

Rapid Application Development in the Internet of Things: A Model- Based Approach

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Ferry Pramudianto aus Jakarta, Indonesien

Berichter: Universitätsprofessor Prof. Dr. rer. pol. Matthias Jarke
Universitätsprofessor Prof. Djamel Sadok, Ph.D.

Tag der mündlichen Prüfung: 25. Februar 2015

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Abstract

The Internet of Things (IoT) is a vision in which physical and digital objects are connected and cooperate to achieve particular goals. Unfortunately, the extent of expertise required to incorporate intelligent hardware, software, and computer network still presents a significant challenge. Service-oriented IoT middleware have been proposed quite often to solve this problem. However, they are mostly designed for professional developers with a high degree of flexibility and extensive features. Consequently, tool's simplicity is often sacrificed, and they present a steep learning curve for entry-level developers. This dissertation aims at addressing this gap by elaborating the state-of-the-art in IoT developments and proposing IoTLink, a rapid IoT software development tool for novice developers.

For designing IoTLink, the author reviewed the available IoT architectures. A typical pattern suggests that a physical object must be uniquely identifiable, has physical qualities that partly can be sensed by sensors, and has some capabilities or services that could affect the environment. Virtual entities may act as proxies to execute services and retrieving information about the physical objects. IoTLink is designed for enabling inexperienced developers to develop proxies representing domain objects and abstracting individual sensors and actuators. IoTLink design concept comprises a five layered architecture. The first layer is responsible for abstracting communication with heterogeneous data sources. The second layer deals with sensor fusion components to process and fuse sensor data into useful information. The third layer is concerned with the definitions of domain models and the concrete objects. The fourth layer provides output components, including interfaces to the application logic, distributed applications, and databases to store the information about the virtual objects. The fifth layer abstracts the application logic that access the domain objects. IoTLink employs a model driven approach for wiring these components visually. The visual model is then serialized into XML data and used to generate a Java implementation which can be executed as proxies. In addition, IoTLink offers a discovery broker allowing developers to share and discover IoT resources within the internet. The key advantage of IoTLink discovery is the ability to detect if similar devices are described with synonymous terms. This approach increases the discoverability of similar devices described with diverse terms.

The author evaluated the practicability of IoTLink and model-driven approach within three distinct case studies in European research projects. The result shows that it could reduce approximately 2/3 of the development efforts. In addition, the author compared IoTLink's usability to a Java middleware approach in a controlled experiment performed by 24 participants. The results show that IoTLink could on average reduce 44% of the development time and 48% of mistakes. Moreover, when used by developers with less than five years object-oriented experience, IoTLink was able to reduce up to 57% of mistakes compared to Java development.

Kurzfassung

Das Internet der Dinge (englisch: Internet of Things, IoT) ist eine Vision, in der physische und digitale Objekte miteinander verbunden sind und zusammenarbeiten, um bestimmte Ziele zu erreichen. Dabei stellt das Maß an notwendiger Expertise, um intelligente Hardware, Software und Rechnernetze zu integrieren, immer noch eine große Herausforderung dar. Zur Lösung dieses Problems wurden bereits oft service-orientierte Middleware vorgeschlagen. Allerdings sind sie meistens für professionelle Entwickler mit einem hohen Maß an Flexibilität und umfangreichen Funktionen konzipiert. Folglich wird bei diesen die Einfachheit oft geopfert, so dass sie von unerfahrenen Entwicklern bzw. Einsteigern eine steile Lernkurve erfordern. Diese Dissertation zielt darauf ab, diese Lücke durch Ausarbeitung aktueller IoT-Entwicklungsmethoden zu schließen und IoTLink vorzustellen, ein Rapid-Entwicklungswerkzeug für Einsteiger.

