world. An example of such a service is the interface to a temperature sensor of a device.
Modelling run-time service discovery	When modelling a process for an augmented entity which requires e.g. a temperature sensor in close proximity, the exact communication address of the sensor is not known during modelling. Therefore, this consuming service needs to be modelled in a way which incorporates the service resolution infrastructure. Ideally, a new BPMN standard element contains the lookup and discovery tasks, in order to avoid the repetitive modelling of the service discovery pattern.
Modelling the service resolution infrastructure registration	Most of the augmented entities require registering themselves in the service resolution infrastructure. In order to avoid the repetitive work of modelling, another standard element needs to be developed, which contains the registration process at the beginning of each entities active life-cycle.
Defining standard messaging services	One essential part in machine-to-human communication is the standardization of notification message exchanges. As seen in the sensor based quality control use case, the disposal message of the orchid was sent to the employee's or owner's mobile phone. However, this phone does not need a specified disposal-message-interface, but rather a generic machine-to-human text-based notification. Therefore, such a standardized interface for each human interaction device, e.g. a mobile phone, is hereby suggested.
Modelling communication protocol dependency	Not all process participants are always using the standard internet protocol for communicating with other participants. In some cases, information about augmented entities is collected by reading attached RFID tags, which are mostly passive devices. For this reason, the lookup of communication partners must distinguish between internet-enabled devices, which are communicating via web services, or passive devices, which need to be scanned in order to get information (e.g. RFID tags, bar codes, etc.).

Table 9 - List of requirements

2.6 Conclusion and Recommendation

The previous chapters have shown that current IoT processes usually follow the orchestration paradigm where a central coordination system is place. By considering the whole life-cycle of the augmented entity, this approach is hindering the application of the remarkable IoT functionality outside this protected environment. A use case analysis showed that a move from the orchestration to the choreography paradigm would result in several benefits. Firstly, more IoT characteristics such as distributed execution, distributed data, and scalability can be obeyed. Secondly, this approach enables the presence of the IoT functionality across the whole augmented entity's life-cycle. Finally, the omnipresence of IoT enabled entities, which are ready to communicate over the internet protocol, encourage other vendors to design processes and applications which require the Internet of Things in order to create a smarter environment.

However, the move to business process and service choreography is not fully ready yet. For enabling this paradigm shift, various identified requirements need to be fulfilled and implemented first. The biggest influencing factor is the mobile nature of augmented IoT entities. This requires for example new architectural elements such as the service resolution infrastructure, as well as ways to model mobile augmented entities, which require the SRI for looking up communication partners during run-time. Another factor is the requirement for more powerful devices, viz. having more computing capabilities, more memory, and more application logic implemented, in order to reduce the need for communication with the backend system. However, this required enhancement of augmented entities is the currently one of the biggest limitations for the choreography approach. Today's hardware vendors already offer the required technology, but from a practical and economical point of view, it is almost impossible to equip each and every augmented entity with such high computing capabilities, because of costs, unmanageable form factor, and energy consumption aspects. There has been the same discussion about a decade ago with regards to RFID tags and nowadays, this discussion does not really exist anymore due to affordable RFID tags it can be assumed that the same applies for smart devices for augmented entities.

Summarized it can be said that the implementation of the requirements is only advisable if the underlying use case is demanding the gained choreography functionalities. Otherwise, the existing orchestration approach should be preferred. Therefore, the following chapters in this deliverable focus only on service orchestrations. However, further research in the field of business process and service choreography is highly recommended.

3. Existing service orchestration Concepts

The following chapter gives a general introduction into different existing modelling approaches. It gives an overview about existing approaches of the last years depicted within a timeframe.

3.1 BPMN orchestration engines

This section gives an overview about BPMN orchestration engines. It starts with a historical overview of business process execution followed by a presentation of current business process execution engines.

3.1.1 Historical overview of business process execution

A historical overview of business process modelling in general is presented in chapter 2.2 of D2.2 (Meyer et al. 2012). Here we describe this history with regard to the execution of business processes. Figure 12 depicts the evolution of the most important business process modelling languages. The history of the Business Process Modeling Notation respectively Business Process Model and Notation (BPMN) is presented in “BPMN Modeling and Reference Guide” written by Stephen A. White and Derek Miers (White and Miers 2008) and in “BPMN Method and Style” written by Bruce Silver (Silver 2011a).

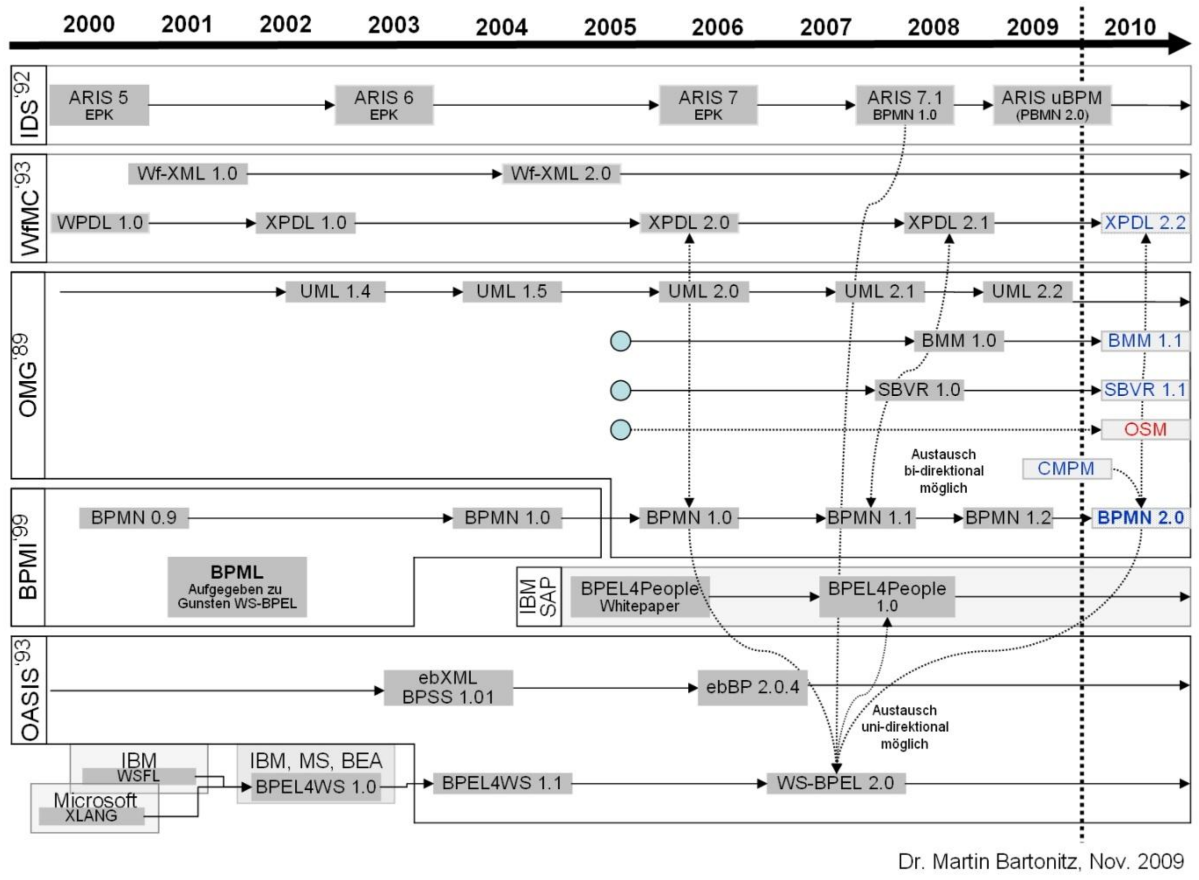


Figure 12 Historic evolvement of Business Process Modelling Languages (Bartonitz 2009)

In 2001 the “Business Process Management Initiative” (BPMI) started the development of an XML language for the execution of business processes. As the “Unified Modeling Language” (UML) was considered to be too technical for the graphical modelling of business processes, the BPMI also developed a specific XML language for this purpose. The execution language was called “Business Process Modeling Language” (BPML), and the modelling language was the first version of the “Business Process Modeling Notation” (BPMN). There was no direct technical connection between these two languages, so it was impossible to execute the graphical models directly, but a transformation was needed in between.

Although the BPMI consortium consisted of up to 200 companies, their standard BPML was superseded by the “Business Process Execution Language” (BPEL) in 2003. BPEL was developed by Microsoft and IBM and handed over to OASIS with version 1.1. In 2005 the BPMI was merged into the “Object Management Group” (OMG), and consequently the BPMN became an OMG standard in 2006. The OMG issued two minor releases BPMN 1.1 in 2008 and BPMN 1.2 in 2009, with which bugs were fixed and inaccuracies were clarified.

Because BPEL is based on the “Web Services Description Language” (WSDL), the exact name of the versions 1.0 and 1.1 was “Business Process Execution Language for Web Services” (BPEL4WS); version 2.0, which was released in 2007, was renamed to “Web Services Business Process Execution Language” (WS-BPEL). Since the emergence of BPEL, many tool vendors implemented execution engines, which are capable of executing BPEL processes. This includes the major software companies like IBM, Microsoft, Oracle, and SAP, as well as open source software packages from e.g. the Apache Software Foundation and jBoss (Wikipedia 2012).

The modelling language BPMN 1.x and the execution WS-BPEL coexisted most prominently for a couple of years. Therefore, a strong need for executing the BPMN models in WS-BPEL execution engines evolved over time. Due to the fact, that BPMN is graph-oriented and WS-BPEL is block-structured, a direct mapping from BPMN to WS-BPEL is not possible, and subsequently much research effort has been made to overcome this problem, but no satisfying solution for all circumstances has been found (Freund, Rücker, and Henninger 2010).

Since BPEL is based on WSDL, it is designed to orchestrate processes, which are built of web services only. In 2007 a consortium of industry companies, including IBM, Oracle, and SAP, specified WS-HumanTask and BPEL4People as extensions to BPEL, adding support for human tasks to processes, which are executed in BPEL compliant engines (Active Endpoints et al. 2007). ActiveEndpoints ActiveVOS (Active Endpoints 2012) allows the usage of human tasks, and Apache HISE (Apache Incubator Project 2012) is an open source implementation of WS-HumanTask, which can expose such tasks as web services.

In 1998 the “Workflow Management Coalition” (WfMC) published the “Workflow Process Definition Language“ (WPDL), which was renamed to “XML Process Definition Language“ (XPDL) with the release of the next version in 2002. This language facilitates the interchange of process models between different modelling tools. The latest version 2.1 was released in 2008 and covers all language constructs, which are defined in BPMN 1.1. XPDL is used by more than 80 tools as a common interchange format (Workflow Management Coalition 2012).

In the beginning of 2011 the OMG finally released the latest version 2.0 of the BPMN; the acronym now stands for “Business Process Model and Notation”. This is the first version of BPMN, which is not limited to the definition of graphical modelling, but also adds execution semantics and an XML serialisation format.

Bottom line: Over the last decade, many different languages in the area of graphical business process modelling, business process execution, and the interchange of business process models existed. Unfortunately, there was no approach, which smoothly integrated these different areas. With the finalisation of BPMN 2.0, this obstacle has been overcome, and it provides a holistic approach for these three subtopics. BPMN has become *the* important standard in BPM; any other notation is seen as “proprietary” or “legacy” now (Silver 2011a). Therefore, we concentrate on tools, which are based on BPMN 2.0, in the following sections.

3.1.2 Current business process execution engines

The OMG maintains a list of tools, which implement BPMN, on their website (Object Management Group 2012). In January 2012 this list contains 73 entries. Unfortunately, the degree of detail is very uneven between the entries of this list. While most of them are made of a link to the website of the tool vendor only, some consist of longer paragraphs explaining the nature of the tool. Most of these explained tools are for modelling business processes, only few are capable of executing processes. Additionally, it is often unclear, which version of BPMN is used by the tools. Therefore, this list does not provide the best possible basis for the comparison of BPMN 2.0 execution engines.

In his Blog entry written on November 7th 2011 Bruce Silver states that open source tools are ahead of commercial tools in implementing executable BPMN 2.0. With the term “executable BPMN 2.0” he refers to tools that use BPMN 2.0 XML for the serialisation of the models and transform those models into an internal representation, with which the execution is performed. The open source tools he mentions are Activiti, Bonita Open Solution, and with restriction jBPM. From the commercial tools he mentions SAP NetWeaver Process Integration, and expects IBM Process Designer and Oracle BPM to implement BPMN 2.0 in the near future (Silver 2011b).

Before the open source tools Activiti, Bonita Open Solution, and jBPM are presented in detail, we give a short overview of the general principle of business process execution.

3.1.2.1 General Principle of Business Process Execution

A business process model is a graphical representation of a business process. Such a model forms a graph, which nodes can be grouped into the three categories events, gateways and activities. The edges, which connect these nodes, are the process flow transitions between the events, gateways and activities. As these transitions are always directed, the process model forms a directed graph.

During the execution of such a process model, one or more tokens flow through the model visiting the events, gateways and activities along the transitions. If the process contains an untyped start event, the first token is generated there, when the execution is started. If a special kind of start event is used, the first token is generated, when the condition corresponding to this special event type is met. When a token visits an activity during its flow through the process model, the activity is executed. When a token visits a diverging exclusive gateway, the token continues along exactly one process flow transition. At a converging exclusive gateway exactly one token is needed to continue the process flow. When a token visits a diverging parallel gateway, multiple tokens are generated for every outgoing process flow transition. At a converging parallel gateway every incoming token must wait until tokens from all incoming process flow transitions arrive, before one token continues along the outgoing process flow transition. At the end event of the process the arriving token is consumed and the process execution finishes.

3.1.2.2 Activiti

The development of Activiti (Activiti 2012) started in 2010, when Tom Baeyens and Joram Barrez leaved the jBPM project and joined Alfresco to build a new BPM workflow engine. This engine Activiti was based on BPMN 2.0 from the beginning. As they were involved in the development of jBPM up to version 4, they started counting the versions numbers of Activiti from 5.0. The first alpha version was released in May 2010, and the release 5.0 appeared in December 2010. In January 2012 the current release is 5.8 from October 2011.

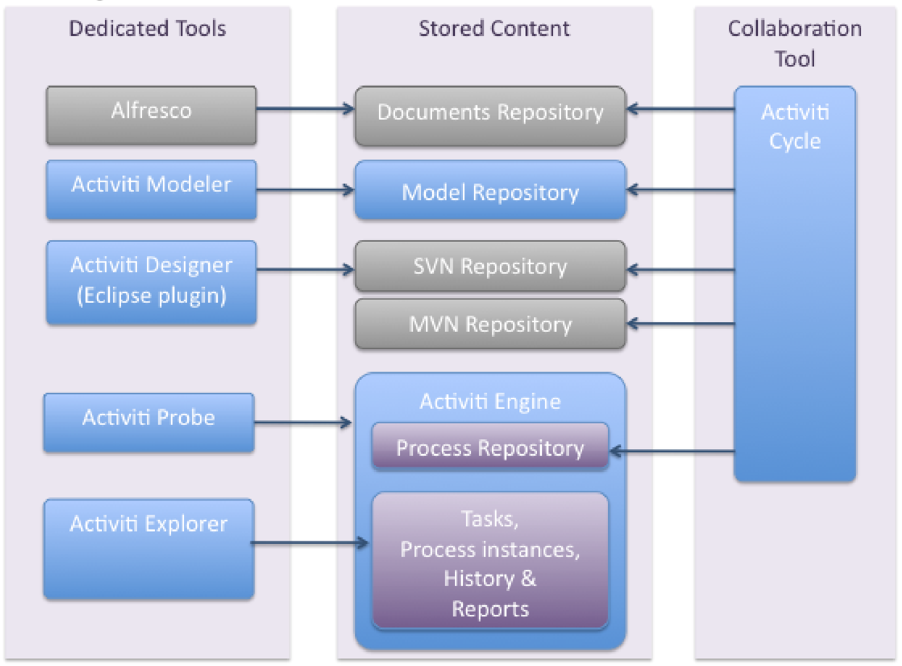


Figure 13 Activiti Components (Activiti 2012)

The overall structure of the Activiti software is shown in Figure 13. The components shown in blue make up the software package, and the components coloured grey are declared to be out of scope for Activiti. For the modelling of business processes Activiti provides two different environments: The Activiti Modeler is a customized version of the Signavio Process Editor (Signavio 2012), which stores the models in BPMN 2.0 XML serialisation directly, while the Activiti Designer is a plugin for the Eclipse IDE (Eclipse Foundation 2012). The Activiti User Guide suggests to use the Modeler for modelling at business level and to use the Designer for refining the model at technical level. The Activiti Engine is the core component of Activiti; it is responsible for the execution of the business processes. The functioning of the engine is explained in detail further down. The Model Repository is used to store the deployed business process models internally. The Activiti Explorer is a web-based interface to the engine, which enables the user to interact with it: He can list the tasks which are currently assigned to him, or which he can take over. If a task is assigned to him, he can complete it or reassign it to another user of the engine. If the user is the owner of a process, he can display additional information about it. Activiti Probe is the management interface to the engine. This is also a web-application, with which the administrator of the engine can monitor its state, and manage the deployment of processes, and review the logs. Activiti Cycle is a web-based tool, which eases the collaboration between business users, developers and IT staff (Activiti 2012).

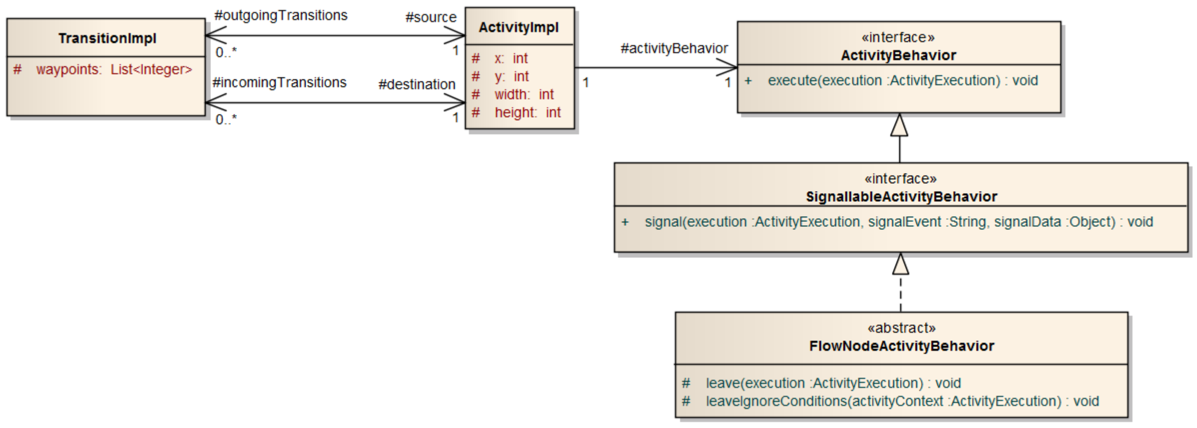
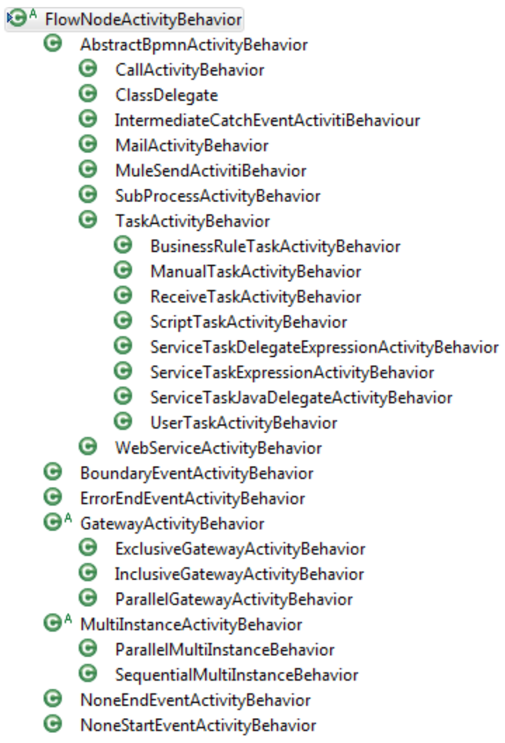

TransitionImpl
# waypoints: List<Integer>
#outgoingTransitions
#source
0..*
1
#incomingTransitions
#destination
0..*
1
ActivityImpl
# x: int
# y: int
# width: int
# height: int
#activityBehavior
1
1
«interface»
ActivityBehavior
+ execute(execution :ActivityExecution) : void
«interface»
SignallableActivityBehavior
+ signal(execution :ActivityExecution, signalEvent :String, signalData :Object) : void
«abstract»
FlowNodeActivityBehavior
# leave(execution :ActivityExecution) : void
# leaveIgnoreConditions(activityContext :ActivityExecution) : void

Figure 14 Representation of BPMN models within Activiti

In order to execute a business process the Activiti Engine parses the corresponding BPMN 2.0 model from its XML serialisation into the engines' internal object structure, which is depicted in Figure 14. For every node in the graph of the process model one ActivityImpl is instantiated, and a TransitionImpl is instantiated for every edge in the graph. According to the structure of the graph, each ActivityImpl holds two Lists of TransitionImpl objects for the incoming and outgoing transitions. Vice versa every TransitionImpl holds to references of ActivityImpl objects for the source and the destination of the connector in the model, which it represents. To keep track of the geometrical relations between the nodes and edges their coordinates are stored in instance member fields of the two classes.



```
FlowNodeActivityBehavior
    AbstractBpmnActivityBehavior
        CallActivityBehavior
        ClassDelegate
        IntermediateCatchEventActivitiBehaviour
        MailActivityBehavior
        MuleSendActivitiBehavior
        SubProcessActivityBehavior
        TaskActivityBehavior
            BusinessRuleTaskActivityBehavior
            ManualTaskActivityBehavior
            ReceiveTaskActivityBehavior
            ScriptTaskActivityBehavior
            ServiceTaskDelegateExpressionActivityBehavior
            ServiceTaskExpressionActivityBehavior
            ServiceTaskJavaDelegateActivityBehavior
            UserTaskActivityBehavior
        WebServiceActivityBehavior
    BoundaryEventActivityBehavior
    ErrorEndEventActivityBehavior
    GatewayActivityBehavior
        ExclusiveGatewayActivityBehavior
        InclusiveGatewayActivityBehavior
        ParallelGatewayActivityBehavior
    MultiInstanceActivityBehavior
        ParallelMultiInstanceBehavior
        SequentialMultiInstanceBehavior
    NoneEndEventActivityBehavior
    NoneStartEventActivityBehavior
```

Figure 15 Class Hierarchy of FlowNodeActivityBehaviour

Every ActivityImpl instance holds a reference of one subclass of ActivityBehavior. The abstract class FlowNodeActivityBehavior is the Activiti-internal counterpart of the FlowNode element in the BPMN 2.0 meta-model (Figure 8.35 in (Object Management Group 2011)). Depending on the concrete element in the parsed BPMN 2.0 model, any of the subclasses of FlowNodeActivityBehavior is instantiated to represent the element internally. The class hierarchy below FlowNodeActivityBehavior is shown in Figure 15. For the different types of tasks, which are shown in the BPMN 2.0 specification (Figure 10.10 in (Object Management Group 2011)), the Activiti program provides a specific subclasses of TaskActivityBehavior. Accordingly, there are Classes for the representation of the other elements, which can occur in a BPMN 2.0 model: Events like start and end event, intermediate events, boundary events, and error events; exclusive, inclusive and parallel gateways; parallel and sequential multi-instance marker; sub-processes and call activities.