Zur Gestaltung von IoTLink untersucht der Autor existierende IoT-Architekturen. Typischerweise wird von einem physischen Objekt verlangt, dass es eindeutig identifizierbar ist, teilweise durch Sensoren erkennbare physikalische Eigenschaften besitzt und Funktionen bzw. Dienste hat, die dessen Umgebung beeinflussen können. Zur Ausführung von Funktionen bzw. Diensten eines physischen Objekts bzw. um Information über dieses abzurufen, können virtuelle Objekte als Stellvertreter (engl. Proxy) verwendet werden. IoTLink ist so konzipiert, um unerfahrenen Entwicklern die Entwicklung von Proxys zu ermöglichen, die Objekte einer bestimmten Domäne darstellen und von einzelnen Sensoren und Aktoren abstrahieren. Das IoTLink-Designkonzept besteht aus einer Fünf-Schichten-Architektur. Die erste Schicht ist für das Abstrahieren der Kommunikation mit heterogenen Datenquellen verantwortlich. Die zweite Schicht beschäftigt sich mit Sensorfusion-Komponenten, die für die Umwandlung von Sensordaten in nützliche Informationen zuständig sind. Die dritte Schicht beschäftigt sich mit der Definition von Domänenmodellen und mit konkreten Objekten. Die vierte Schicht liefert Ausgabekomponenten und Schnittstellen zu Anwendungslogik, verteilten Anwendungen und Datenbanken zur Speicherung der Daten bzw. Informationen. Die fünfte Schicht abstrahiert die Anwendungslogik, die auf die Domänen-Objekte zugreift. IoTLink verwendet einen modellgetriebenen Ansatz um diese Komponenten visuell miteinander zu verbinden. Das visuelle Modell wird dann in XML-Daten serialisiert und in eine Java-Implementierung umgewandelt, die als Proxys ausgeführt werden können. Zusätzlich bietet IoTLink eine Komponente an, die Entwicklern das Teilen und das semantische Suchen von IoT-Ressourcen im Internet ermöglicht. Der entscheidende Vorteil dieser Komponente ist die Fähigkeit zur Erkennung ob ähnliche Geräte mit unterschiedlichen, aber synonymen Begriffen beschrieben werden, was die Auffindbarkeit solcher ähnlicher Geräte erhöht.

Der Autor untersucht die Praktikabilität von IoTLink und dem modellgetriebenen Ansatz anhand drei verschiedener Fallstudien im Rahmen von europäischen Forschungsprojekten. Das Ergebnis zeigt, dass etwa zwei Drittel des Entwicklungsaufwands reduziert werden konnten. Außerdem wird IoTLink hinsichtlich dessen Gebrauchstauglichkeit einem Vergleich mit einem Java-Middleware-Ansatz in einem kontrollierten Versuch mit 24 Teilnehmern

unterzogen. Die Ergebnisse zeigen, dass IoTLink im Durchschnitt die Entwicklungszeit um 44% und die Fehlerquote um 48% reduzieren konnte. Bei Entwicklern mit weniger als fünf Jahren Erfahrung in objektorientierter Entwicklung konnte die Fehlerquote sogar um bis zu 57% im Vergleich zur Java-Entwicklungsumgebung reduziert werden.

Acknowledgements

I would like to thank my first supervisor, Prof. Dr. Matthias Jarke from the RWTH Aachen University for his guidance and invaluable advises, and my second supervisor Prof. Djamel F.H. Sadok Ph.D. from the Federal University of Pernambuco for his guidance, as well as my advisor Prof. Dr. Carlos Alberto Kamienski from the Federal University of ABC for his advises and his continuous support in publications.

I express my warm thanks to Dr. Markus Eisenhauer and my colleagues in User-Centered Ubiquitous Computing group for their support and guidance and of all the people who provided me with the facilities being required and conducive conditions for my doctoral research at the Fraunhofer Institute for Applied Science.

I would like to thank you Henny Liwan M.D. and Dr.rer.nat Wong Kariato who have helped me translating the abstract of this work to German.

I would also like to use this opportunity to express my gratitude to everyone who supported me throughout the course of this doctoral research. I am thankful for their aspiring guidance, invaluable constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

I would also express my greatest appreciation to my wife, parents, and sisters who have continually provided their moral supports for completing this research work.

Bonn, the 1st of November 2014

Ferry Pramudianto

Table of Contents

Abstract	i
Kurzfassung	ii
Acknowledgements	v
Table of Contents	vi
Abbreviations	xi
List of Figures	xiii
List of Tables	xix
Chapter 1. Introduction	1
1.1 Problem Statement	2
1.2 Research Questions	3
1.3 Contributions.....	4
1.4 Thesis Statement	5
1.5 Assumption	5
1.6 Overall Methodology	6
1.7 Document Outline	7
Chapter 2. The Notions of the Internet of Things	9
2.1 Definition	9
2.2 The Notion of “Things” within the IoT	11
2.3 Context and Scope of IoT	13
2.3.1 IoT related and enabling technologies	15
2.4 Conclusion	16
Chapter 3. IoT Application Domains and Related Technology	19
3.1 Industrial Automation	19
3.1.1 Interoperability	20

TABLE OF CONTENTS

3.1.2	Industrial WSN.....	23
3.2	Product Traceability.....	25
3.2.1	RFID for Product Tagging	25
3.3	Building and Home Automation.....	27
3.4	Smart Grid.....	30
3.4.1	Communication in Smart Grid	31
3.5	Conclusion and Common Requirements towards IoT	33
Chapter 4. State-of-the-art in IoT Architecture and Development Platforms		37
4.1	IoT Architectures	37
4.1.1	Communication between Things and the Internet.....	40
4.1.2	Functional view of IoT architecture	46
4.1.3	Middleware for IoT	48
4.1.4	Semantic Discovery.....	52
4.2	Model Driven Development	54
4.2.1	Embedded System Software Modeling	57
4.2.2	MDD drawbacks	59
4.2.3	Mashup development	59
4.2.4	Web of Things.....	60
4.3	Summary and Conclusion	62
Chapter 5. Design Concept and Technical Implementation of IoTLink		65
5.1	Requirement elicitation.....	66
5.1.1	Scenario and Personas	67
5.1.2	IoTLink Conceptual Design	70
5.1.3	IoTLink Workflow Design.....	76
5.1.4	IoTLink User Interface Design	78
5.1.5	IoTLink Implementation	79
5.2	Conclusion	111
Chapter 6. Sharing and Discovering IoT Semantically		113
6.1	Device Discovery.....	113
6.2	Semantic Device Description.....	116
6.2.1	Identification	117

TABLE OF CONTENTS

6.2.2	Sensors and Actuators Categorization.....	117
6.2.3	The design of the Discovery Manager knowledge base.....	122
6.2.4	Adding arbitrary device attributes at runtime	125
6.2.5	Web Service interface for the applications	126
6.3	Using SSN Ontology and WordNet.....	127
6.4	Conclusion	136
Chapter 7.	Case Studies	139
7.1	Evaluation methodology	139
7.2	Case studies.....	140
7.2.1	SEEMPubS: Building automation.....	140
7.2.2	ebbits: Enabling traceability in meat production.....	145
7.2.3	BEMOCOFRA: Monitoring a flexible manufacturing prototype.....	150
7.3	Conclusion and Lesson learned	157
Chapter 8.	IoTLink Formal Evaluation	161
8.1	Initial formal evaluations	161
8.1.1	Usability test of the initial IoTLink.....	162
8.2	The Final Usability Test of IoTLink.....	167
8.2.1	Usability Study Design.....	167
8.2.2	Final Evaluation Results.....	173
8.3	Summary and conclusion.....	182
Chapter 9.	Conclusion and Future Work	185
9.1	Summary of contributions.....	185
9.2	Threats to validity	189
9.3	Future works	190
9.4	Outlook of IoT developments	191
Bibliography		195
Appendix 1.	Own Publications	209
	Publications related to this dissertation.....	209
	Other Publications.....	210
Appendix 2.	Comprehension Study Tasks	211

TABLE OF CONTENTS

Appendix 3. Programming Tasks	217
Appendix 4. Post-Study Questionnaire.	219
Appendix 5. After-Scenario Questionnaire.	223
Appendix 6. Curriculum Vitae	225