During the execution of the process the engine calls the methods execute and leave respectively signal on the different instances of the subclasses of FlowNodeActivityBehavior. The execute method defines, which steps need to be performed within the FlowNode, and the leave method defines the desired behaviour when the FlowNode is left. In the signal method it is specified, if any actions need to be taken, if an event occurs in the current state; e. g. the ReceiveTaskActivityBehavior simply calls the leave method, when the signal method is called, and so the process execution continues.

3.1.2.3 Bonita Open Solution

The development of Bonita Open Solution was initialised by Miguel Valdés Faura within the French National Institute for Research in Computer Science and Control (INRIA) in 2001 (Bonitasoft 2012). For some years the development was performed within the French company Bull. In 2009 the company BonitaSoft was especially founded for the further development of the software. In January 2012 the current release is 5.6.1 from December 2011.

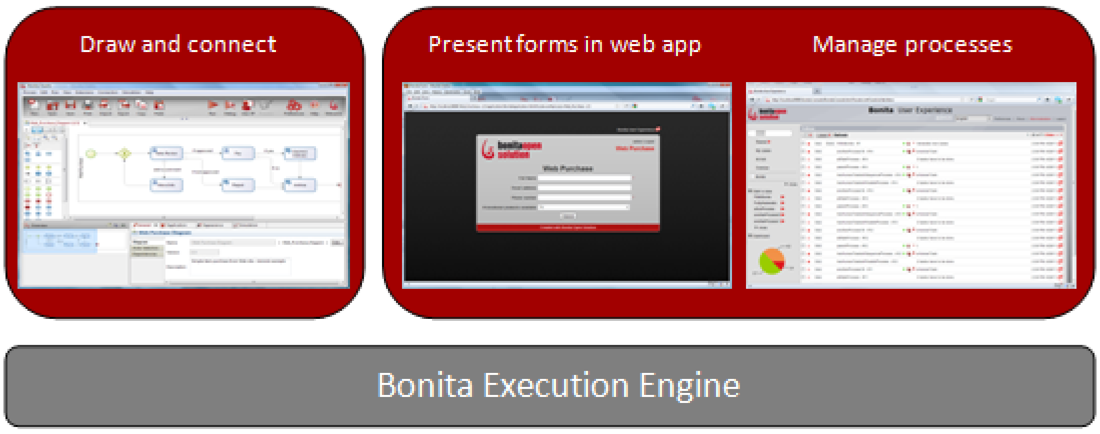


Figure 16 Bonita Open Solution Components (Bonitasoft 2012)

Bonita Open Solution consists of the following parts, which are shown in Figure 16: Bonita Studio provides the graphical user interface, with which the users can model their processes. The editor is based on eclipse and can export the models in a BPMN 2.0 compliant serialisation. In order to specify the interaction between the modelled process and external tools like databases, ERP systems, or web platforms Bonita Studio provides so-called ready-to-use connectors. With specific wizards the user can configure the connections to the systems he wants to use with his process. Additionally, the creation of new connectors for custom tools is possible. At last Bonita Studio facilitates the design of custom forms, which are presented to the user for the execution of manual tasks. The Bonita User Experience is the web-based frontend during the execution of processes. For the normal users this is the UI, via which they interact with the process: They see which steps need to be performed and they enter their data into the designed forms. The administrating users can manage the deployed processes and review logs and reports within the Bonita User Experience. The Bonita Execution Engine is the system in the back, which actually runs the deployed processes and controls the Bonita User Experience.

3.1.2.4 jBPM

jBPM (jBoss Community 2012) is a business process management suite, which is developed within the jBoss community. Version 1.1 was released back in November 2007, and therefore it was based on BPEL. The last iterations of the major versions 3 and 4 were 3.2.7 (released in September 2009) and 4.4 (released in July 2010). These versions used a proprietary language for the definition of processes, which was named "jBPM Process Definition Language" (JPDL). Starting from Version 5.0 jBPM is natively based on BPMN 2.0; this version was released in February 2011. In January 2012 the current release is 5.2 from December 2011.

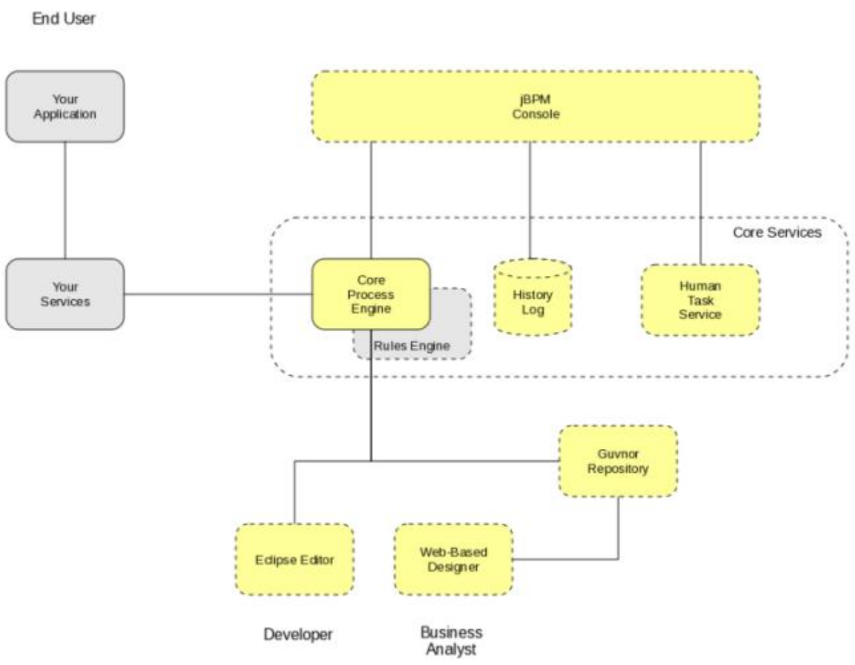


Figure 17 Overview of jBPM Components (jBoss Community 2012)

Figure 17 shows the components, which are comprised in jBPM or related to it. The Core Process Engine is the central part of the system; its deployment is mandatory, but all other shown components are optional. The Core Process Engine is a general process engine, which can execute processes modelled in BPMN 2.0 by default, but it is not limited to this format. The engine can run standalone, or it can be integrated into any Java software. Furthermore, it is possible to integrate it with Java persistence and Java transaction implementations. If it is needed the process engine can interact with a running Rules Engine, e. g. jBoss Drools Expert. Additionally, the History Log can log all historical data about the running processes for later analyses. If the modelled processes include human tasks, the Human Task Service can be used to enable the interaction of human users with the process engine. For the editing of business processes jBPM provides two different tools. The Web-Based Designer is suitable for modelling at business level and targets business analysts. Technologically it is based on the Signavio Process Editor (Signavio 2012). The Designer is integrated with the Guvnor Repository, which stores the created process models. This repository facilitates collaborative modelling by multiple users, and supports versioning of process models. It also has a web interface, into which the Web-Based Designer is incorporated. The Eclipse Editor is a custom plugin for the Eclipse IDE (Eclipse Foundation 2012), which provides modelling facilities at a technical level and targets developers. The jBPM Console is also a web interface, with which the users can interact with the process engine in the following ways: He can start a new instance of a process, review a list of all running processes, and inspect the current state of one running process. For this functionality a visual representation of the process is displayed to the user and the activity, which is currently executed, is highlighted. If the History Log is enabled, the jBPM Console provides the user interface, with which the user can review the logs, query the log database, and perform analyses on the logs. Analogously, if the Human Task Service runs, the jBPM Console is the interface via which the users interact with the process engine to fulfil their tasks; especially the task lists and input forms for user data are displayed here.

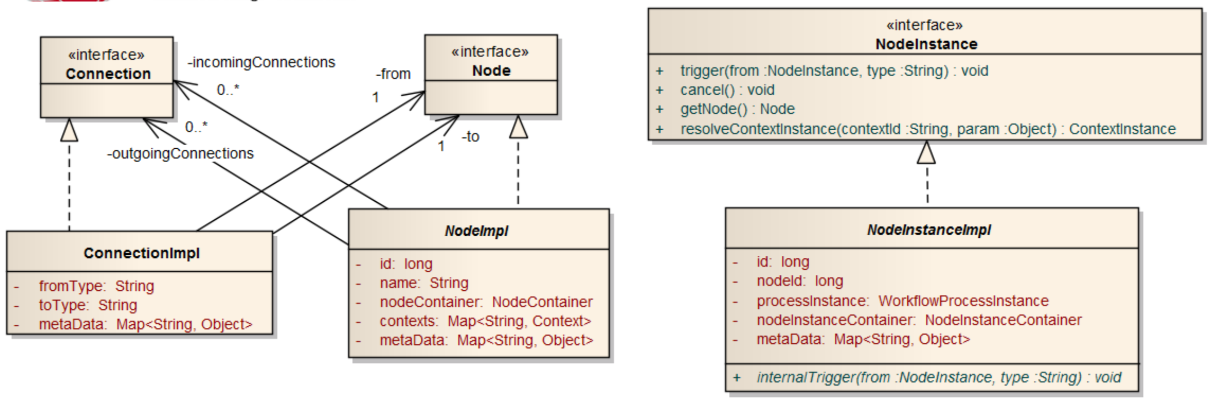
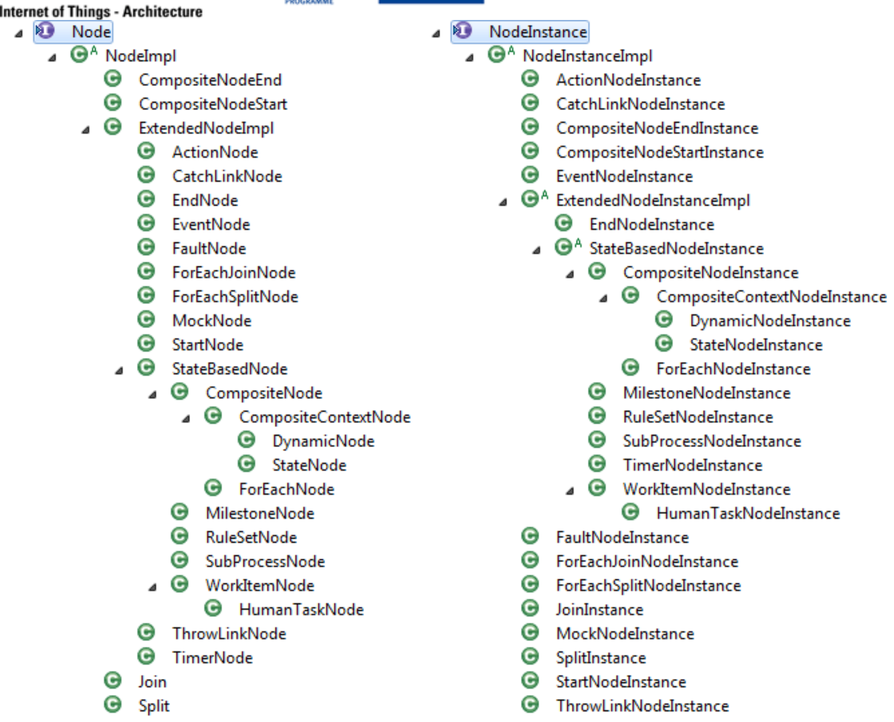

«interface»
Connection
-incomingConnections
0..*
-from
1
«interface»
Node
-to
1
0..*
-outgoingConnections
ConnectionImpl
- fromType: String
- toType: String
- metaData: Map<String, Object>
NodeImpl
- id: long
- name: String
- nodeContainer: NodeContainer
- contexts: Map<String, Context>
- metaData: Map<String, Object>
«interface»
NodeInstance
+ trigger(from :NodeInstance, type :String) : void
+ cancel() : void
+ getNode() : Node
+ resolveContextInstance(contextId :String, param :Object) : ContextInstance
NodeInstanceImpl
- id: long
- nodeId: long
- processInstance: WorkflowProcessInstance
- nodeInstanceContainer: NodeInstanceContainer
- metaData: Map<String, Object>
+ internalTrigger(from :NodeInstance, type :String) : void

Figure 18 Representation of process models in jBPM

In Figure 18 the internal representation of process models within the jBPM software package is depicted. When a BPMN 2.0 process model is loaded into jBPM it is parsed into the structure, which is shown in the left side of the figure. For every node in the model an instance of a subclass of NodeImpl is created. Accordingly, for every edge in the process model an instance of ConnectionImpl is created. Every NodeImpl instance holds references to all incoming and outgoing connections, and every ConnectionImpl instance holds references to the node it comes from and to the node it goes to. The name and id of the nodes are stored in the NodeImpl attributes name and id. If a node is contained within another structural element in the model (like e. g. nested sub-processes are), this container is referenced with the nodeContainer variable. The field contexts in NodeImpl is used to reference contexts like variable scopes, exception scopes or swimlane contexts. Additional information from the BPMN 2.0 process model like the graphical layout of the nodes and edges are stored in the metaData fields of NodeImpl and ConnectionImpl. When a parsed process is instantiated in order to execute it, an instance of a subclass of NodeInstanceImpl is created for every NodeImpl instance, which is part of the process representation. The connection between an instance of a subclass of NodeInstanceImpl and an instance of a subclass of NodeImpl is not implemented by a reference, but the former stores the id of the latter in the nodeId variable. If this referenced object is part of a NodeContainer, then the referencing object is included in a NodeInstanceContainer accordingly. With the processInstance variable the WorkflowProcessInstance is referenced, to which the object belongs. When the process execution engine executes the node instance, it calls the trigger method on it. The cancel method is called to deactivate a NodeInstance for the remainder of the process execution.



```
Node
  NodeImpl
    CompositeNodeEnd
    CompositeNodeStart
    ExtendedNodeImpl
      ActionNode
      CatchLinkNode
      EndNode
      EventNode
      FaultNode
      ForEachJoinNode
      ForEachSplitNode
      MockNode
      StartNode
      StateBasedNode
        CompositeNode
          CompositeContextNode
            DynamicNode
            StateNode
          ForEachNode
        MilestoneNode
        RuleSetNode
        SubProcessNode
        WorkItemNode
          HumanTaskNode
      ThrowLinkNode
      TimerNode
    Join
    Split
```

```
NodeInstance
  NodeInstanceImpl
    ActionNodeInstance
    CatchLinkNodeInstance
    CompositeNodeEndInstance
    CompositeNodeStartInstance
    EventNodeInstance
    ExtendedNodeInstanceImpl
      EndNodeInstance
      StateBasedNodeInstance
        CompositeNodeInstance
          CompositeContextNodeInstance
            DynamicNodeInstance
            StateNodeInstance
          ForEachNodeInstance
        MilestoneNodeInstance
        RuleSetNodeInstance
        SubProcessNodeInstance
        TimerNodeInstance
        WorkItemNodeInstance
          HumanTaskNodeInstance
    FaultNodeInstance
    ForEachJoinNodeInstance
    ForEachSplitNodeInstance
    JoinInstance
    MockNodeInstance
    SplitInstance
    StartNodeInstance
    ThrowLinkNodeInstance
```

Figure 19 Class Hierarchies of Node and NodeInstance

Figure 19 shows the class hierarchies of the interfaces Node and NodeInstance. The instantiation of the concrete subclasses depends on the types of the elements in the BPMN 2.0 model. Although jBPM claims to execute BPMN 2.0 process models natively, the class hierarchy is not clearly aligned with the meta-model of BPMN 2.0.

3.2 Web service composition systems

This section discusses existing Web service orchestration methods and their relationship to process execution systems as presented before. In the Service Oriented Computing (SOC) paradigm individual application components (possibly directly accessible via the Web) are being flexibly assembled into complex processes. The underlying technology on which SOC is often assumed to be based is the technology of Web Services (Grossmann et al. 2011). Web services use Web-based communication protocols and XML-based data formats and communication interface definitions (W3C 2004).

A widely used standard for web service-based orchestration of business processes is the Web Services Business Process Execution Language Version 2.0 (WS-BPEL 2.0)[2] formerly named BPEL4WS. The standard „defines a language for specifying business process behaviour based on Web Services." WS-BPEL is designed to specify executable business processes as well as abstract business processes. Abstract business processes are partially specified processes that are not intended to be executed, whereas executable business processes specify the actual behaviour of a participant in a business interaction.