Abbreviations

<i>Abbr.</i>	<i>Meaning</i>	<i>Abbr.</i>	<i>Meaning</i>
6LowPAN	IPv6 over Low power Wireless Personal Area Network	FBP	Flow Based Programming
AAA	Authentication, Authorization, and Accounting	FCAPS	Fault, Configuration, Accounting, Performance, and Security
AMQP	Advanced Message Queuing Protocol	FDD	Full Function Device
AODV	Ad-hoc On-Demand Distance Vector	FPGA	Field-Programmable Gate Array
ARM	Architecture Reference Model	HART	Highway Addressable Remote Transducer
ASQ	After-Scenario Questionnaire	HF	High Frequency
B2MML	Business to Machine Mark-up Language	HVAC	Heating, Ventilation, and Air Conditioning
BMS	Building Management system	ICT	Information and Communications Technology
BPMN	Business Process Model and Notation	ID	Identification
CANBus	Controller Area Network	IERC	The Internet of Things European Research Cluster
CE2E	Communication End-to-End	IIP	Intelligent Information Processing
CEP	Complex Event Processing	IoT	Internet of Things
CIM	Computation Independent Model	ITU	International Telecommunication Union
CoAP	Constrained Application Protocol	LF	Low Frequency
CPS	Cyber Physical System	LTE	4G Long-Term Evolution
DCOM	Distributed Component Object Model	M2M	Machine-to-Machine communication
DPWS	Devices Profile for Web Services	MDA	Model Driven Architecture
DRTC	Distributed Real Time Control	MDD	Model driven development
DSL	Domain Specific Language	MEMS	Micro-Electro-Mechanical System
DSO	Distribution System Operator	MES	Manufacturing Execution System
EPC	Electronic Product Code	MQTT	Message Queuing Telemetry Transport
EPL	Event Processing Language	NB	Narrow Band
ERP	Enterprise Resource Planning System	OLE	Object Linking and Embedding

ABBREVIATIONS

<i>Abbr.</i>	<i>Meaning</i>	<i>Abbr.</i>	<i>Meaning</i>
OMG	Object Management Group	RFID	Radio-Frequency Identification
ONS	Object Naming Service	SCADA	Supervisory Control and Data Acquisition
OPC	OLE for Process Control	SCM	Supply Chain Management
OPC-DA	OPC Data Access	SOA	Service-oriented Architecture
OPC-HDA	OPC Historical Data Access	SOAP	Simple Object Access protocol
OPC-UA	OPC Unified Architecture	SSN	Semantic Sensor Network
P2P	Peer-to-Peer	Synset	Synonym set
PIM	Platform Independent Metadata	UCD	User-Centered Design
PLC	Power Line Communication (in Smart Grid Context)	UHF	Ultra-High Frequency
PLC	Programmable Control Logic (in industrial automation Context)	UNB	Ultra Narrow Band
PLM	Product Lifecycle Management	VAS	Value Added Service
PSM	Platform Specific Metadata	WoT	Web of Things
PSSUQ	Post-Study System Usability Questionnaire	WSAN	Wireless Sensor and Actuator Network
RDF	Reduced Function Device	WSN	Wireless Sensor Network
REST	Representational State Transfer	XMI	XML Metadata Interchange

List of Figures

Figure 1. The 2014 Emerging Technologies Hype Cycle.....	1
Figure 2. The envisioned prototyping toolkit for IoT	5
Figure 3. Research Methodology	6
Figure 4. Distributed computing evolution to the IoT (Perera et al., 2014).....	10
Figure 5. Types of devices and their relationship with physical things (ITU-T, 2012).....	13
Figure 6. The relation between IoT and related fields such as M2M, CPS, and WSN (M. Chen et al., 2012)	14
Figure 7. Technologies related to IoT research and development (Lee et al., 2013).....	15
Figure 8. Distributed control architecture in factory automation.....	20
Figure 9. OPC Architecture.....	21
Figure 10. (a) Plant information model according to MESA (International, 1997) and (b) hierarchical enterprise model according to ISA-95 and the scope of the standards (Sauter, 2007)	22
Figure 11. Typical WirelessHART deployment architecture (Gao et al., 2013).	24
Figure 12. Overview of ZigBee network(Baronti et al., 2007).....	24
Figure 13. RFID tag attached to the ear of a pig (Wasserman, 2009).....	26
Figure 14. An example of Siemens Desigo consisting of various network standards, including KNX, BACnet, and LonWorks.	27
Figure 15. Comparison of wireless technology used in home and building automation (Rathnayaka et al., 2011).	28
Figure 16. Possible solutions to integration existing BMS to IoT (Jung et al., 2012).	29
Figure 17. Smart Grid vision.....	31
Figure 18. An example of monthly smart meter traffics sent to the DSO by the aggregators (Wenpeng et al., 2013).....	32
Figure 19. Comparison of wireless technologies for smart grid (Parikh et al., 2010)	33
Figure 20. EPCglobal Architecture(Shih et al., 2005)	38
Figure 21. The IoT-A Architecture Reference Model (ARM).....	39
Figure 22. The seven OSI layers	40

LIST OF FIGURES

Figure 23. IoT communication layers according to IoT-A (IoT-A, 2013).....	41
Figure 24. Gateway translating the lower layer protocols (IoT-A, 2013).....	42
Figure 25. A Gateway that translates the lower layer protocols by piggy backing the content into a protocol in a different layer (IoT-A, 2013).	42
Figure 26. M2M Network Architecture	43
Figure 27. Publish-subscribe pattern involving two clients and a service (IoT-A, 2013)	44
Figure 28. MQTT-S Architecture to enable communications between WSNs (Hunkeler et al., 2008)	45
Figure 29. IoT reference model (ITU-T, 2012).....	46
Figure 30. IoT Reference Functional View (IoT-A, 2013)	48
Figure 31. Generic Layered Architecture for IoT (D. Bandyopadhyay & Sen, 2011)	49
Figure 32. Classification of middleware approaches for wireless sensor network (Hadim & Mohamed, 2006)	49
Figure 33. SOA-based architecture for the IoT middleware (Atzori et al., 2010)	50
Figure 34. LinkSmart Architecture	51
Figure 35. Relations between device, application, and Mashup services as envisioned by the W3C (Lefort et al., 2011)	53
Figure 36. Elevating abstractions in software development (Mellor, 2004).....	55
Figure 37. The transformation from the CIM, PIM and PSM [adapted from (M. OMG, 2003)].	56
Figure 38. PID control algorithm represented in a graphical software view (Gretlein, 2013).....	57
Figure 39. The evolution of programming languages for PLCs (Frey & Litz, 2000).....	58
Figure 40. An example of ladder logic used for programming Siemens PLCs	58
Figure 41. An example of a Yahoo! Pipes service for querying the Amazon Web Service.	60
Figure 42. ThingWorx, a centralized platform for analyzing data from IoT	61
Figure 43. User-centered design process.....	66
Figure 44. Example of a persona card used during the design of IoTLink	69
Figure 45. Internet of Things Metamodel simplified from (IoT-A, 2013).	71
Figure 46. Abstraction levels of IoT, which hides sensors and actuators within domain objects mapped to the IoT layered architecture (Pramudianto, Rusmita, et al., 2013).....	72

LIST OF FIGURES

Figure 47. IoTLink Development Concept	76
Figure 48. The mock-up user interface for the development tool (Rusmita, 2012).....	78
Figure 49. Illustration of Eclipse plug-ins used to build IoTLink.....	80
Figure 50. Logical view of the Platform-Independent Metamodel (PIM) for IoTLink (the properties of the classes are omitted to simplify the picture).....	82
Figure 51. The first iteration of the model development tool	83
Figure 52. The final iteration of the model development tool	83
Figure 53. The setting of the project to be generated on the property sheet.....	84
Figure 54. The connection components	85
Figure 55. Arduino boards which can be extended to create a more complex sensor and actuator.....	85
Figure 56. Wizard for selecting the Web Service method	86
Figure 57. Plugwise devices use ZigBee for enabling wireless monitoring and automation in homes and commercial buildings	87
Figure 58. Supported Sensor Fusion Modules	89
Figure 59. A mesh of sensor fusion modules.....	89
Figure 60. An example of EPL query of a moving window average (a) and sum of the sensor values based on the timestamp (b).....	90
Figure 61. Configuring Esper engine through EPL in the property sheet of the Esper Engine component.....	91
Figure 62. Defining the sensor fusion algorithms in Java.....	91
Figure 63. Components to build the objects.....	92
Figure 64. Defining class template.....	92
Figure 65. Setting the class of an object.....	93
Figure 66. An example of the URL generated by the RESTOutput component to access a virtual object	96
Figure 67. REST output for retrieving the values of the objects.....	98
Figure 68. Virtual object schema generated by IoTLink	99
Figure 69. Drools rule language format	101
Figure 70. Defining rules, responding to the state of the physical objects.	102
Figure 71. An example of domain model serialization in XMI format.....	104
Figure 72. Example of the generated Java project.	104