Although web-based techniques are widely used and well-understood by service consumers as well as service providers an interoperability problem still remains. Nowadays it is still a manual task to integrate Web services with Business Process Execution models. For automated service orchestration as intended to be applied in IoT-A the Web services need to be annotated with additional information that describe the nature and purpose and other details of the services as described in IoT-A deliverable D2.1. IoT-A has chosen Linked USDL[3] as service description language. The Universal

[2] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[3] http://www.linked-usdl.org/

Service Description Language (USDL)[4] is the outcome of several research projects and is currently under standardisation. The language is platform independent and can be applied for describing WSDL as well as RESTful web services. To address interoperability between service consumers and service providers, Linked USDL is realised with Semantic Web[5] techniques. Using Semantic Web style service descriptions forces more requirements onto service orchestration and service composition systems. A lifecycle for dynamic composition of services is presented in (Grossmann et al. 2011)

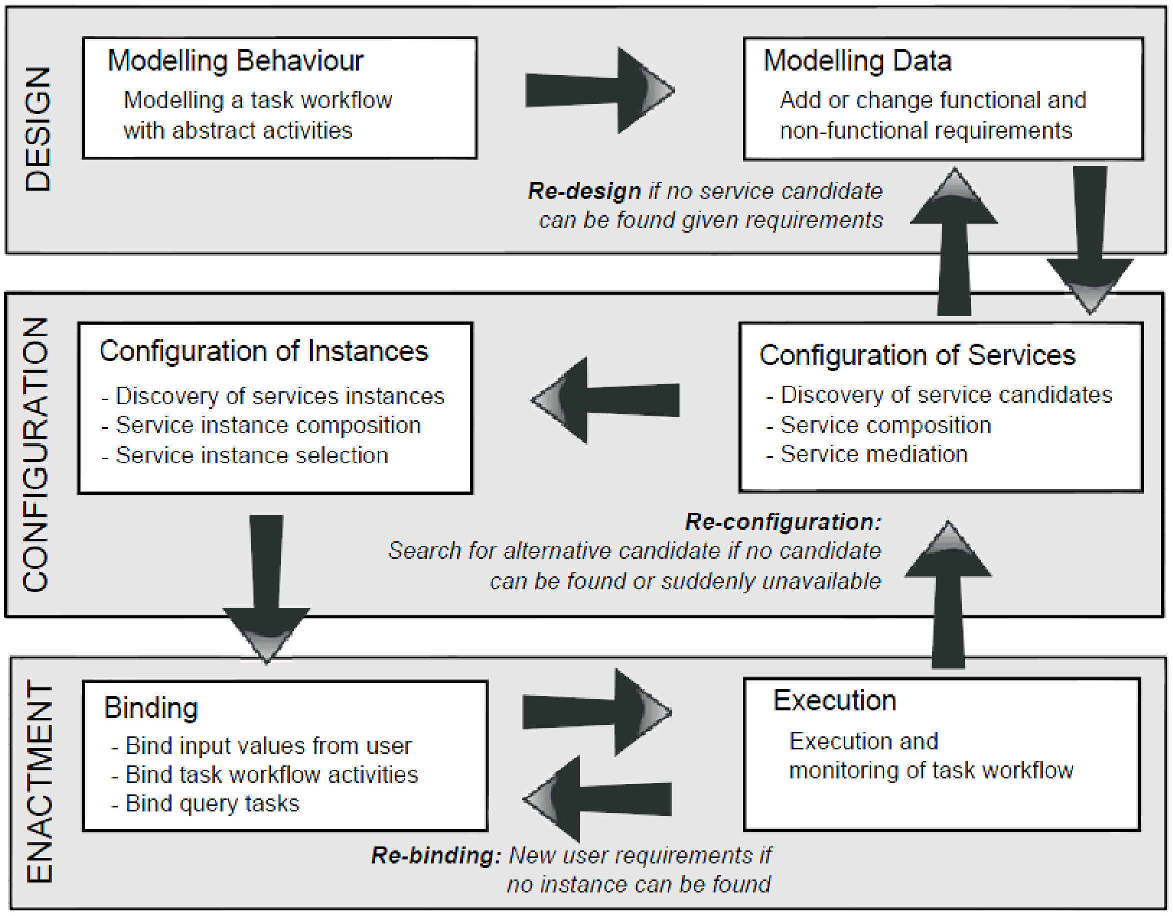

DESIGN
Modelling Behaviour
Modelling a task workflow with abstract activities
Modelling Data
Add or change functional and non-functional requirements
Re-design if no service candidate can be found given requirements
CONFIGURATION
Configuration of Instances
- Discovery of services instances
- Service instance composition
- Service instance selection
Configuration of Services
- Discovery of service candidates
- Service composition
- Service mediation
Re-configuration: Search for alternative candidate if no candidate can be found or suddenly unavailable
ENACTMENT
Binding
- Bind input values from user
- Bind task workflow activities
- Bind query tasks
Execution
Execution and monitoring of task workflow
Re-binding: New user requirements if no instance can be found

Figure 20 Life-cycle for the dynamic composition of services (Grossmann et al. 2011)

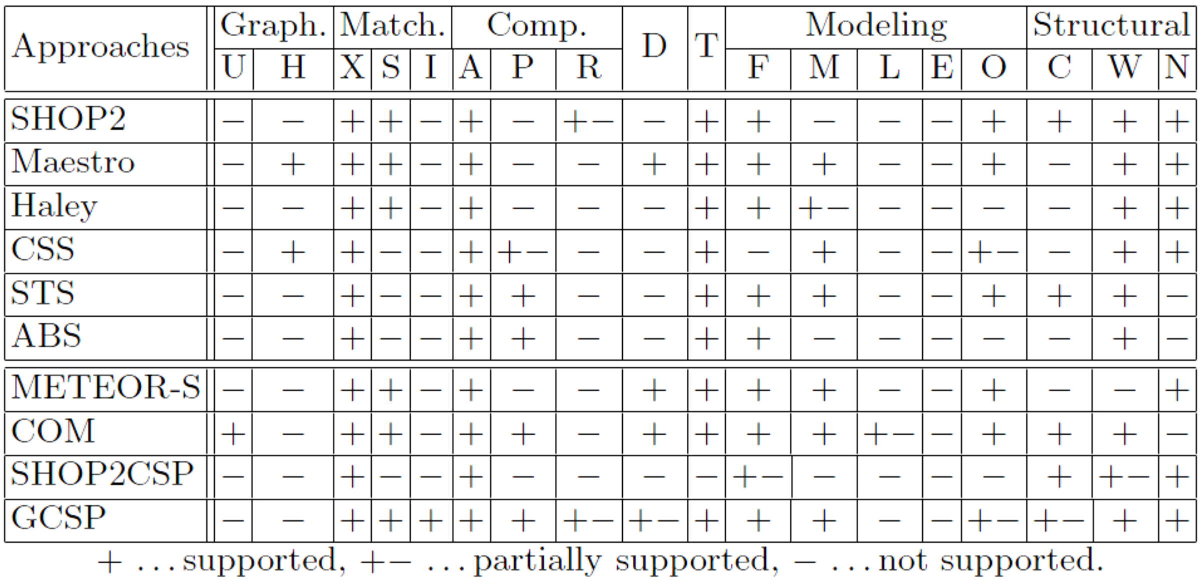
Figure 20 taken from (Grossmann et al. 2011) shows three phases of Web service composition together with the flow of the steps between and within the phases. During the **design phase** the behaviour of the workflow is modelled together with the data and data types that are required in each step of the workflow. The data model uses semantic concepts that determine Input Output Precondition Effect (IOPE), service level agreement information, and technical details on how to invoke the service. The design phase is followed by the **configuration phase** during that suitable service candidates are discovered in service repositories and registries. A suitable service candidate matches the requirements defined in the design phase before. If a requirement cannot be satisfied directly the service mediation step tries to find additional services that can be used in conjunction with another service to meet the requirement. If there is, for instance, a service candidate that can provide temperature values in degree Fahrenheit, but values in degree Celsius are wanted then a mediating service can be looked for that converts degree Fahrenheit in degree Celsius. The composition of the temperature service together with the unit conversion service meets the requirement then. The same mediation concept is foreseen to overcome heterogeneities on service interaction protocols. If a service can be invoked only be using protocol A and the service client can only query the service by

[4] http://www.internet-of-services.com/index.php?id=288&L=0

[5] http://www.w3.org/2001/sw/

using protocol B then a mediating service can be looked for that translates between the two protocols. In case more than one service candidate meets the requirements one service needs to be selected that is used in the actual composition. Grossmann suggests ranking services and service combinations according to Quality of Service parameters and picking the highest ranked service for composition then. Another way of selection is to leave the decision to the service client, but in the intended use case of automatic service composition this option is not really appropriate. The configuration phase is followed by the **enactment phase** in which the services selected before are bound to the workflow activities and query tasks they belong to. The services are being executed during this phase and monitored in case of longer lasting service calls, like subscriptions. Grossmann's phase model has high potential to be applied in IoT-A, the phases have to be detailed more, though to find its way into a reference architecture.

The same work (Grossmann et al. 2011) presents a comparison of existing static as well as dynamic composition approaches evaluated against criteria significant for IoT-A.



Approaches	Graph.		Match.			Comp.			D	T	Modeling					Structural		
	U	H	X	S	I	A	P	R			F	M	L	E	O	C	W	N
SHOP2	−	−	+	+	−	+	−	+−	−	+	+	−	−	−	+	+	+	+
Maestro	−	+	+	+	−	+	−	−	+	+	+	+	−	−	+	−	+	+
Haley	−	−	+	+	−	+	−	−	−	+	+	+−	−	−	−	−	+	+
CSS	−	+	+	−	−	+	+−	−	−	+	−	+	−	−	+−	−	+	+
STS	−	−	+	−	−	+	+	−	−	+	+	+	−	−	+	+	+	−
ABS	−	−	+	−	−	+	+	−	−	+	+	−	−	−	−	−	+	−
METEOR-S	−	−	+	+	−	+	−	−	+	+	+	+	−	−	+	−	−	+
COM	+	−	+	+	−	+	+	−	+	+	+	+	+−	−	+	+	+	−
SHOP2CSP	−	−	+	−	−	+	−	−	−	−	+−	−	−	−	−	+	+−	+
GCSP	−	−	+	+	+	+	+	+−	+−	+	+	+	−	−	+−	+−	+	+

+ ...supported, +− ...partially supported, − ...not supported.

Table 10 Comparison of dynamic composition approaches (Grossmann et al. 2011)

Table 10 lists the approaches analysed by (Grossmann et al. 2011) on the left side and shows the support of evaluation criteria in the columns of the table. The criteria are as follows: **Graph** Indicates whether the analysed approach accepts graphical design models being accepted by the orchestration. It is further distinguished whether the de-facto standard UML (**U**) is supported or other (**H**) notations are used. This criterion is not significant for IoT-A since this project focuses on BPMN graphical models. **Match** indicates the matchmaking technique applied in each approach. Syntactic (**X**), semantic (**S**) as well as instance-level (**I**) techniques are listed here. IoT-A is interested in semantic matching since the services are described semantically. **Comp.** classifies the composition type, e.g. atomic (**A**), composite or process (**P**), and re-configuration or re-planning (**R**). IoT-A is aiming at supporting all of the listed types. Discovery (**D**) denotes whether discovery is supported by the approach as also required in IoT-A. If a tool has been developed to implement the approach is indicated by a **+** in column **T. Modeling** lists five modelling aspects as follows: there is a formal language defined for service descriptions and requirements (**F**), model-driven architecture is supported (**M**), service life-cycles can be modelled (**L**), models are evolutionary (**E**) meaning they can be adapted during execution, and user requirements can be expressed through an orchestrator (**O**). Especially the orchestrator support is essential for IoT-A since the business process sets the requirements for service selection. The **Structural** criterion denotes the support of workflow structures such as control flow (**C**), data flow (**W**), and multiple instances of a service (**N**). This criterion is important for IoT-A, but workflow structures will be handled on process execution level.

The evaluation identifies GCSP (Mayer, Thiagarajan, and Stumptner 2009) as "especially suited for dynamic scenarios" (Grossmann et al. 2011), but the re-configuration criterion essential for unreliable IoT service environments is not supported sufficiently by any of the listed approaches.

Another survey of existing Web service composition approaches is presented in (Dessislava and Dimov 2011). That work analyses existing approaches along following dimensions reflected in Table 11 taken from the same paper: the Orchestration Model (**OM**), the Component Model (**CM**), the Quality Model (**QM**), Transaction support (**T**), the Data and Data Access Model (**DDAM**), the support of Exception Handling (**EH**), the Service Selection Model (**SSM**), and Tool Support (**TS**). OM shows the languages used for modelling the control flow within the composition. CM lists the languages for describing the components of the composition. The QM column indicates whether the composition approach takes Quality of Service (QoS) parameters into account. Column T denotes if the approach supports handling of transactions. The column DDAM shows if developers can explicitly specify the data flow between inputs and outputs of services. This determines the wiring between composite services. Under the EH column is listed, which approaches are able to take unexpected behaviour into account. This capability is essential for IoT service environments. Another important column is SSM indicating, which approach does support dynamic binding as it is required for mobile or unreliable IoT-services. The TS column shows that for almost all the approaches a tool has been implemented.

Ref.	OM	CM	QM	T	DDAM	EH	SSM	TS
2	BPMN	WSDL	Yes	No	Partial	Yes	Yes	Yes
3	BPEL	WSDL	Partial	No	No	No	Yes	Yes
4	PG	WSDL	No	No	No	No	No	Yes
5	PMC	WSDL	No	No	No	Yes	Yes	Yes
6	AIP	OWL–S,OWL	No	No	No	No	No	Yes
7	OWL–S	SWRL	No	No	Yes	No	No	Yes
8	WS–BPEL	WSDL,SLA	No	Yes	Partial	No	No	No
9	GM	WSDL,UDDI	Yes	Partial	No	Partial	No	Yes
10	GM	WSDL,WSLA,OWL	Partial	No	No	No	No	Yes
11	S	OWL-S,SAWSDL	Partial	No	Yes	No	No	Yes
12	HTN	OWL-S	Yes	No	No	No	Yes	Yes
13	PMC	WSDL	No	No	Yes	Yes	Yes	Yes
14	AIP	WSDL	Yes	No	No	No	Yes	Yes
15	S	WSDL,SLA	Partial	No	No	Yes	Yes	Yes
16	SM	WSDL,UDDI	No	No	No	No	No	Yes
17	XML Nets	OWL–QoS	Yes	No	Yes	No	Yes	No
18	HTN	OWL–S	No	No	No	No	No	Yes
19	OWL	WSDL	No	No	Yes	No	No	Yes

Table 11 Comparison of Web Service Composition Approaches (Dessislava and Dimov 2011)

The evaluation of the approaches listed in Table 11 proofs BPMN as suitable candidate for service orchestration as it supports most of the criteria (exception handling and dynamic service selection) essential for IoT-A conform systems. The authors of (Dessislava and Dimov 2011) have addressed the ability to specify abstract services in BPMN especially. This allows specifying compositions without hard-wiring of services at design time. It rather paves the way for dynamic binding of services depending on their availability at runtime of the composition.

3.3 Related Research Initiatives

During the ICT-FP7 project "SENSEI"[6] a service orchestration and composition approach has been developed that takes IoT specific requirements into account.

The SENSEI architecture depicted in Figure 21 taken from SENSEI deliverable D2.6 (Martin Strohbach et al. 2010) contains components that support dynamic service compositions and take associations to Entities into account, that are similar to what is called Virtual Entity in the terminology of the IoT-A project.

[6] http://www.sensei-project.eu/

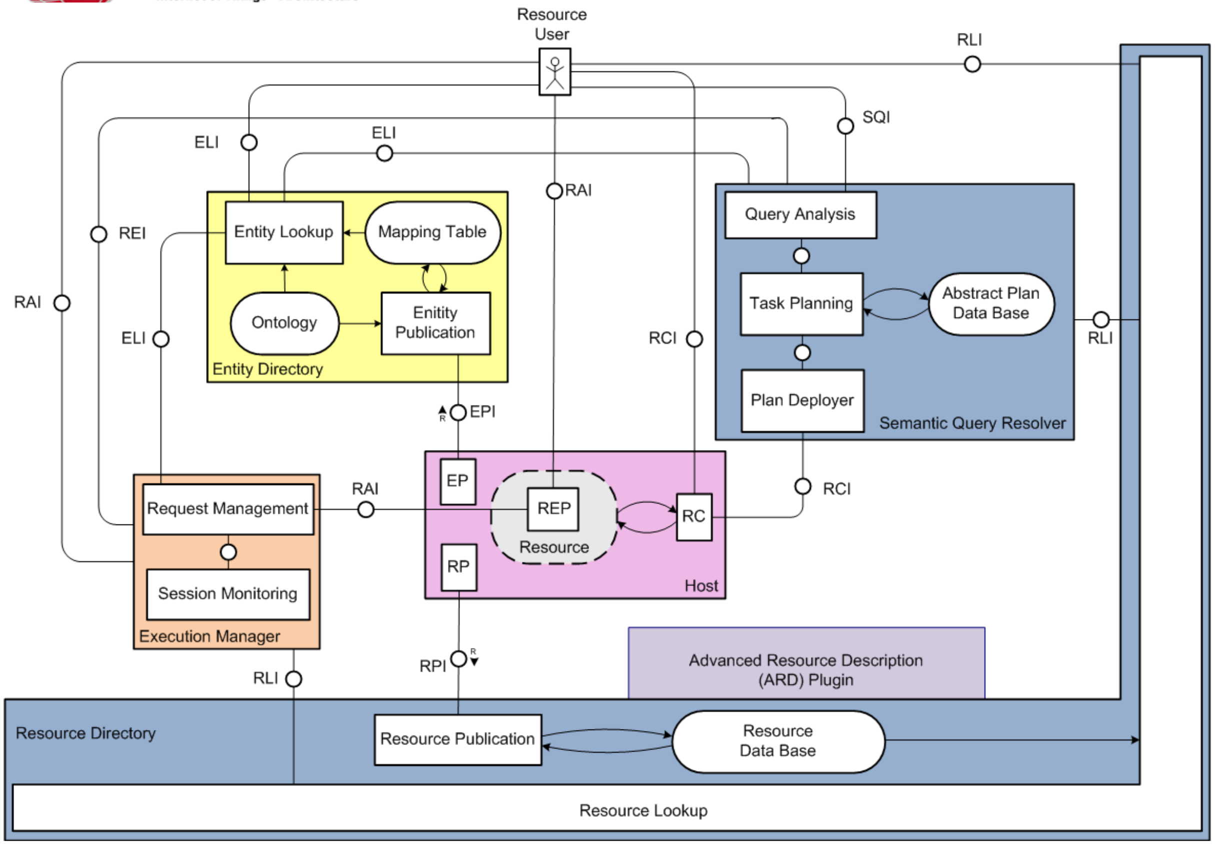


Figure 21 SENSEI Architecture (Martin Strohbach et al. 2010)

The **Semantic Query Resolver** is responsible for the lookup of (Virtual) Entities and the (IoT) Resources that are associated to the Entities. The associations are stored in the **Entity Directory** that contains **binding**-entries of following structure.

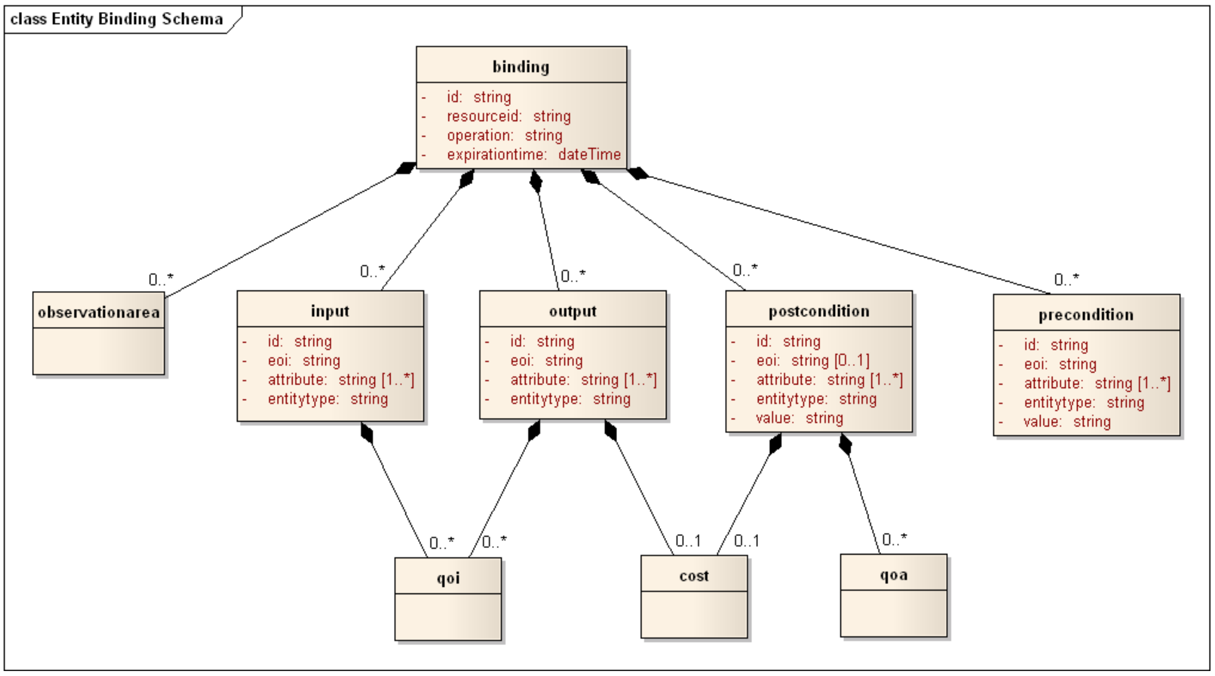


Figure 22 SENSEI Entity Binding schema (Martin Strohbach et al. 2010)

Based on the Binding Schema shown in Figure 22 attributes of (Virtual) Entities are bound to IOPE parameters of Resource operations or to IoT services in terms of IoT-A. The bindings allow a two way service discovery process, which works as follows: First it looks at the service that is associated to an Entity the service consumer is interested in. After that the lookup to the **Resource Directory** that acts as IoT service repository in this case, is performed. In the Resource Directory the Resource Descriptions are stored. These descriptions contain information about the IOPE parameters of the services together with Quality of Service parameters, like cost, on the parameters provided, and they contain details on how to invoke the services in order to execute them.

Especially the reconfiguration criterion that has been identified as missing in the existing approaches by (Grossmann et al. 2011) is addressed in (Villalonga et al. 2010) The figure below illustrates that an IoT Resource, that has become unavailable for a service composition during execution, is being replaced by another IoT-Resource of a similar type.

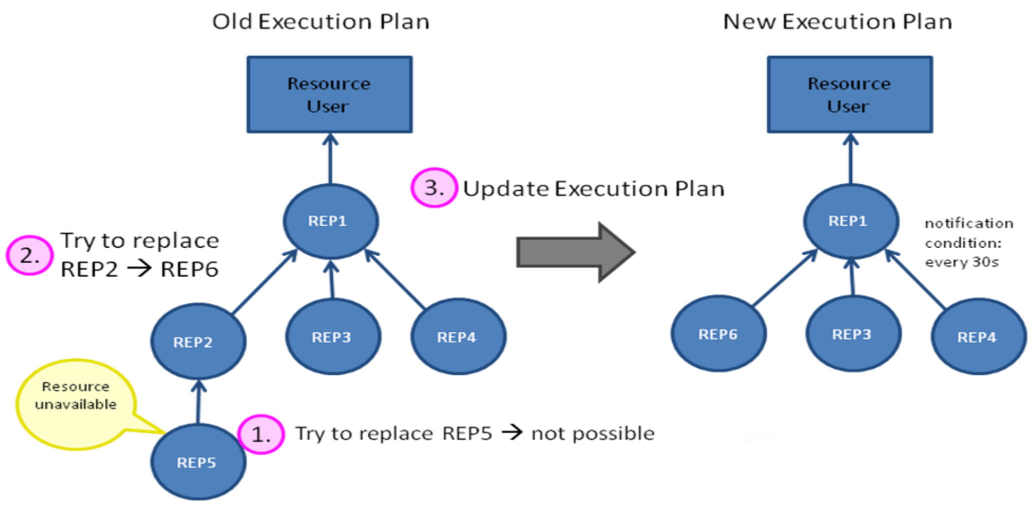


Figure 23 Re-planning of Service Compositions (Villalonga et al. 2010)

The self-repair ability in SENSEI is assured by the **Execution Manager** that monitors service executions. If one of the Resources (IoT Services) becomes unavailable during execution, the Execution Manager gets informed about the loss and reacts by triggering re-planning at the orchestrator, the Semantic Query Resolver.

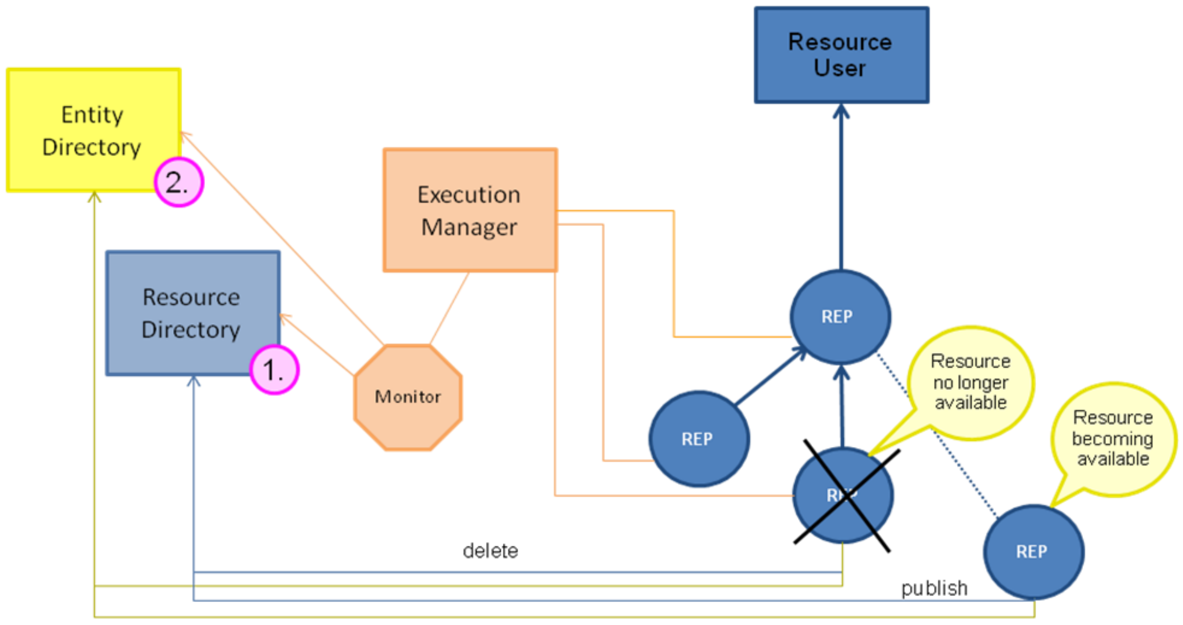


Figure 24 SENSEI Service Execution Monitoring (Villalonga et al. 2010)

Figure 24 illustrates that service execution monitoring is achieved by subscriptions to the Entity Directory as well as to the Resource Directory. The Entity Directory gets informed when one side of the association changes, either the Resource is no longer associated to the Entity, or the Resource, to which the Entity was associated, became unavailable. The Resource Directory also maintains a list of currently available Resources, and notifies subscribers about Resources that have not refreshed their entry in the directory and are therefore considered to be out of function.

4. Requirements of the IoT-aware Service Orchestration Concept

4.1 Methodology

IoT-A functional requirements presented in this chapter have been collected previously by WP2, WP6, and WP7. External mainly non-functional requirements have been collected from the service-oriented middleware / computing (SOM/C) literature and placed in the context of the Future Internet, from the works of (Issarny et al. 2011; Teixeira et al. 2011) for example.

4.2 IoT-specific Characteristics of Service Orchestration

This section includes the task T2.2 internal requirements gained by the WP7 use case as in the WP2 requirements document and additionally includes the WP2 specific requirements coming from WP6 by the stakeholder group.

The section distinguishes between functional and non-functional requirements applicable to IoT-aware services and service compositions. Non-functional (equivalently Quality of Service or simply QoS) requirements have been identified to be of high relevance in the IoT context. Examples of QoS attributes of services and service compositions include response time, reputation, price, etc.

The external QoS requirements introduced in this section address the need for adaptive and fault-tolerant service composition in the IoT context. These will serve as a basis QoS-aware service orchestration concepts as developed in deliverable D2.5.

4.2.1 Functional IoT-Aware Service Composition Aspects

If no single service satisfies the specific request issued by a user, then it may be required to compose existing services. The existing services can either be elementary (also called atomic services), or themselves a composition of existing services. The top of a composition hierarchy consists of services which are offered as applications to clients.

In Service Oriented Computing (SOC), service composition is usually either realized (a) in a centralized manner, via a single controller node, or (b) in a decentralized fashion. The former is the service orchestration approach, and assumes a single coordinator service that (1) interacts with the service consumer and (2) controls the execution of the services participating in the execution. The latter, known as choreography has no notion of a centralized coordinator, and every participant executes some (possibly partial) composite service, exchanging messages with other participants in a peer-to-peer manner.

As identified in chapter 2, while orchestration is easier to implement, it also has the drawback that the central coordinator node may become a bottleneck in terms of bandwidth and computational capabilities when the number of participants and /or the number of concurrent invocations are large. Moreover, if the coordinator is unreliable (e.g. hosted on a resource constrained, mobile hardware node), the coordinator represents a single point of failure, endangering the entire service composition. In contrast, choreography scales much better in terms of the total number of messages exchanged (since messages flow only between service provider and consumer) and since it does not rely on a central coordinator, it has no bottleneck and is more robust to failures. On the downside, the choreography approach to service composition may require a set of additional assumptions regarding a minimum amount of resource and computational capabilities of the individual participants.

The process of service composition is a sequence of steps carried out between the service requester and the service oriented middleware (e.g. consisting of a composition, execution and monitoring component) including internal steps of the middleware. Briefly, the process works as follows: the service consumer specifies its requirements in a specification language and the service composition engine (e.g. the orchestration engine) generates a composition that satisfies the needs of the consumer based on (a) its functional and non-functional requirements and (b) the set of available services, which then is deployed in the execution engine of the middleware. The middleware may further monitor the execution of the service composition as described in the next section.

The table below summarizes the list of internal functional requirements identified as critical for a successful service orchestration.

ID	Priority	Description	Rationale	Fit Criterion
IR2.44	High	The orchestration engine shall interpret service descriptions	service orchestration is done based on service descriptions	Correct service compositions can be created
IR2.45	High	The orchestration engine shall support creation of new applications	Higher level services should create new functionality	Higher level service is created based on lower level services
IR2.46	High	The orchestration engine shall create new service descriptions	The newly created service must be registered with service discovery	Valid service descriptions registered to be discovered correctly
IR2.47	High	The orchestration engine shall support flexible composition	Services involved in compositions can fail and need to be replaced by some serving equal needs	Services are replaced by similar ones in case of error
IR2.48	High	The orchestration engine shall handle scopes for selecting services for composition	Scopes selected for composed service must be applied to the atomic services as well	Scopes are applied correctly to all services a composition contains
IR2.53	Low	The orchestration engines shall support setting preferences for selecting services involved in composition	Users can have the possibility to prefer one service over another for any reason	Preferred services are selected for composition
IR2.49	Low	The orchestration engine shall increase quality of information by service composition	QoI can be increased by using additional information as reference	QoI is increased
IR2.50	High	The orchestration shall access service resolution	Orchestration depends on service descriptions provided by discovery	Service resolution can be accessed
IR2.51	High	The orchestration shall provide a feedback to the user who sent a composition request	The feedback should contain a message about the success of the requested composition	Feedback is send in any case (success/failed)

Table 12 Functional Requirements

4.2.2 Non-functional Aspects of IoT-Aware Service Composition

If multiple services with similar functionality exist, from which one can be selected to participate in a composition, often non-functional (equivalently QoS) attributes of the advertised services serve as the criteria for selection. QoS attributes of services translate to the QoS attributes of the overall composition, whereby the translation function depends on the composition pattern (e.g. sequential, parallel, conditional and loop). Service composition that takes into account QoS attributes is called QoS-aware and introduced in the pioneering work of (Zeng et al. 2004).

QoS-aware service composition is an important cornerstone of IoT-aware service composition, and briefly works as follows. The service consumer, in addition to the required functionality, also expresses QoS requirements that need to be satisfied by the composition (e.g. response time, availability). In order to continuously fulfil the QoS requirements specified by the user, the middleware also needs to monitor the execution of the composition to enable adapting the composition to changing QoS properties of the composed services.

A taxonomy of the non-functional requirements that have been identified as critical in the literature is illustrated in Figure 25 and explained below.

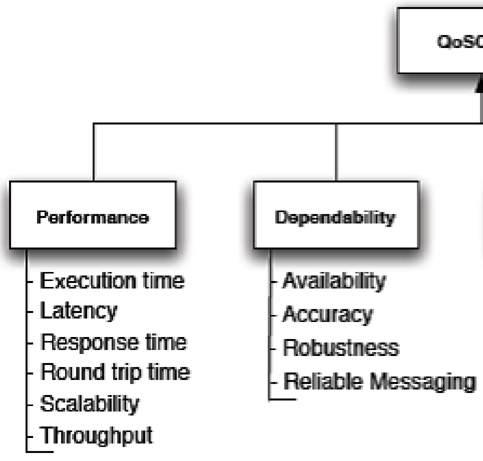


Figure 25 Taxonomy of QoS Attributes of Services and Compositions (Rosenberg 2009)

Any of the QoS attributes represent quantifiable measures and enables ranking of services in accordance to the preference of the user. In any of the QoS classes there can be positive and negative QoS attributes. A higher value of a positive attribute translates to a higher quality (e.g. throughput), whereas a higher value of a negative attribute corresponds to a lower quality (e.g. latency).

In the next sections we will explain the identified QoS requirements one by one.

4.2.2.1 Performance

To the performance class belongs a set of QoS metrics related to the measurable runtime performance of a service. Note that some of the attributes are user-specific and some are service-specific. Latency for instance, depends on the distance of the user to the service and the network conditions, whereas throughput is solely service specific. The performance attributes refer to a specific operation of the service.

- **Execution time:** the timespan the service provider needs to process a request, including unwrapping request message (e.g. in XML format), processing the request and wrapping response message (e.g. in XML format) at the server.
- **Latency:** the timespan a request from a client (e.g. a XML message) needs to reach its destination (i.e. the service)
- **Response time:** the *execution time* plus two times the *latency*
- **Round trip time:** response time plus the timespan for wrapping request message (e.g. in XML format) and unwrapping response message (e.g. in XML format) at the client.
- **Throughput:** The number of requests by a client for an operation that the service can process per time unit.
- **Scalability:** Calculated as the average round trip time for one request in relation to the average round trip time for m parallel requests. High scalability means that the scalability value tends to 1, whereas low scalability means that the value tends to 0.

4.2.2.2 Dependability

Dependability is defined by Avizîienis (Avizienis et al. 2004) as “the ability to avoid service failures that are more frequent and more severe than is acceptable”. It evolved to a concept that integrates QoS attributes such as availability, reliability, etc. It is noteworthy that in contrast to performance that is measured at the level of individual operations, dependability is measured at the level of the entire service.

- **Availability:** The probability that the service is up and running and produces correct results.
- **Accuracy:** The success rate produced by a service, calculated by counting all invocations in timeframe and relating that to the number of failed requests.
- **Robustness:** Defines the probability that a system can react properly to incorrect (invalid or incomplete) input messages. Robustness can be calculated by tracking all incorrect input messages and relating them to all valid responses generated in a time frame.
- **Reliable Messaging:** Binary metric indicating whether reliable messaging is available for a service.

4.2.2.3 Security and Trust

Security is an important issue in SOC and IoT since many of the services in a composition stem from disparate, possibly untrusted providers. In IoT, sensing can be “participatory” and opportunistic, e.g. any smartphone can act as a sensing device. In this environment, the trustworthiness of many sensing services is inherently low. This calls for security mechanisms as specified as part of the WS-Security framework [as below], and complemented by reputation techniques. Like dependability, security and reputation refer to the service as a whole.

- **Security:** defines the security feature supported by the service. Possible values are `None`, `UsernamePassword`, `X.509`, `SAML` or `Kerberos` (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss).
- **Reputation:** A measure of trustworthiness as rated by the users of the service, computed as the mean over n ratings, assuming numerical, one-dimensional rating values, e.g., 1 to 5, where 1 is worst and 5 is best.

4.2.2.4 Cost and Payment

Cost and payment are related to the bi-directional monetary costs from service consumer to service provider and vice-versa when using a service. Cost and Payment refer to a single operation of a service.

- **Price:** Monetary value associated with the invocation of a specific service operation.
- **Penalty:** The monetary value that the service provider has to pay to the service consumer in case the value of a QoS attribute as specified in the SLA (Service Level Agreement) for an operation has been violated. For example, if the maximum latency featured as a QoS parameter of the service is exceeded.

4.2.3 Service Choreography Aspects

Apart from the already identified advantages in terms of scalability and resiliency, service choreography is more appropriate as a means of coordination among different organizations.

For instance, in the retail use case described in chapter 2, the product travels along a supply chain during its lifecycle, thereby crossing the boundaries of several organizations. In order to get a complete picture of the history of conditions the product is exposed to during its lifecycle, data captured within the involved organisation needs to be communicated from the producer all the way down to the retail store.

The relevant data corresponds to the environmental conditions captured as time-stamped sensor measurements about the orchid, such as ambient temperature, humidity, luminosity, etc. that needs to be passed from one organisation to another across the supply chain, and that needs to be accumulated at the retailer.

In this use case, choreography at the level of the whole supply chain would enable to track the overall condition of the product and would complement the orchestration at the level of individual organisations. For this purpose, the different organisations would have to expose services that would correspond to the participants in a choreography, in which each organisation, upon registering the product in their stock, would use the service to request data about that product from the previous organisation in the supply chain.

Following this two-layered principle, a hybrid architecture, combining orchestration at the level of devices (where the gateway orchestrates a set of service-enabled sensing devices) and choreography among backed services provided by gateways has been recently proposed in (Dar et al. 2011).

4.3 Evaluation of Existing Service Orchestration Concepts and Tools

This section gives a brief evaluation of the existing service orchestration concepts and their tool support as presented in chapter 3.

REQ	description	BPMN / BPEL	GCSP	SENSEI
IR2.44	Interpret service descriptions	Semantic service descriptions cannot be matched against BPMN	Independent of service descriptions	Semantic service descriptions can be analysed
IR2.45	Create new applications	Supported	Supported	Compositions Supported for Input Output matching
IR2.46	Create new service descriptions	Not clear	Not clear	Composite services can be published as own service
IR2.47	Support flexible composition	Not automatically	Supported	Supported
IR2.48	Handle scopes for selection	Not supported on semantic models	Supported through constraints	Supported through standard query language
IR2.53	Setting preferences for service selection	Not supported on semantic services	Not supported	Supported through standard query language
IR2.49	Increase QoI through composition	Supported on process level	supported	Supported
IR2.50	Access service resolution	Discovery step is not foreseen	Partially supported	Fully supported
IR2.51	Send user feedback	Not clear	Not clear	Supported

Table 13 Evaluation of Service Orchestration Concepts

Table 13 shows how the functional requirements listed in Table 12 are met by the most suitable service concepts analysed in chapter 3. For all three of them respective tools exist.

The evaluation illustrates that BPMN/BPEL orchestration cannot orchestrate semantic services. Since the concepts on which BPMN/BPEL actions are modelled are lacking semantics, the orchestration cannot happen on a semantic layer. The service cannot be matched against a process activity if the semantic of the activity is not specified enough. It cannot be expressed in BPMN that a certain service must be associated to a particular Virtual Entity for instance. It is also not possible to define service orchestration for a class of Virtual Entities without naming a particular one. This would allow modelling business logic for any street lights that turn on if the day light turns off instead of modelling the same behaviour for each and every streetlight the process should be executed on. In BPMN world a service discovery step is not really foreseen as all the services are assumed to be available. This assumption does not hold in IoT environments. Another IoT specific requirement can be satisfied by BPMN/BPEL orchestration. Exception handling can be well specified on process level with BPMN concepts.

In opposite to the explicit modelling of exception handling, it would be better to handle self-repairing of processes being executed below process level. If a service being already orchestrated becomes unavailable only the orchestration needs to be repeated to discover an available service similar to the previous one. There is no change on process level. Current BPEL engines are not able to handle service discovery or self-repairing of orchestrations.

The missing self-repairing capability is what rules out GCSP as suitable candidate for service orchestration in IoT-A. It is suitable to include a dynamic service discovery and can be used to match semantic process models against semantic service descriptions, but the approach leaves open to which description languages it fits best.

SENSEI‘s orchestration model takes IoT-specific requirements into account and provides solutions for self-repairing of orchestrations. Furthermore SENSEI tackles actuation services besides sensing services. Also Quality of Service parameters are taken into account for selecting suitable service candidates as long as they are expressed in their semantic Advanced Resource Description model (Martin Strohbach et al. 2010). Only SENSEI provides the environment needed to support flexible service composition and service discovery by providing necessary repositories for Virtual Entities and IoT Services. The capability to subscribe to newly discovered services or updated VE-Service associations as well as the absence of the VE-Service associations allows self-repairing of on-going orchestrations. SENSEI although only provides service compositions through Input-Output matching. Matching of Preconditions and Effect is not supported, but important for actuation related services.

4.4 Conclusion

BPMN/BPEL orchestration provides very sophisticated models for control flow and data flow, but lacks in semantic descriptions. If BPMN/BPEL can be enriched with semantic meta-data, the orchestration concept will be able to handle semantic services. A service discovery phase needs to be included into the orchestration procedure. The relation between a BPMN activity and the service to be executed is dynamic in the IoT-world and can therefore only be decided upon runtime of the orchestration and not at design time of the process. The discovery capability will enable re-configuration of the service orchestration too. The dynamic orchestrator to be used in IoT-A can be based on SENSEI’s Semantic Query Resolver (SQR). The SQR provides flexible service composition for semantically described services. The composition method is restricted to Input-Output matching only and needs to be extended to Precondition-Effect checking in order to support actuation services. Also SENSEI’s Entity Directory and Resource Directory are good candidates for being used in IoT-A. They need to be able to handle service specifications as defined in D2.1 (Martin et al. 2012) though. The functionality of the Execution Manager contributes very well to IoT-A. A marriage of BPMN/BPEL orchestration with extended SENSEI components will be a good contribution towards the objectives defined in IoT-A WP2.