LIST OF FIGURES

Figure 73. Interaction diagram between the generated classes for updating properties.	107
Figure 74. Deployment artifacts for redistributing the prototype on another hardware platform.....	110
Figure 75. An example of the Web output which could be used for debugging the application.....	111
Figure 76. Architecture of the semantic Discovery Manager.	114
Figure 77. Interaction between target service and client in two different modes of WS-Discovery(Modi & Kemp, 2009)	115
Figure 78. Device description that must be carried by the proxy of the device (Pramudianto et al., 2014).	117
Figure 79. The role of ontologies within an IoT middleware adapted from (Hachem et al., 2011).....	118
Figure 80. Partial illustration of the Automatic Weather Station ontology (Barnaghi et al., 2011).	119
Figure 81. Overview of the SSN Ontology classes and properties (W3C, 2011).....	120
Figure 82. Enumeration of Measurement Properties in the SSN Ontology (W3C, 2011).....	122
Figure 83. Linking devices in the ontology (Pramudianto et al., 2014).....	124
Figure 84. The logic used to homogenize terminology used for describing devices (Pramudianto et al., 2014).	125
Figure 85. The administration tool to modify the synsets (Pramudianto et al., 2014).	126
Figure 86. Stimulus-Sensor-Observation pattern (Stasch et al., 2009).	127
Figure 87. Ontology mapping between the SSN ontology & Base Ontology.....	128
Figure 88. An example of instantiating SSN ontology for describing an Observation.....	129
Figure 89. Workflow used to build the device type taxonomy.	131
Figure 90. Examples of some synsets, hyponym, hypernym of "light" exposed by WordNet	132
Figure 91. Inserting the capability of devices and linking to available concepts.....	133
Figure 92. SEEMPubS Architecture(Osello et al., 2013)	141
Figure 93. Domain model of the SEEMPubS system	143
Figure 94. Linking the power meter that measure the energy consumptions and actuators in the two single offices	143
Figure 95. The lifecycle of pork meat product (Udsen et al., 2010)	145

LIST OF FIGURES

Figure 96. The architecture of data acquisition through the ebbits platform (Madsen et al., 2013).	147
Figure 97. Domain model for recording livestock feed.	148
Figure 98. DigiCow database which is accessible through REST (left) and generated link encoded in QRCode (right).....	149
Figure 99. Flexible manufacturing testbed at COMAU's site	150
Figure 100. The iPad user interface for monitoring a manufacturing line by a line manager.....	151
Figure 101. Architecture of the monitoring application.....	152
Figure 102. Classes used to represent the entities in the domain.	153
Figure 103. The concrete implementation model where instances of virtual objects are linked to sensor fusion modules and connections to the sensors.	154
Figure 104. Database schema generated by the DatabaseOutput component.....	156
Figure 105. Illustration of UML's class diagram notation in EcoreTool (A) compared to the DSL notation in IoTLink (B) (Pramudianto, Rusmita, et al., 2013)	164
Figure 106. Comparison between EMF tool & IoT Modeling Tool (Pramudianto, Rusmita, et al., 2013).....	164
Figure 107. Users' satisfaction of (A) Device developer's role and (B) The administrator's role (Avila, 2013).	166
Figure 108. Users' satisfaction of (A) Application developer's role and (B) The overall system (Avila, 2013).....	166
Figure 109. 3x3 Latin square design	169
Figure 110. The UML diagram used for the comprehension task.	169
Figure 111. The IoTLink diagram used for the comprehension task.	170
Figure 112. The source code used for the comprehension task.	170
Figure 113. A participant was doing a comprehension task.	171
Figure 114. Age distribution of the participants	172
Figure 115. Participants' experiences in Object-Oriented, UML, Network Programming and IoT.....	172
Figure 116. The average time required by the participants to answer the questions from three different presentations of the classes and objects.	173
Figure 117. The average mistakes found in the participants' answers.....	174
Figure 118. Total time required by the participants to perform all programming tasks (left) and the number of errors made by the participants (right).	176
Figure 119. Time required by participants to complete the tasks.	177

LIST OF FIGURES

Figure 120. Total time required by the participants to perform all programming tasks (left) and the number of errors made by the participants (right) categorized by the object-oriented experiences. 178

Figure 121. Participants' satisfactions post-study questionnaire with the complete questions. 180