5. IoT-aware Service Orchestration Concept

This chapter introduces the concepts that have been developed during the project for IoT-aware Service Orchestration, Service Composition and Service Execution.

5.1 Service Orchestration Phase Model

Service orchestration describes which service will be invoked to achieve the goal of a BPMN activity designed in a business process. Due to the uncertain availability of IoT-Resources and their services this binding from activity to service cannot be done at design time of the business process. Thus the service orchestration model applied in IoT-A needs to be of dynamic nature. Taking this dynamicity into account the orchestration model includes the following four phases:

1. Modelling – an activity is outlined in a BPMN process

2. Resolution – finding widely distributed services needed for composition that are currently available and are suitable to serve the needs specified by the activity

3. Binding – assigning one service to the activity currently orchestrated and in case of a composite service also assigning the atomic services needed for composition

4. Execution – the assigned service is invoked and in case of a composite service the atomic services are invoked as well

The sequence diagram below illustrates the phases described above and shows which components are involved in service orchestration. The resolution phase makes use of Virtual Entity resolution and IoT Service Resolution functions provided by WP4. The interface descriptions are work in progress and only shown here for illustrative purpose.

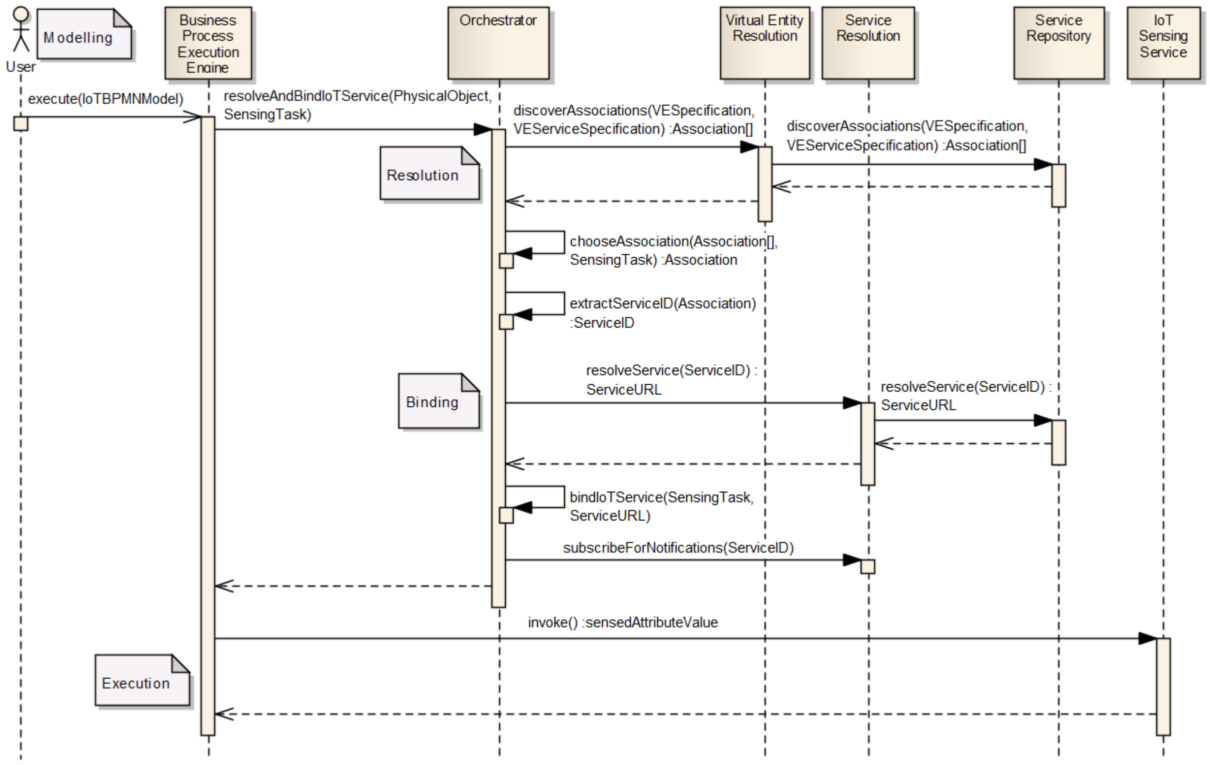

User
Modelling
Business Process Execution Engine
Orchestrator
Virtual Entity Resolution
Service Resolution
Service Repository
IoT Sensing Service
execute(IoTBPMNModel)
resolveAndBindIoTService(PhysicalObject, SensingTask)
discoverAssociations(VESpecification, VEServiceSpecification) :Association[]
discoverAssociations(VESpecification, VEServiceSpecification) :Association[]
Resolution
chooseAssociation(Association[], SensingTask) :Association
extractServiceID(Association) :ServiceID
resolveService(ServiceID) : ServiceURL
resolveService(ServiceID) : ServiceURL
Binding
bindIoTService(SensingTask, ServiceURL)
subscribeForNotifications(ServiceID)
invoke() :sensedAttributeValue
Execution

Figure 26 Sequence Diagram for Resolution, Binding, and Execution of IoT Services

After modelling an IoT-aware business process the user loads the model into a business process execution engine and requests the engine to execute the process. The execution engine asks the orchestrator to resolve the services for the interaction with the modelled real-world entities and to bind the services to the respective tasks in the model. During the execution of the process, the engine invokes these services directly. These phases of modelling, resolution, binding, and execution are explained in more detail in the following sections.

5.1.1 Modelling

To enable the creation of IoT-aware business processes we propose to use a language for business process modelling, which is augmented with IoT concepts, and an appropriate business process modelling environment. D2.2 (Meyer et al. 2012) describes the approaches for the augmentation of business process modelling languages for the IoT domain in general. Based on the IoT domain model, which was developed in D1.2 (Walewski et al. 2011), new modelling concepts for the interaction with physical entities in BPMN 2.0 (Object Management Group 2011) processes were introduced in (Sperner, Meyer, and Magerkurth 2011). For the depiction of a business process task, which interacts with a physical entity, new stencils for BPMN 2.0 were also proposed in (Sperner, Meyer, and Magerkurth 2011).

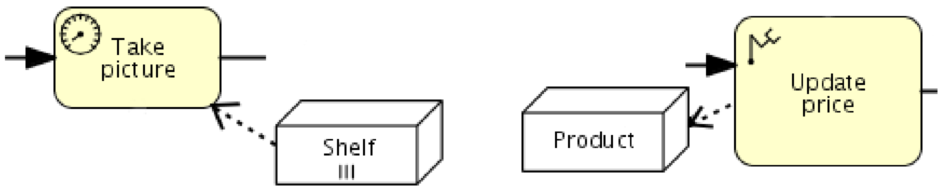


Figure 27 Examples for SensingTask, ActuationTask, and PhysicalObject as proposed in (Sperner, Meyer, and Magerkurth 2011)

Figure 27 shows examples of these new stencils. The task “Take picture” on the left side is a SensingTask, which is marked with a gauge icon. The brick labelled “Shelf” is a PhysicalObject and is decorated with a multi instance marker. The association between these two is a PhysicalAssociation, and the direction from the PhysicalObject to the task represents that information is flowing from the physical world to the process model, so this is a sensing interaction. On the right side, an ActuationTask “Update price” is shown, which interacts with a PhysicalObject “Product”. A robot arm is used as icon for the decoration of the ActuationTask. The direction of the PhysicalAssociation from the ActuationTask to the PhysicalObject denotes that information is flowing from the process model to the physical world, so this is an actuating interaction. (Cf. (Sperner, Meyer, and Magerkurth 2011))

BPMN 2.0 defines the ServiceTask as a specific task for the execution of arbitrary services. In a BPMN 2.0 model the endpoint of the desired service can be configured in the operationRef attribute of the ServiceTask instance (cf. Table 10.8 in (Object Management Group 2011)). For normal services in the internet of services such a static configuration of the endpoint is reasonable, because their addresses and interfaces change rather seldom. In contrast, services in the internet of things cannot be considered static, because the availability of the devices, which host the resources, which are exposed by the services, can change heavily over time. Due to the mobile nature of IoT Devices, the addresses of the services are also subject to change (cf.(Meyer et al. 2011)). Therefore it is reasonable not to describe the endpoints of SensingTasks and ActuationTasks explicitly in the IoT-aware BPMN model at modelling time, but to describe them in an abstract way, so they can be resolved and bound when starting the process execution.

For this abstract specification of the IoT-related tasks we propose a declarative approach based on ontologies. Ontologies are defined using the Resource Description Framework (RDF)[7] and its Schema

[7] http://www.w3.org/RDF/

RDFS[8]. The actual data is stored in a so-called Triple Store and can be queried using the SPARQL[9] query language. An initial definition of ontologies modelling entities, resources, and services in the IoT domain was presented in (De et al. 2011). In D4.1 (de las Heras et al. 2012) the service model and the entity model serve as a basis for the IoT Service Resolution and the Virtual Entity Resolution.

The specification of the PhysicalObject in the IoT-aware BPMN 2.0 process model is performed via a SPARQL query using the Entity Model from (De et al. 2011). To specify a room, which is located in the Zurich area, the query would look like the following:

```
PREFIX em: <http://purl.oclc.org/net/unis/EntityModel.owl>
PREFIX lm: <http://www.owl-ontologies.com/LocationModel.owl>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT ?room
WHERE {
  ?room em:hasType lm:Room;
        em:hasLatitude ?lat;
        em:hasLongitude ?long.
  FILTER ( ?lat > "47.3"^^xs:float && ( ?lat < "47.4"^^xs:float &&
               ( ?long > "8.5"^^xs:float && ( ?long < "8.6"^^xs:float ))))
}
```

If the actual room to be used is already known at design time, the user can formulate the SPARQL query for this exact room. The following example queries for the room U38 at the University of Surrey, which also serves as an example in (De et al. 2011):

```
PREFIX em: <http://purl.oclc.org/net/unis/EntityModel.owl>
PREFIX lm: <http://www.owl-ontologies.com/LocationModel.owl>
PREFIX db: <http://dbpedia.org/resource/>
SELECT ?room
WHERE {
  ?room em:hasType lm:Room;
        em:hasGlobalIdentifier db:University_of_Surrey
        em:hasLocalIdentifier lm:U38.
}
```

The specification of an associated SensingTask for sensing the temperature of the room uses the Service Model from (de las Heras et al. 2012) and could look like this:

```
PREFIX sm: <http://purl.oclc.org/net/unis/OWL-IoT-S.owl>
PREFIX lm: <http://www.owl-ontologies.com/LocationModel.owl>
PREFIX qu: <http://purl.oclc.org/NET/ssnx/qu/quantity#>
PREFIX pr: <http://www.linked-usdl.org/ns/usdl-price>
PREFIX db: <http://dbpedia.org/resource/>
PREFIX do: <http://dbpedia.org/ontology/>
SELECT ?service
WHERE {
  ?service sm:ObservationArea lm:Room;
           sm:hasOutputType qu:temperature;
           pr:hasPrice ?price.
  FILTER ( ?price < "0.05db:Euro"^^do:currency )
}
ORDER BY ASC(?price)
```

In this query only the type of the observation area is specified as “Room”, but the actual room, for which a service is sought-after, is not specified explicitly. As the SensingTask is bound to a PhysicalObject via a PhysicalAssociation in the BPMN process model, the actual room is meant to be the same, which is resolved by the SPARQL query for the PhysicalObject. The filter expression states, that the price of the service must not exceed 5 cent. The order clause defines, that the cheapest service should be the first result in the result set.

[8] http://www.w3.org/TR/rdf-schema/

[9] http://www.w3.org/TR/rdf-sparql-query/

In order to specify further IoT properties of the service, like e.g. its accuracy, the SPARQL query could utilize a further ontology, which models such IoT properties. Therefore, this approach is extensible and scalable enough, to deal with the IoT specific characteristics, which were introduced in chapter 4.2.

5.1.2 Resolution

On the start-up of the execution of an IoT-aware business process the execution engine requests the orchestrator to resolve and bind a service for every SensingTask (and ActuationTask) by calling the method resolveAndBindIoTService(PhysicalObject, SensingTask) with the associated PhysicalObject. The orchestrator compiles a VESpecification instance from the SPARQL query in the definition of the PhysicalObject and a VEServiceSpecification instance from the SPARQL query in the definition of the SensingTask. With these objects as parameters the orchestrator calls on the virtual entity resolution component the method discoverAssociations(VESpecification, VEServiceSpecification), which is defined in chapter 3.2.3 of (de las Heras et al. 2012). The virtual entity resolution routes the query to the service repository, which could be a USDL repository that can be queried with SPARQL queries. The service repository returns an array of Associations between Virtual Entities and Services, which fulfil the criteria defined in the SPARQL queries, and the virtual entity resolution passes this result to the orchestrator. The orchestrator uses the sorting of the services, which is defined in the SensingTask, to choose one of the Associations, and extracts its ServiceID.

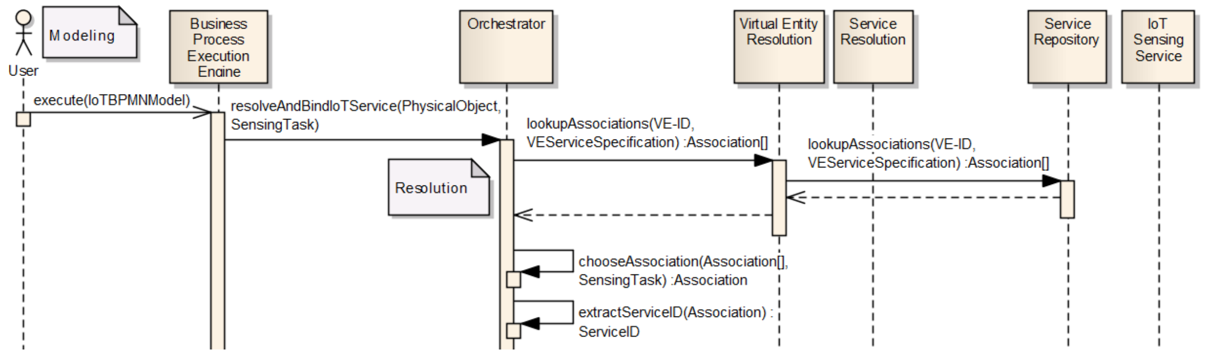

User
Modeling
Business Process Execution Engine
Orchestrator
Virtual Entity Resolution
Service Resolution
Service Repository
IoT Sensing Service
execute(IoTBPMNModel)
resolveAndBindIoTService(PhysicalObject, SensingTask)
lookupAssociations(VE-ID, VEServiceSpecification) :Association[]
lookupAssociations(VE-ID, VEServiceSpecification) :Association[]
Resolution
chooseAssociation(Association[], SensingTask) :Association
extractServiceID(Association) : ServiceID

Figure 28 Resolution of IoT Service for exactly specified PhysicalObject

If the user specified an exact room in the SPARQL query in the definition in the PhysicalObject, the resolution phase works as shown in Figure 28: The Orchestrator extracts the VE-ID from the query and calls the method lookupAssociation(VE-ID, VEServiceSpecification) on the Virtual Entity Resolution, as it is defined in chapter 3.2.2 of (de las Heras et al. 2012). The remainder of the resolution phase works as described above.

5.1.3 Binding

After the resolution of the ServiceID is done, the orchestrator continues with the binding phase. Therefore, the orchestrator calls the method resolveService(ServiceID) on the service resolution component, as it is defined in chapter 3.1.2 of (de las Heras et al. 2012). Like in the resolution step the service resolution component routes the query to the service repository. The service repository resolves the ServiceURL for the given ServiceID and returns it to the orchestrator. The orchestrator performs the actual binding by updating the BPMN SensingTask with the resolved ServiceURL. The orchestrator subscribes at the service resolution component for notifications about the bound service, so that it can react, if e. g. the service endpoint changes. Therefore it calls the method subscribeForNotifications(ServiceID), which needs to be introduced into the interface of the service resolution component.

5.1.4 Execution

During the execution of the business process the execution engine uses the service address bound to the SensingTask to invoke it. If the service became unavailable since it was resolved and doesn't respond to the invoking call, a re-resolution of a service for the SensingTask like described in the previous sections is performed automatically and the newly bound service is used in the process execution.

5.2 Service Composition

IoT-A provides support for orchestration of composite services which are composed of atomic (or already composed) services in order to provide higher order functionality. Those services require other services to provide them with data or actuation functionalities. To achieve flexibility those supporting atomic services are not hard-wired, but are resolved and bound during service orchestration as described in chapters 5.1.2 and 5.1.3. Composite services have inputs defined in their service specification that need to be resolved during orchestration. The figure below depicts a composite service that requires two inputs.

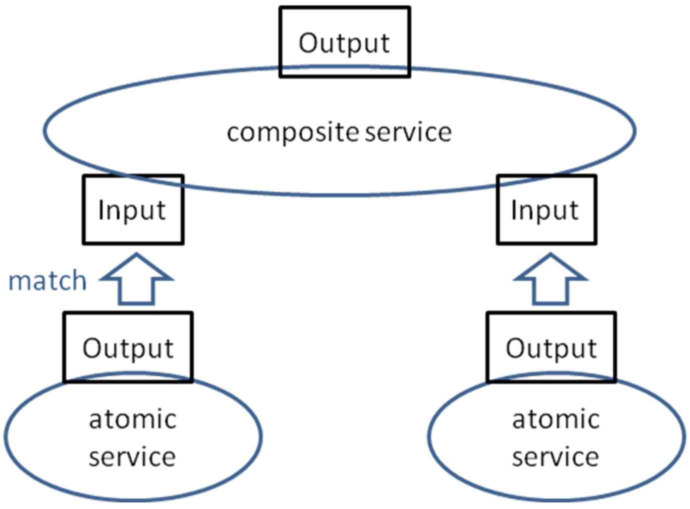


Figure 29 Input Output Matching

Figure 29 illustrates that the outputs of the atomic services need to match the inputs of the composite service. The following condition needs to be satisfied for each atomic service being part of the composite service:

$$(1)\ AtomicService_{Output} \equiv CompositeService_{Input}$$

Furthermore it needs to be assured that the atomic services are associated with the same Virtual Entity as the composite service is associated to. Under this condition the composite service affects the same Virtual Entity as the atomic services.

$$(2)\ For\ \forall AtomicService,\ CompositeService,\ VirtualEntity\_1,\ VirtualEntity\_2:$$

$$\exists Association(AtomicService,VirtualEntity)\ \wedge\ \exists Association(CompositeService,VirtualEntity)\ \Rightarrow VirtualEntity\_1 \equiv VirtualEntity\_2$$

Only in this case the functionality of the composite service can be achieved. This input-output matching is achieved by semantic equality. This means input and output need to be of the same class of semantic concept to achieve data integrity. It must be avoided that for instance a temperature value given in degree Fahrenheit is matched to a value expected in degree Celsius.

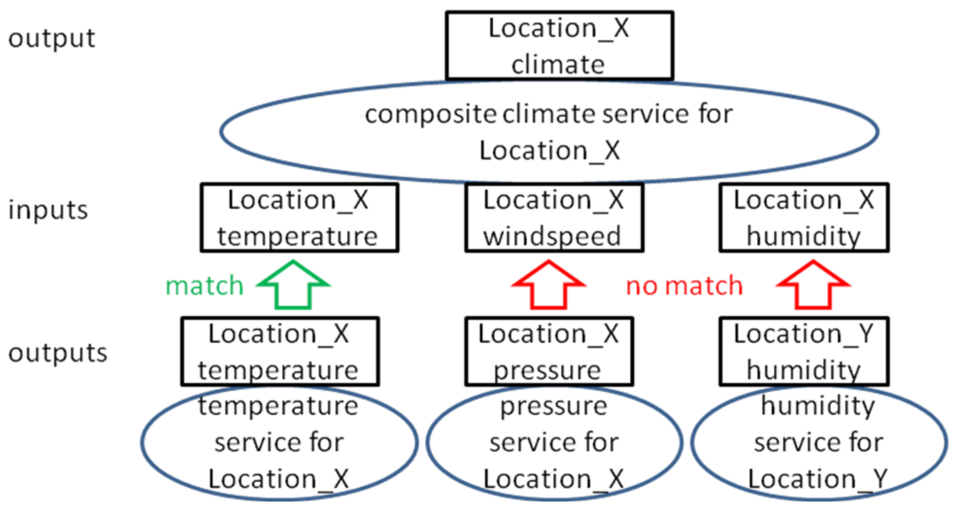


Figure 30 Input Output Matching Example

Figure 30 shows a composite service that provides the climate data for a certain Location_X as service output. Climate in this case is composed of temperature, wind speed, and humidity. Thus the respective data needs to be delivered as input to the composite service and appropriate atomic services need to be resolved. For the temperature input a suitable service candidate could be found; the atomic service that provides temperature satisfies condition (1). To fulfil condition (2) the temperature service must be associated to Virtual Entity Location_X, which is indeed the case in this example. For the second input a service is required that can provide wind speed values for Location_X. The service candidate shown in the figure can provide pressure values only for Location_X. In this case condition (2) is satisfied, but condition (1) is not fulfilled. For the humidity input the contrary is the case. The candidate service is able to deliver humidity data as required (condition (1) is met), but for a different location only what violates condition (2). The humidity value is only useful for the climate service if it is related to the same location as the temperature and wind speed values.

A similar orchestration method is applied for precondition-effect (post-condition) matching for composite services that have preconditions defined in their service specification.

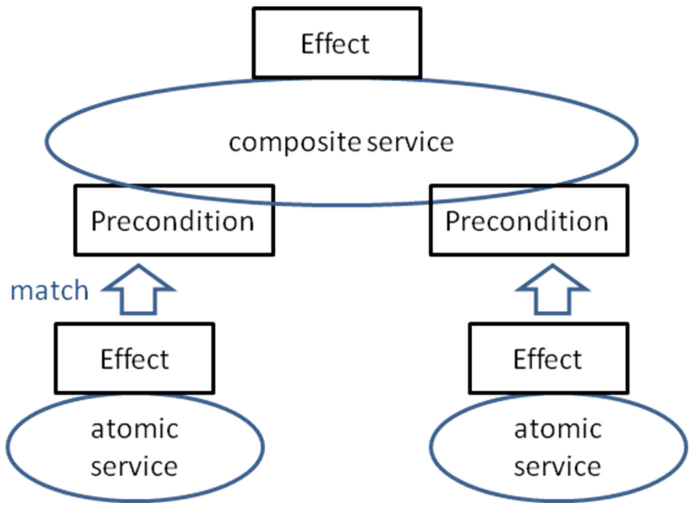


Figure 31 Precondition Effect Matching

Figure 31 shows a composite service that requires an atomic service that is able to satisfy the precondition defined in its service specification. An atomic service has exactly this condition denoted as effect in its own service specification so that this effect matches the precondition of the composite service perfectly. If following condition holds for any atomic service, the orchestration step succeeds:

$$(3)\ \mathit{AtomicService}_{\mathit{Effect}} \equiv \mathit{CompositeService}_{\mathit{PreCondition}}$$

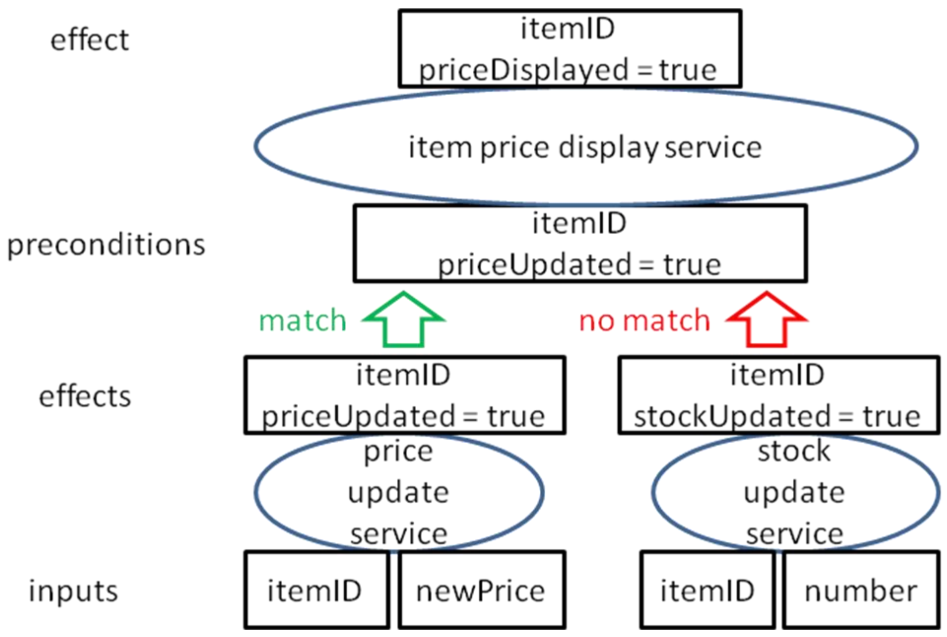


Figure 32 Precondition Effect Matching Example

Figure 32 illustrates a composite service that displays the price of an item in a store on an Electronic Shelf Label. Such label can be configured to display different numbers and therefore operates as actuator in this example. The ‘item price display service’ will be triggered only if the price of an item has been updated. Thus this service requires the precondition ‘priceUpdated = true’ to be satisfied. During decomposition a service needs to be found that can assure this condition for the item of the particular ‘itemID’ in store. The ‘price update service’ in this example has the requested condition as effect described in its service specification. This service is a suitable candidate for the service composition because condition (3) is fulfilled. The composition can be established since the effect of the atomic service matches the precondition of the composite service. The ‘price update service’ itself requires an item ID as well as the new price for the item as input. These inputs can be filled by the consumer of the ‘price update service’. The ‘stock update service’ is listed as another atomic service candidate in Figure 32 that provides ‘stockUpdated = true’ for an itemID as effect according to its service specification. Since ‘stockUpdated’ is semantically not equal to ‘priceUpdated’ this effect does not match the precondition of the ‘’item price display service’. Therefore the ‘stock update service’ is not suitable for the composition and is not considered during decomposition.

The following sequence diagram shows the decomposition steps done during service orchestration of composite services. The atomic services will be resolved one by one. Only if all inputs and preconditions can be satisfied the composition is considered as valid and the binding phase of orchestration can be entered. Having all inputs and preconditions resolved is an assumption for now. There might be use cases in which not all the possible atomic services need to be required, but the composition can still work.

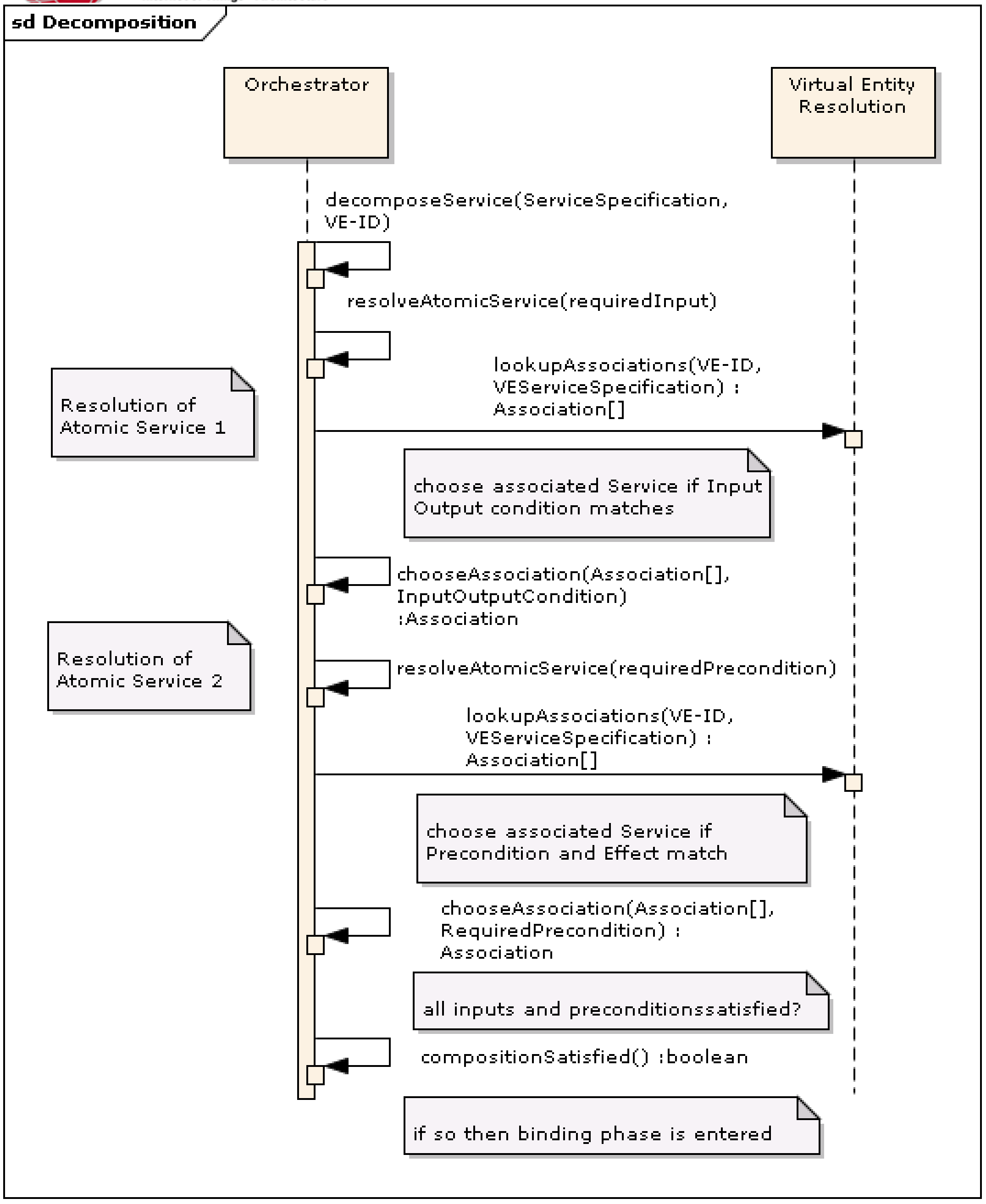

sd Decomposition
Orchestrator
Virtual Entity Resolution
decomposeService(ServiceSpecification, VE-ID)
resolveAtomicService(requiredInput)
lookupAssociations(VE-ID, VEServiceSpecification) : Association[]
Resolution of Atomic Service 1
choose associated Service if Input Output condition matches
chooseAssociation(Association[], InputOutputCondition) :Association
Resolution of Atomic Service 2
resolveAtomicService(requiredPrecondition)
lookupAssociations(VE-ID, VEServiceSpecification) : Association[]
choose associated Service if Precondition and Effect match
chooseAssociation(Association[], RequiredPrecondition) : Association
all inputs and preconditionssatisfied?
compositionSatisfied() :boolean
if so then binding phase is entered

Figure 33 Decomposition of Services

Figure 33 illustrates the steps the orchestrator has to do in order to decompose a composite service. At first the orchestrator has analysed the service description of the service and has detected that there are inputs to the service that need to be delivered by other services. The orchestrator then resolves the atomic services for each input of the composite service. The resolution phase for the atomic services is identical to the method described in chapter 5.1.2. The Virtual Entity Resolution component returns candidate services that fit to the VEServiceSpecification and are associated to the particular VE-ID. The returned candidate services are matched against the required input or precondition. If one service candidate matches the required conditions (1) and (2) or (2) and (3) the service is bound to the respective input or precondition. If all inputs and/or preconditions of the composite service are fulfilled the service composition is declared as satisfied and the orchestrator enters the binding phase as explained in chapter 5.1.3.

5.3 Service Execution

Although the focus of this document is on service orchestration and service composition, we also want to briefly inform about how the actual service execution will be realised both in the context of the forthcoming deliverable D2.4 and the final realisation of some of the use cases in work package 7. Especially the retail domain is notorious for its heterogeneous IT infrastructures with many legacy information systems and physical devices and therefore requires universally accessible service execution platforms. We will demonstrate the service execution by utilising IoT platforms brought in as background IP to the IoT-A project. In one of the retail use cases, even the interoperability of the two IoT service execution platforms will be shown by distributing functionality to two different platforms with in the same use case, namely sensor based quality control (see D7.2).

5.3.1 Context Management Framework

The NEC Context Management Framework (CMF) is a platform that simplifies the development of context-aware applications. It supports semantic access both of sensors and actuators and has been applied to many application areas – in particular to pervasive advertising and Digital Signage (M Strohbach and Martin 2011). For the purpose of service execution we focus on CMF's capability in managing sensor and context information.

As depicted in Figure 34, the CMF consists of multiple Context Agents grouped in clusters. The CMF can further be organized in clusters of clusters. Each agent provides access to information stored locally and at connected agents. Agents also enforce policies that define which applications and other agents can access which entity/attribute pairs.

A Context Agent offers information to applications from any of the following three sources: sensors accessed by retrievers, persistent context available from the storage component and processing units.

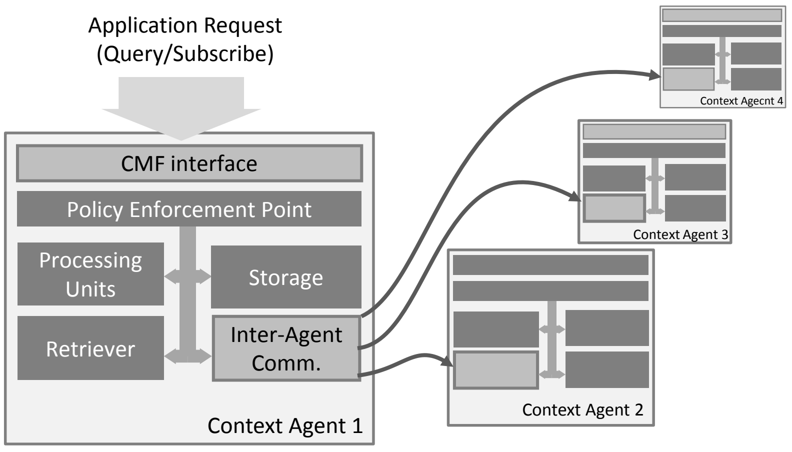


Figure 34 High level architecture of the Context Management Framework consisting of coordinating Context Agents

Applications typically access the CMF using the CMF Interface on a local Context Agent, i.e. an agent deployed on the same computing node as the application. The CMF Interface offers access via a declarative Context Access Language (CALA) that operates on entity/attribute based data structure. Entity/attribute pairs model real world objects like orchids, shelves, the retail store and their properties. CALA supports query, subscribe, insert, delete, and update operations that are executed on a cluster of Context Agents. Thus applications can access information in a fully declarative way and do not need to address individual agents directly. They are able to use a unified interface to access information generated by any sensor.

The CMF is OSGi based and runs on most devices that support a Java virtual machine, ranging from regular computers to embedded PCs – as used in professional Digital Signage displays – to Windows Mobile and Android mobile phones, home gateways and set top boxes. The programming interface is based on XML-RPC, and its main concepts are currently being standardized as part of the Context API in the Open Mobile Alliance (OMA) Next Generation Service Interface (NGSI).

5.3.2 Real World Integration Platform

The corresponding second middleware that we utilize for the realization of service execution is SAP's "Real World Integration Platform" (Rode and Küstner 2011) which is an OSGi based framework like CMF that facilitates the development of applications integrating real-world entities. Generally, this integration is performed by agents, which are specifically designed for each entity type. Furthermore, an agent can implement certain programming logic, not integrating any real world entities at all.

The overall architecture of the middleware is depicted in Figure 35. The Site Manager is used at design time for configuring the participating nodes and the distribution of the needed agents to these nodes. Additionally, the configuration of the agents itself is done in the Site Manager. All configuration data and agent code is stored in a central repository referred to as Central Instance.

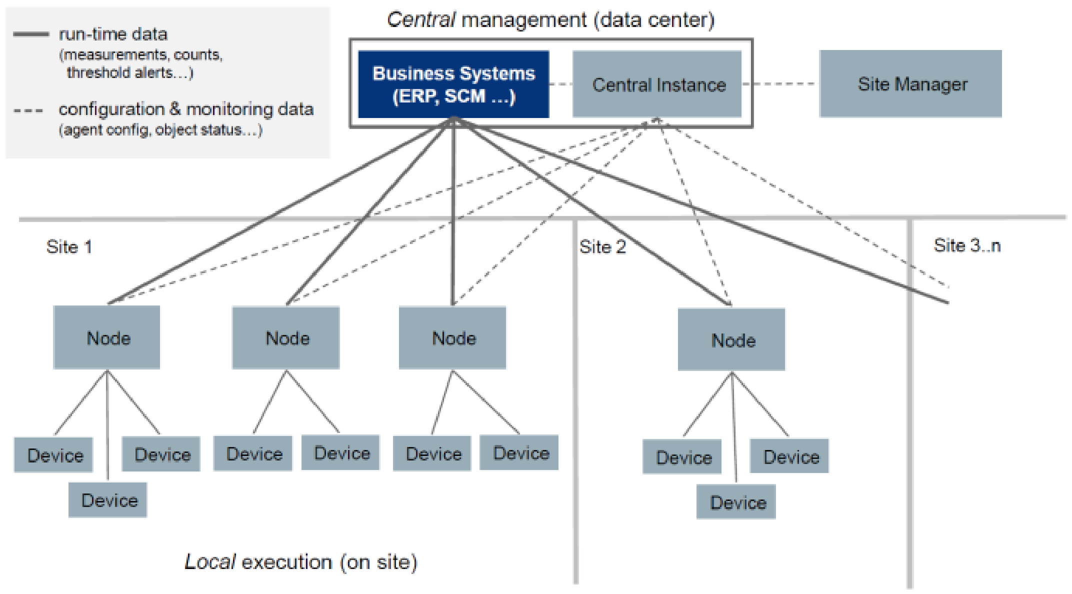

run-time data
(measurements, counts, threshold alerts…)
configuration & monitoring data
(agent config, object status…)
Central management (data center)
Business Systems (ERP, SCM …)
Central Instance
Site Manager
Site 1
Site 2
Site 3..n
Node
Device
Local execution (on site)

Figure 35 Real World Integration Platform

At run time the nodes load the agents and their configurations from the Central Instance and execute them. In order to reach a common goal, the agents communicate via messages, which are defined within them. These messages can be handled synchronously or asynchronously by the agents, constituting request-response-protocols or event-based communication.

5.3.3 Integration

On implementation level both platforms are comparable as they utilize a lightweight component-based OSGi approach for representing IoT devices such as sensors or actuators. The RWIP has a stronger focus on static, design-time system configuration through the Site Manager and the Central Instance, allowing e.g. the modelling and central monitoring of different retail stores with individual nodes representing different stores. Through the declarative CALA interface the CMF decouples applications from sensors and actuators. It is therefore more appropriate for integrating highly dynamic IoT entities such as mobile phones that are not predefined in a static system configuration that RWIP offers. Consequently, within concrete demonstrator realizations, for certain parts such as the identification of consumers via their mobile phones CMF could be used, whereas parts of the statically available store infrastructure such as the radio-controlled Electronic Shelf Labels could be integrated with RWIP agents.

As it can be seen, the execution of services itself is rather straightforward, as standard technologies are used for the actual service invocations. In accordance with the service specifications discussed in D2.1, from the viewpoint of the service execution platforms, there is no knowledge required at about the entity concept defined in work package 1 that forms the foundation of the service orchestration discussed in this document. In contrast, the service orchestration intelligence is located above the layer of the service execution platforms, so that the role of the IoT service execution platforms comes down to abstract things away from the specificities of the concrete IoT devices such as certain RFID manufacturers or sensor platforms. Both of the platforms presented in this section fulfil this requirement by offering standardised OSGI-based interfaces that can easily be accessed by the service orchestration.

5.4 IoT-Aware Aspects of Service Orchestration

The vision of autonomic computing (Kephart and Chess 2003), the second most cited paper in computer science, identifies “autonomic computing” as the most promising solution to the software complexity challenge faced by the modern IT industry. The software complexity problem is exacerbated in a pervasive environment in which billions of heterogeneous devices, running a huge number of distinct software stacks need to be integrated into a coherent IoT architecture.

Autonomic computing stipulates that systems should manage themselves according to the users’ and administrators’ goals, hiding the complexity of “how” these goals are achieved behind the autonomic “curtain” formed by four self-management pillars: *self-organization*, *self-optimization*, *self-healing* and *self-protection*. This section walks through the requirements formulated by each of these four pillars and translates these requirements into the context of IoT service composition.

5.4.1 Self-configuration in Service Orchestration

This section discusses the notion of self-configuration in the context of service orchestration, including the role of service resolution in the service composition process and how the dataflow is established with the orchestrated services.

Following the vision of self-configuration, services should autonomously advertise their capabilities by registering their service profile (as part of their service description) with the resolution infrastructure. The advertised capabilities include the inputs, outputs, preconditions and effects (IOPEs) as well as their QoS attributes related to performance, dependability and security (as detailed in Chapter 4).

When requesting a service, the user declaratively specifies the requested service on a high level and the middleware (consisting of execution, orchestration engine and service resolution) automatically and seamlessly coordinates to map the high level specification to a concrete service composition.

The user has the option of requesting a service by providing (a) a model of the service composition or (b) a high-level specification in terms of service capabilities. Concerning the former case, the model (expressed in BPMN for example) corresponds to a connected graph, where the nodes represent the tasks that need to be carried out and the edges denote the ordering of invocations. A task expresses the requested capabilities in terms of IOPEs and QoS attributes and is decoupled from any concrete service description. The binding of services to tasks happens at runtime, when the orchestration engine invokes the service resolution infrastructure to discover services whose advertised capabilities match the requested ones as described in chapter 5.1.2. In order to accommodate the latter case (b), models of service compositions (processes expressed in BPMN for example) should autonomously advertise as services by self-registering with the service resolution infrastructure.

In a self-configuring system, the orchestration shall continuously learn about newly registered or unregistered services and continuously adapt the service composition accordingly. Such self-configuring, adaptive behaviour is especially important in pervasive environments such as the IoT, where service endpoints are often subject to change due to the inherent mobility of the underlying hardware. In order to achieve the goal of self-configuration, a monitoring component is required that keeps track of service compositions and the services participating in the composition.

5.4.2 Self-optimization in Service Orchestration

This section explains how service orchestration allocates resources for providing composite services, how service orchestration controls lifecycle of composite service execution environments, and discusses how to handle scalability and distribution or load balancing.

If multiple matching services exist for a given task, then the orchestration engine shall be able to optimize the allocation of services to tasks with respect to a utility function. For this purpose, the user should have the ability to express the QoS requirements that the composite service should fulfil, in terms of non-functional properties such as performance, dependability and/or security and possibly various others (See Chapter 4). Based on the QoS specification and the model of the service composition, the orchestrator should be able to compose services such that some global utility function is maximized, for instance savings in terms of monetary costs, yet the QoS of the resulting composition must not violate the constraints imposed by the user.

Self-optimization is a continuous process in which the orchestrator continually seeks opportunities to improve the service composition’s performance and cost efficiency. Similar to self-configuration, a monitoring component is required to track changes in the service landscape and adjust compositions to reflect the optimal configuration. Besides that, the monitoring component needs to monitor the demands and QoS requirements and to react to changes appropriately. Frequent load fluctuations, due to a sudden surge in the demand of the compound service for example, call for (discovering and) binding additional services with identical capabilities (i.e. replica services) and spreading the load across them. Similarly, when the demand drops, the orchestrator needs to release services and consolidate invocations in order to minimize costs and increase efficiency.

In order to cope with unpredictable peaks of heavy load on web services and web service compositions, the authors of (Pautasso, Heinis, and Alonso 2005) have proposed a web service composition framework designed to support autonomic scalability. In the face of load variations, the framework alters its configuration enabling to optimally use the available resources. A follow-up work (Pautasso, Heinis, and Alonso 2006) describes and evaluates the architecture of the “JOpera” autonomic service orchestration middleware (http://www.jopera.org/), currently available as an Eclipse plug-in.

5.4.3 Self-healing in Service Orchestration

This section explains the re-setup of data flow from elementary services to composite services in the case one or more links to elementary services fail due to faults or mobility.

The vision of autonomic computing (Kephart and Chess 2003) stipulates that a self-healing system automatically detects diagnoses and repairs localized software and hardware problems. Translated to service composition in the context of IoT, the orchestration middleware should be able to trace failures down to individual elementary services, and provide capabilities to diagnose a problem by recognizing failure patterns from online analysis of log files (for example). The orchestration engine would then autonomously repair the composition by substituting the failed elementary services for identical functioning services in all affected compositions using a bottom-up approach, thereby re-establishing the data flow between the services participating in the composition.

The following sequence diagram shows an approach for a self-healing service subscription. The Business Process Execution Engine acting as service user subscribes to an IoT service and awaits notifications from time to time. During setup of the service subscription the IoT service was available and sent notifications to the subscriber. After a while the service becomes unavailable and is not able to send notifications to its subscriber. The absence of the IoT service is recognised by the Virtual Entity Resolution infrastructure being developed in WP4. Because the Orchestrator has subscribed to the association containing the IoT service in use, the Orchestrator gets notified about the association not being valid anymore. The orchestrator then initiates a new resolution phase with the VEServiceSpecification stored after the initial service request. If another service can be found that is suitable to replace the previous service adequately the Business Process Execution Engine receives notifications again without being notified about the service interruption. Only in case no suitable service can be found as replacement, the service user needs to be informed about the service loss. Upon that notification the service user has the chance to react in a way that suits her needs.

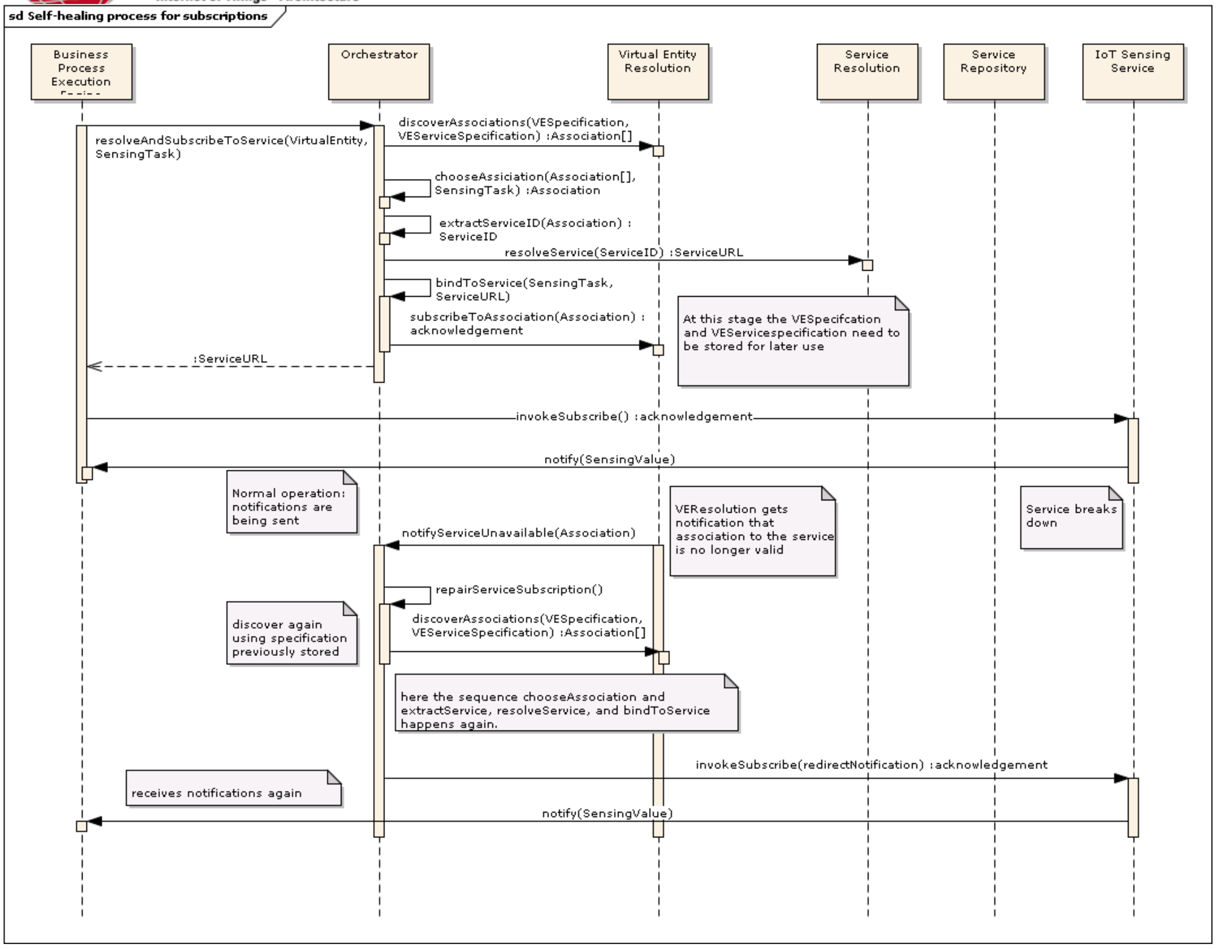

sd Self-healing process for subscriptions
Business Process Execution Engine
Orchestrator
Virtual Entity Resolution
Service Resolution
Service Repository
IoT Sensing Service
resolveAndSubscribeToService(VirtualEntity, SensingTask)
discoverAssociations(VESpecification, VEServiceSpecification) :Association[]
chooseAssiciation(Association[], SensingTask) :Association
extractServiceID(Association) : ServiceID
resolveService(ServiceID) :ServiceURL
bindToService(SensingTask, ServiceURL)
subscribeToAssociation(Association) : acknowledgement
At this stage the VESpecifcation and VEServicespecification need to be stored for later use
:ServiceURL
invokeSubscribe() :acknowledgement
notify(SensingValue)
Normal operation: notifications are being sent
VEResolution gets notification that association to the service is no longer valid
Service breaks down
notifyServiceUnavailable(Association)
repairServiceSubscription()
discover again using specification previously stored
discoverAssociations(VESpecification, VEServiceSpecification) :Association[]
here the sequence chooseAssociation and extractService, resolveService, and bindToService happens again.
invokeSubscribe(redirectNotification) :acknowledgement
receives notifications again
notify(SensingValue)

Figure 36 Self-Healing Service Subscription

5.4.4 Self-protection in Service Orchestration

Self-protection in service orchestration encompasses masking of, detection of and recovery from attacks and cascading failures. The system should automatically defend against malicious attacks and/or cascading failures, by using fault containment regions that prevent the propagation of failures across system components. For instance, a failure or malicious attack affecting one elementary service should be prevented to manifest at the interface of the composition including that service.

Fault containment can be achieved by redundancy, for instance via replication and voting. One important assumption is that failures are uncorrelated, translating to having multiple different versions of software / hardware components. The rationale behind multi-versioning is that compromising N replicas should be N times harder than compromising a single replica.

Software diversity can be achieved via N-version programming, where the code is implemented by N different teams. Hardware diversity is based on having different underlying hardware. In that respect, the IoT setting, in which the devices involved often are heterogeneous, both in software and hardware, increases the coverage of the failure independence assumption required for replication-based containment of malicious attacks.

As important as failure masking is fault-detecting and isolating the faulty service, for instance by employing majority voting, black-listing the service, and informing the service provider. The reason for immediate isolation is to prevent further usage of the resource (avoiding unnecessary reconfiguration). Informing the provider triggers the recovery and re-establishment of the compromised service.

5.5 Conclusions

Providing self-healing of orchestrations and compositions has an impact on the VE Resolution and IoT-Service Resolution infrastructure components being developed in WP4. Both frameworks need to provide interfaces on which the orchestrator can subscribe. Then the orchestrator will get notified when a service being part of an on-going orchestration becomes unavailable and is therefore no longer present in the service repository. A similar notification is expected from the VE Resolution infrastructure that informs the orchestrator about VE to Service associations that are no longer valid or were updated. Upon these notifications the orchestrator can trigger the resolution again to find suitable replacements being able to repair the orchestration and continue service delivery. Further IoT-aware aspects of service orchestration like self-optimisation and self-protection, will be researched during the next months of the project. The outcome of this research will be published in the respective deliverable D2.5.

6. Conclusions and Outlook

In this deliverable we have presented the IoT-A service orchestration concept based on a detailed analysis of existing service orchestration approaches and IoT specific aspects of service orchestration. We have also dealt in general with the question of service orchestration versus service choreography. This discussion has revealed a fundamental challenge in IoT-A service resolution. On the one hand, the aim of the IoT-A project is to unify the heterogeneous island solutions that currently make up the majority of IoT solutions. It is simply a given fact that the IoT world as it exists today is fragmented and demands a unifying reference architecture that takes the existing approaches into consideration. Likewise, the domain model as it is developed in work package 1 of the IoT-A project goes beyond the state of the art of existing solutions by introducing the concept of augmented entities that may temporarily and spatially exist beyond the scope of the individual technical island solution. If we orient our view of service orchestration with the implications of the IoT-A domain model, we perceive the need of service infrastructures that revolve around the concept of an augmented entity. One implication of the augmented entity is that it has a life-cycle that is potentially much larger than the scope of the IoT-A aware middleware that might currently be managing service orchestration and resolution. If we are serious about the fact that, for instance, the orchid does not only exist in the retail store where it's temperature and humidity is being measured (see the retail use case discussed in D7.1 and D7.2), but was at a distribution centre before, and will be at a consumer’s place afterwards, we must consider service choreography as a conceptually attractive approach of dealing with the changing contexts of the augmented entities, for which centralised service orchestration has its limitations. On the other hand, the real-world constraints and limitations of current IoT technologies, both in terms of technical resources on the devices and large-scale software infrastructures, demands a focus on viable solutions and thus lets us favour orchestration over choreography.

Based on the service orchestration paradigms we have then closely examined and discussed both the traditional means of service orchestration, mostly from the BPM and web service orchestration worlds, and the specificities of the Internet of Things, in order to come up with a service orchestration concept that both integrates with existing enterprise systems and does the idiosyncrasies of the Internet of things justice. As scalability is one of the key challenges of the Internet of things, we have especially looked at the different possibilities of realising self-* properties within service orchestration approaches.

A central contribution of the service orchestration concept is the phase approach that spans the typical phases of design, resolution, binding, and then finally execution. This is closely linked to the results from work package 4 in terms of resolution and binding, as well as the forthcoming deliverable D2.4 that discusses the tool support contributed by work package 2. If we take an outlook on how the service orchestration concept discussed in this deliverable influences and relates to forthcoming work in the IoT-A project, this is clearly the central link apart from a realisation in the work package 7 use cases, especially the retail use case. The phase model as it is discussed in this document marks the underlying approach of the process modelling tool that will be implemented directly after M18. This means that the theoretical work outlined in this deliverable will immediately be implemented and thereby evaluated within the work package 2 tool development activities. To look even further, the work presented in this document D2.3 that will manifest in a respective tool discussed in D2.4 will be utilised in the use case development, so that as a general IoT-A project result, service orchestration will be demonstrated in a use case demonstration implemented with a tool developed in D2.4 based on the results of D2.3.

Glossary

The following table describes the definitions of the main concepts relevant for the work in WP2. It consists of the official terms of the project, as they are referenced at http://www.iot-a.eu/public/terminology as well as the WP2 specific terms that are relevant for WP2 deliverables, but not necessarily for the work done in other work packages. These WP2-specific terms such as Business Process Management, Business Process Execution, Service Composition, Service Orchestration, Service Choreography, Complex Event Processing, etc. mostly relate to the Future Internet technologies and higher level enterprise systems that WP2 is concerned with.

Term	Definition	Source
Active Digital Entity	Any type of active code or software program, usually acting according to a *Business Logic*.	Internal
Actuators	"An *actuator* is a mechanical device for moving or controlling a mechanism or *system*. It takes energy, usually transported by air, electric current, or liquid, and converts that into some kind of motion."	[Sclater2007]
Address	An address is used for locating and accessing – "talking to" – a *Device*, a *Resource*, or a *Service*. In some cases, the ID and the Address can be the same, but conceptually they are different.	Internal
Application Software	"Software that provides an application *service* to the *user*. It is specific to an application in the multimedia and/or hypermedia domain and is composed of programs and data".	[ETSI- ETR173]
Architectural Reference Model	The IoT-A architectural reference model follows the definition of the IoT reference model and combines it with the related IoT reference architecture. Furthermore, it describes the methodology with which the reference model and the reference architecture are derived, including the use of internal and external stakeholder requirements.	Internal
Architecture	"The fundamental organization of a *system* embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution".	[IEEE-1471-2000]
Architecture Vision	"A high-level, aspirational *view* of the target *architecture*."	[TOGAF9]
Aspiration	"*Stakeholder* Aspirations are statements that express the expectations and desires of the various *stakeholder*s for the *service*s that the final [*system*] implementation will provide."	[E-FRAME]
Augmented Entity	The composition of a *Physical Entity* together with its *Virtual Entity*.	Internal
Association	An association establishes the relation between a *service* and *resource* on the one hand and a *Physical Entity* on the other hand.	Internal
AutoID and Mobility Technologies	"Automatic Identification and Mobility (AIM) technologies are a diverse family of technologies that share the common purpose of identifying, tracking, recording, storing and communicating essential business, personal, or product data. In most cases, AIM technologies serve as the front end of enterprise software *systems*, providing fast and accurate collection and entry of data. AIM technologies include a wide range of solutions, each with different data capacities, form factors, capabilities, and "best practice" uses. AIM technologies also include mobile computing devices that facilitate the collection, manipulation, or communication of data from data carriers as well as through operator entry of data via voice, touch screens or key pads.	[AIMglobal]
Business Logic	Goal or behaviour of a system involving Things serving a particular business purpose. Business Logic can define the behaviour of a single Thing, a group of Things, or a complete business process.	Internal

Business Process	“A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.”	[Weske2007]
Business Process Execution	Business Process Execution (BPE) consists of the manual, semi-manual and automated execution of business processes over actors. An important precondition for the automated execution of business processes is the technical specification of the business process model by technical analysts. The relevant industry standards for describing an executable business process model are BPEL and BPMN. Common IT-Systems supporting the execution of a huge number of automated processes use a process execution engine as a central controlling component.	Internal
Business Process Management	Business Process Management (BPM) is a structured approach that employs methods, policies, metrics, management practices and software tools to manage and continuously optimize the activities and processes of an organization. Process management uses an iterative process revision cycle known as the BPM lifecycle, which enables the continuous improvement of business processes. Depending on the methodology the following lifecycle phases are distinguished: Definition, Modelling, Simulation, Deployment, Execution, Monitoring, Analysis, Optimization and Discovery. In the IoT-A project we mostly focus on parts of the lifecycle phases Modelling, Deployment and Execution.	Internal
Business Process Modelling	Business Process Modelling is the activity of representing processes of an enterprise by abstracting the business process in a graphical or non-graphical model for progressing with one of the other BPM lifecycle phases in any order. Modelling a business process is typically performed by business analysts. The industry relevant standards are UML, EPC and BPMN. WP2 focuses on extending business process modelling approaches by IoT-specific characteristics.	Internal
Complex Event Processing	Complex event processing (CEP) describes a method of processing a multitude of events triggered by various information sources, and consecutively derive more meaningful events from a set of low level events. In IoT-A CEP is considered as an approach to bridge real-world events to the execution of business processes. We use the term event processing in the narrowest sense, i.e. analysis of events and subsequent decision making in terms of taking actions in real-time, are not considered as event processing, but are rather considered part of the business logic.	Internal
Constrained Network	A constrained network is a network of devices with restricted capabilities regarding storage, computing power, and / or transfer rate.	Internal
Controller	Anything that has the capability to affect a *Physical Entity*, like changing its state or moving it.	Internal
Credentials	A credential is a record that contains the authentication information (credentials) required to connect to a resource. Most credentials contain an user name and password.	Internal
Device	Technical physical component (hardware) with communication capabilities to other IT systems. A *device* can be either attached to or embedded inside a *Physical Entity*, or monitor a *Physical Entity* in its vicinity.	Internal
Digital Entity	Any computational or data element of an IT-based system.	Internal
Discovery	Discovery is a *service* to find unknown *resources/entities/services* based on a rough specification of the desired result. It may be utilized by a human or another *service*. Credentials for authorization are considered when executing the discovery.	Internal
Domain Model	“A domain model describes objects belonging to a particular area of interest. The domain model also defines attributes of those objects, such as name and identifier. The domain model defines relationships between objects such as “instruments produce data sets”. Besides describing a domain, domain models also help to facilitate correlative use and exchange of data between domains”.	[CCSDS 312.0-G-0]
Energy-harvesting Technologies	“*Energy-harvesting* (also known as power harvesting or energy scavenging) is the process by which energy is derived from external sources (e.g., solar power, thermal energy, wind energy, salinity gradients, and kinetic energy),	[Wikipedia EH]

	captured, and stored. Frequently, this term is applied when speaking about small, wireless autonomous *device*s, like those used in wearable electronics and wireless *sensor network*s.	
Future Internet	Future Internet is a summarizing term for worldwide research activities dedicated to the further development of the original Internet. In the IoT-A project we focus mostly on the Internet of Things and the Internet of Services as major constituents of the Future Internet that are consolidated within WP2.	Internal
Gateway	A Gateway is a forwarding element, enabling various local networks to be connected.	Internal
Global Storage	*Storage* that contains global information about many *entities of interest*. Access to the *global storage* is available over the *Internet*.	Internal
Human	A human that either physically interacts with Physical Entities or records information about them, or both.	Internal
Identity	Properties of an entity that makes it definable and recognizable.	Internal
Identifier (ID)	Artificially generated or natural feature used to disambiguate things from each other. There can be several Ids for the same *Physical Entity*. The set of Ids is an attribute of a *Physical Entity*.	Internal
Information Model	"An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context. The information model is an abstract representation of entities which can be real objects such as devices in a network or logical such as the entities used in a billing system. Typically, the information model provides formalism to the description of a specific domain without constraining how that description is mapped to an actual implementation. Thus, different mappings can be derived from the same information model. Such mappings are called data models."	[AutoI]
Infrastructure Services	Specific services, which are essential for any IoT implementation to work properly. Such services provide support for essential features of the IoT.	Internal
Interface	"Named set of operations that characterize the behaviour of an entity."	[OGS]
Internet	"The *Internet* is a global *system* of interconnected computer networks that use the standard *Internet* protocol suite (TCP/IP) to serve billions of *user*s worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope that are linked by a broad array of electronic and optical networking technologies. The *Internet* carries a vast array of information *resources* and *service*s, most notably the inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail. Most traditional communications media, such as telephone and television *service*s, are reshaped or redefined using the technologies of the *Internet*, giving rise to *service*s such as Voice over *Internet* Protocol (VoIP) and IPTV. Newspaper publishing has been reshaped into Web sites, blogging, and web feeds. The *Internet* has enabled or accelerated the creation of new forms of *human* interactions through instant messaging, *Internet* forums, and social networking sites. The *Internet* has no centralized governance in either technological implementation or policies for access and usage; each constituent network sets its own standards. Only the overreaching definitions of the two principal name spaces in the *Internet*, the *Internet*-protocol address space and the domain-name *system*, are directed by a maintainer organization, the *Internet* Corporation for Assigned Names and Numbers (ICANN). The technical underpinning and standardization of the core protocols (IPv4 and IPv6) is an activity of the *Internet* Engineering Task Force (IETF), a non-profit organization of loosely affiliated international participants that anyone may associate with by contributing technical expertise."	[Wikipedia IN]
Internet of Things (IoT)	The global network connecting any smart object.	Internal
Interoperability	"The ability to share information and services. The ability of two or more systems or components to exchange and use information. The ability of systems to provide and receive services from other systems and to use the services so interchanged to enable them to operate effectively together."	[TOGAF 9]
IoT Service	Software component enabling interaction with resources through a well-defined interface. Can be orchestrated together with non-IoT services (e.g.,	Internal

	enterprise services). Interaction with the service is done via the network.	
Local Storage	Special type of *resource* that contains information about one or only a few *entities* in the vicinity of a *device*.	Internal
Location Technologies	All technologies whose primary purpose is to establish and communicate the location of a *device* e.g. GPS, RTLS, etc.	Internal
Look-up	In contrast to *discovery*, *look-up* is a *service* that *addresses* exiting known *resources* using a key or *identifier*.	Internal
M2M (also referred to as machine to machine)	"The automatic communications between *device*s without *human* intervention. It often refers to a *system* of remote *sensor*s that is continuously transmitting data to a central *system*. Agricultural weather sensing *system*s, automatic meter reading and *RFID* tags are examples."	[COMPDICT-M2M]
Microcontroller	"A *microcontroller* is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. *Microcontroller*s are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications. *Microcontroller*s are used in automatically controlled products and *device*s, such as automobile engine control *system*s, implantable medical *device*s, remote controls, office machines, appliances, power tools, and toys. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output *device*s, *microcontroller*s make it economical to digitally control even more *device*s and processes. Mixed signal *microcontroller*s are common, integrating analogue components needed to control non-digital electronic *system*s".	[Wikipedia MC]
Network resource	*Resource* hosted somewhere in the network, e.g., in the cloud.	Internal
Next-Generation Networks (NGN)	"Packet-based network able to provide telecommunication *service*s and able to make use of multiple broadband, QoS-enabled transport technologies and in which *service*-related functions are independent from underlying transport-related technologies"	[ETSI TR 102 477]
Observer	Anything that has the capability to monitor a *Physical Entity*, like its state or location.	Internal
On-device Resource	*Resource* hosted inside a *Device* and enabling access to the *Device* and thus to the related *Physical Entity*.	Internal
Operator	The operator owns administration rights on the services it provides and/or on the entities it owns, is able to negotiate partnerships with equivalent counterparts and define policies specifying how a service can be accessed by users.	Internal
Passive Digital Entities	A digital representation of something stored in an IT-based system.	Internal
Physical Entity	Any physical object that is relevant from a user or application perspective.	Internal
PhysicalObject	BPMN element representing a physical entity.	[SPERNER2011]
Perspective (also referred to as architectural perspective)	"Architectural perspective is a collection of activities, checklists, tactics and guidelines to guide the process of ensuring that a *system* exhibits a particular set of closely related quality properties that require consideration across a number of the *system*'s architectural *views*."	[ROZANSKI2005]
Privacy	Information Privacy is the interest an individual has in controlling, or at least significantly influencing, the handling of data about themselves.	Internal
Process Execution Engine (also referred to as Business Process Execution Engine)	The process execution engine (PEE) is the central component with the complete process overview, which handles the specified technical process model. The PEE decides which service calls take place under which conditions and controls the process by orchestrating services and resources via interfaces. Automated processes are impacted by activities of human operators, services, and their descriptions, as well as resources and their descriptions. The PEE supports the process controlling with key indicators for evaluating and monitoring the process.	Internal
Reference Architecture	A reference architecture is an architectural design pattern that indicates how an abstract set of mechanisms and relationships realises a predetermined set	Internal

	of requirements. It captures the essence of the architecture of a collection of systems. The main purpose of a reference architecture is to provide guidance for the development of architectures. One or more reference architectures may be derived from a common reference model, to address different purposes/usages to which the Reference Model may be targeted.	
Reference Model	“A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. A reference model may be used as a basis for education and explaining standards to non-specialists.”	[OASIS-RM]
Resolution	Service by which a given *ID* is associated with a set of *Addresses* of information and interaction *Services*. Information services allow querying, changing and adding information about the thing in question, while interaction services enable direct interaction with the thing by accessing the *Resources* of the associated *Devices*. Based on a priori knowledge.	Internal
Resource	Computational element that gives access to information about or actuation capabilities on a *Physical Entity*.	Internal
Requirement	“A quantitative statement of business need that must be met by a particular *architecture* or work package.”	[TOGAF9]
RFID	“The use of electromagnetic or inductive coupling in the radio frequency portion of the spectrum to communicate to or from a tag through a variety of modulation and encoding schemes to uniquely read the *identity* of an RF Tag.”	[ISO/IEC 19762]
Self-* properties	Self-* properties subsumes the desired capabilities of information systems to be self-configuring, self-organizing, self-managing, and self-repairing.	Internal
Semantic Web	"The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation."	[BERNERS-LEE2001]
Sensor	A sensor is a special *Device* that perceives certain characteristics of the real world and transfers them into a digital representation.	Internal
Security	"The correct term is 'information security' and typically information security comprises three component parts: 1.) Confidentiality. Assurance that information is shared only among authorised persons or organisations. Breaches of confidentiality can occur when data is not handled in a manner appropriate to safeguard the confidentiality of the information concerned. Such disclosure can take place by word of mouth, by printing, copying, e-mailing or creating documents and other data etc.;2.) Integrity. Assurance that the information is authentic and complete. Ensuring that information can be relied upon to be sufficiently accurate for its purpose. The term 'integrity' is used frequently when considering information security as it represents one of the primary indicators of information security (or lack of it). The integrity of data is not only whether the data is 'correct', but whether it can be trusted and relied upon; 3.) Availability. Assurance that the systems responsible for delivering, storing and processing information are accessible when needed, by those who need them.	[ISO27001]
Service	"Services are the mechanism by which needs and capabilities are brought together"	[OASIS-RM]
Service Choreography	Service Choreography is a form of Service Composition in which the participating services interact without being coordinated by a central component. The messaging schema between the elementary services is defined from a global point of view outside the involved services.	Internal
Service Composition	Service Composition denotes the composition of loosely coupled elementary services to form a higher-order composite service to provide additional functionality by re-using existing functionality. IoT-A focuses on hierarchical service composition that follows a black box approach keeping the composition implementation opaque to service consumers.	Internal
Service Composition	A Service Composition Model is a formal graphical or textual representation	Internal

Model	of a Service Composition.	
Service Orchestration	Service Orchestration is a form of Service Composition in which the coordination of the involved services is performed by a central component. This component is called the orchestrator and defines the messaging schema, needed to fulfil the composite service.	Internal
Stakeholder (also referred to as system stakeholder)	“An individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a *system*.”	[IEEE-1471-2000]
Storage	Special type of *Resource* that stores information coming from *resources* and provides information about *Entities*. They may also include *services* to process the information stored by the *resource*. As *Storages* are *Resources*, they can be deployed either on-device or in the network.	Internal
System	“A collection of components organized to accomplish a specific function or set of functions.”	[IEEE-1471-2000]
Tag	Label or other physical object used to identify the *Physical Entity* to which it is attached.	Internal
Thing	Generally speaking, any *physical object*. In the term ‘*Internet of Things*’ however, it denotes the same concept as a *Physical Entity*.	Internal
Unconstrained Network	An unconstrained network is a network of devices with no restriction on capabilities such as storage, computing power, and / or transfer rate.	Internal
User	A *Human* or any *Active Digital Entity* that is interested in interacting with a particular physical object.	Internal
View	“The representation of a related set of concerns. A *view* is what is seen from a *viewpoint*. An architecture *view* may be represented by a model to demonstrate to stakeholders their areas of interest in the architecture. A *view* does not have to be visual or graphical in nature”.	[TOGAF 9]
Viewpoint	“A definition of the perspective from which a *view* is taken. It is a specification of the conventions for constructing and using a *view* (often by means of an appropriate schema or template). A *view* is what you see; a *viewpoint* is where you are looking from - the vantage point or perspective that determines what you see”.	[TOGAF 9]
Virtual Entity	Computational or data element representing a *Physical Entity*. Virtual Entities can be either Active or Passive *Digital Entities*.	Internal
Web service	“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”	[W3C Web Services Glossary]
Wireless communication technologies	“Wireless communication is the transfer of information over a distance without the use of enhanced electrical conductors or "wires". The distances involved may be short (a few meters as in television remote control) or long (thousands or millions of kilometres for radio communications). When the context is clear, the term is often shortened to "wireless". Wireless communication is generally considered to be a branch of telecommunications.”	[Wikipedia WI]
Wireline communication technologies	“A term associated with a network or terminal that uses metallic wire conductors (and/or optical fibres) for telecommunications.”	[setzer-messtechnik2010]
Wireless Sensors and Actuators Network	“Wireless sensor and actuator networks (WSANs) are networks of nodes that sense and, potentially, control their environment. They communicate the information through wireless links enabling interaction between people or computers and the surrounding environment.”	[OECD2009]

References

Active Endpoints. 2012. ActiveVOS. http://www.activevos.com/products/activevos.

Active Endpoints, Adobe Systems, BEA Systems, IBM, Oracle, and SAP. 2007. WS-BPEL Extension for People. http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/30c6f5b5-ef02-2a10-c8b5-cc1147f4d58c.

Activiti. 2012. Activiti BPM Platform. http://activiti.org/.

Apache Incubator Project. 2012. Apache HISE. http://incubator.apache.org/hise/.

Avizienis, Algirdas, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. "Basic Concepts and Taxonomy of Dependable and Secure Computing." *IEEE Trans. Dependable Secur. Comput.* 1 (1): 11-33. http://dx.doi.org/10.1109/TDSC.2004.2.

Barros, Alistair, Marlon Dumas, and A H M Ter Hofstede. 2005. Service interaction patterns. In *In Proceedings 3rd International Conference on Business Process Management 3649*, 302-318. Nancy, France: Springer. http://www.springerlink.com/index/8NHCTR5QMY7EQUFY.pdf.

Bartonitz, Dr. Martin. 2009. Historische Entwicklung wichtiger Standards im Business Process Management. http://de.wikipedia.org/w/index.php?title=Datei:Standards_BPM_2009_11.jpg&filetimestamp=20100702012135.

Bonitasoft. 2012. Bonitasoft Open Source Workflow & BPM software. http://www.bonitasoft.com/.

Dar, Kashif, Amirhosein Taherkordi, Romain Rouvoy, and Frank Eliassen. 2011. Adaptable service composition for very-large-scale internet of things systems. In , 2:1–2:6-2:1–2:6. New York, NY, USA: ACM. http://doi.acm.org/10.1145/2093190.2093192.

De, Suparna, Payam Barnaghi, Martin Bauer, and Stefan Meissner. 2011. Service Modelling for the Internet of Things. *Federated Conference on Computer Science and Information Systems*. Szczecin.

Dessislava, and Aleksandar Dimov. 2011. Towards a Taxonomy of Web Service Composition Approaches. In *Workshops on Software Services*. doi:citeulike-article-id:9844762.

Eclipse Foundation. 2012. Eclipse. http://www.eclipse.org/.

Freund, Jakob, Bernd Rücker, and Thomas Henninger. 2010. *Praxishandbuch BPMN: Incl. BPMN 2.0*. Carl Hanser Verlag GmbH & CO. KG.

Glombitza, Nils, Dennis Pfisterer, and Stefan Fischer. 2010. "A Comprehensive Approach to Integrating Sensor Networks and Enterprise IT." *International Journal of Next-Generation Computing* 1 (1).

Grossmann, Georg, Rajesh Thiagarajan, Michael Schrefl, and Markus Stumptner. 2011. Conceptual Modeling Approaches for Dynamic Web Service Composition The Evolution of Conceptual Modeling. In , ed. Roland Kaschek and Lois Delcambre, 6520:180-204. Springer Berlin / Heidelberg. doi:10.1007/978-3-642-17505-3_9. http://dx.doi.org/10.1007/978-3-642-17505-3_9.

Hagedorn, Pascal, Carsten Magerkurth, Sonja Meyer, Stephan Haller, Klaus Sperner, Edward Ho, Alfonso Tierno, et al. 2011. Project Deliverable D7.1 – Initial definition of Use Cases 1/2.

Haller, Stephan, Stamatis Karnouskos, and Christoph Schroth. 2008. "The Internet of Things in an enterprise context." Ed. John Domingue and Paolo Traverso. *Future Internet–FIS 2008*: 14-28. http://www.alexandria.unisg.ch/export/DL/46662.pdf.

de las Heras, Ricardo, Martin Bauer, Nuno Santos, Nils Gruschka, Gerd Voelksen, Benoit Christophe, Sameh Ben Fredj, et al. 2012. Project Internal Report D4.1 Concepts and Solutions for Identification and Lookup of IoT Resources.

Issarny, Valérie, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadis, Marco Autili, Marco Aurélio Gerosa, and Amira Ben Hamida. 2011. “Service-oriented middleware for the Future Internet: state of the art and research directions.” *J. Internet Services and Applications* 2 (1): 23-45. http://dblp.uni-trier.de/db/journals/jisa/jisa2.html#IssarnyGHZVAGH11.

Kephart, J O, and D M Chess. 2003. “The vision of autonomic computing.” *Computer* 36 (1): 41-50.

Kopp, Oliver, Frank Leymann, and Sebastian Wagner. 2011. “Modeling Choreographies: BPMN 2.0 versus BPEL-based Approaches author.” *Proceedings of the International Workshop on Enterprise Modelling and Information Systems Architectures - EMISA 2011*.

Martin, Gregorio, Matthias Thoma, Dan Dobre, and Stefan Meissner. 2012. Project Deliverable D2.1 – Resource Description Specification.

Mayer, W, R Thiagarajan, and M Stumptner. 2009. Service Composition as Generative Constraint Satisfaction. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, 888-895.

Meyer, Sonja, Klaus Sperner, Carsten Magerkurth, Stefan Debortoli, and Matthias Thoma. 2012. IoT-A Project Deliverable D2.2 – Concepts for Modeling IoT-Aware Processes.

Meyer, Sonja, Klaus Sperner, Carsten Magerkurth, and Jacques Pasquier. 2011. Towards Modeling Real-World Aware Business Processes. In *Proceedings of the Second International Workshop on Web of Things*, 1-6. ACM. http://www.webofthings.com/wot/2011/papers/Pervasive2011.Workshop09.Paper08.pdf.

Object Management Group. 2011. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/.

———. 2012. BPMN Supporters. http://www.omg.org/bpmn/BPMN_Supporters.htm.

Pautasso, Cesare, Thomas Heinis, and Gustavo Alonso. 2005. Autonomic Execution of Web Service Compositions. In , 435-442.

———. 2006. “JOpera: Autonomic Service Orchestration.” *IEEE Data Eng. Bull.* 29 (3): 32-39. ftp://ftp.research.microsoft.com/pub/debull/A06sept/pautasso-heinis-alonso1.ps.

Peltz, Christian. 2003. “Web services orchestration and choreography.” *Computer* 36 (10): 46-52. doi:10.1109/MC.2003.1236471. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1236471.

Rode, Jochen, and Markus Küstner. 2011. *Real World Integration Platform*. Berlin, Heidelberg: Springer-Verlag.

Rosenberg, Florian. 2009. “Dissertation – QoS-Aware Composition of Adaptive Service-Oriented Systems.” *Computing*: 203. http://www.craax.ctvg.upc.es/papers/2009/phd-thesis-rene.pdf.

Russell, Nick, A H M Ter Hofstede, W M P van der Aalst, and N Mulyar. 2006. “Workflow controlflow patterns: A revised view.” *BPM Center Report BPM-06-22*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.6974.

Santos, Ivo José Garcia Dos, and Edmundo Roberto Mauro Madeira. 2006. “Applying orchestration and choreography of web services on dynamic virtual marketplaces.” *International Journal of Cooperative Information Systems* 15 (1): 57-85. http://www.worldscinet.com/abstract?id=pii:S0218843006001281.

Signavio. 2012. BPM + SaaS by Signavio. Business Process Management. http://www.signavio.com/.

Silver, Bruce. 2011a. *BPMN Method and Style, 2nd Edition, with BPMN Implementer’s Guide*. Aptos, CA: Cody-Cassidy Press.

———. 2011b. Executable BPMN 2.0. *BPMS Watch*. http://www.brsilver.com/2011/11/07/executable-bpmn-2-0/.

Sperner, Klaus, Sonja Meyer, and Carsten Magerkurth. 2011. Introducing Entity-based Concepts to Business Process Modeling. Ed. Remco Dijkman, Jörg Hofstetter, and Jana Koehler. *Business Process Model and Notation, Third International Workshop, BPMN 2011*. Lucerne, Switzerland: Springer-Verlag.

Strohbach, M, and M Martin. 2011. “Toward a Platform for Pervasive Display Applications in Retail Environments.” *Pervasive Computing, IEEE* 10 (2): 19-27.

Strohbach, Martin, Stefan Meissner, Claudia Villalonga, Fernando López, Frederic Montagut, Vincent Huang, Harald Kornmayer, and Jochen Bauknecht. 2010. D2.6 Sensor Information Services with Open Service Interfaces.

Teixeira, Thiago, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. 2011. Service oriented middleware for the internet of things: a perspective. In , 220-229. Berlin, Heidelberg: Springer-Verlag. http://dl.acm.org/citation.cfm?id=2050869.2050893.

Villalonga, Claudia, Martin Bauer, Martin Strohbach, Jochen Bauknecht, Harald Kornmayer, Stefan Meissner, Vincent Huang, et al. 2010. D2.5 Adaptive and Scalable Context Composition and Processing.

W3C. 2004. Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

Walewski, Joachim, Martin Bauer, Nicola Bui, Pierpaolo Giacomin, Nils Gruschka, Stephan Haller, Edward Ho, et al. 2011. Project Deliverable D1.2 – Initial Architectural Reference Model for IoT.

Weske, Mathias. 2007. *Business process management: concepts, languages, architectures*. *Business Process Management*. Springer-Verlag New York Inc. http://www.springerlink.com/index/9YH5WYAWLWV20UAE.pdf.

White, Stephen A, and Derek Miers. 2008. *BPMN Modeling and Reference Guide*. Lighthouse Pt, FL: Future Strategies Inc.

Wikipedia. 2012. Comparison of BPEL engines. http://en.wikipedia.org/wiki/Comparison_of_BPEL_engines.

Workflow Management Coalition. 2012. XPDL Support and Resources. http://www.wfmc.org/xpdl.html.

Zaha, J, Alistair Barros, Marlon Dumas, and A H M Ter Hofstede. 2006. “Let’s Dance: A Language for Service Behavior Modeling.” *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*: 145-162. http://www.springerlink.com/index/p2744227xq966130.pdf.

Zeng, Liangzhao, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. 2004. “QoS-Aware Middleware for Web Services Composition.” *IEEE Trans. Softw. Eng.* 30 (5): 311-327. http://dl.acm.org/citation.cfm?id=987527.987638.

jBoss Community. 2012. jBPM. http://www.jboss.org/jbpm.