Figure 122. Participants' satisfaction to IoTLink compared to Java in a rapid prototyping..... 181

List of Tables

Table 1. Scenario defined to summarize typical problems faced by the actors	67
Table 2. The requirements of the actors involved in the aimed scenario.....	69
Table 3. SOAPInput uses XPath to extract single data from a SOAP message.....	87
Table 4. An example of representing a virtual object with a SOAP-message.	95
Table 5. Response and request of the virtual object through the RESTOutput	96
Table 6. Example of customizing the generated code in Java and how to obtain the virtual objects.....	109
Table 7. Transitive inference of device types	123
Table 8. Example of searching device with keywords.....	135
Table 9. Summary of software engineering experiments classifications.	139
Table 10. Control strategy for non-dimming lighting	144
Table 11. Comparison of estimated efforts in project developments with and without IoTLink.....	158

Chapter 1.

Introduction

The internet is constantly evolving at a very rapid pace. In the last decade, the convergence of physical devices into the internet known as the Internet of Things (IoT) has drawn a tremendous interest from the industries and academia. Cisco (Evans, 2011b) and Ericsson (Ericsson-Australia, 2010) predicted that there are going to be 50 billion of “Things” connected to the internet by 2020. Gartner predicted that in 2020, the economic impact of IoT will be 1.9 trillion US dollars in different application domain, such as healthcare, logistics, and retail (Petty, 2013).

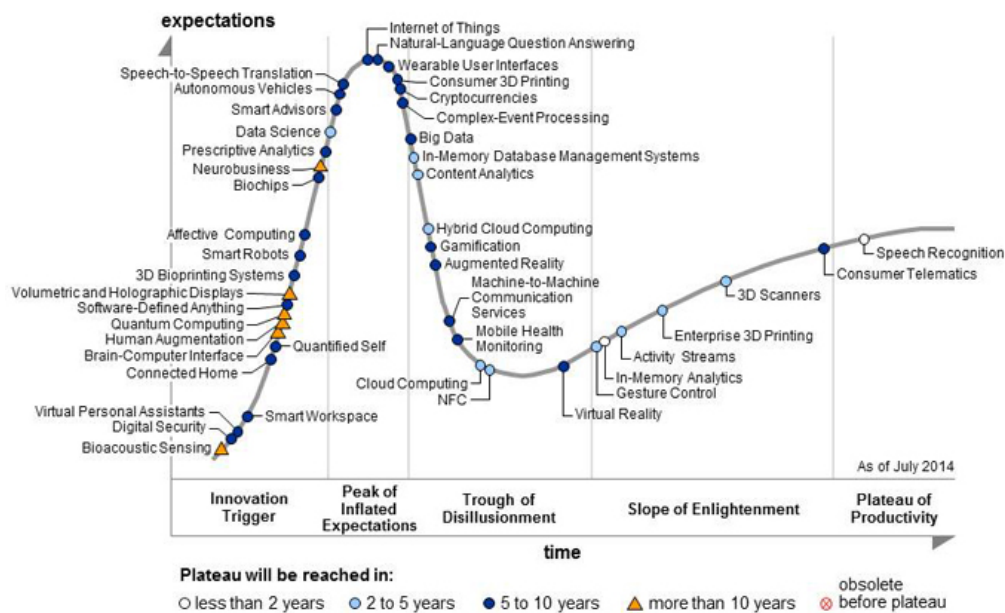


Figure 1. The 2014 Emerging Technologies Hype Cycle¹.

The IoT bears a vision of seamless interaction between millions of devices that are connected to the internet. This vision opens various possibilities, providing new integrated services that could ease our everyday lives. For instance, in the future, smart refrigerators could detect that milk and eggs were running out and would inform users’ smart phones. Allowing users’

¹ <https://www.gartner.com/newsroom/id/2819918> (retrieved on August 14, 2014)

phones to receive information from various smart devices at home, make it possible for it to compile a grocery list for them (Hanshen & Dong, 2009). In emergency response, victims of catastrophic events can be tagged by a digital triage system which allows the first responders to monitor their status, detect their locations, and prioritize their treatment through mobile devices (Jentsch et al., 2013). These examples have shown future scenarios that IoT could enable and the high expectations of the community for the future of IoT.

Gartner identified IoT as one of the emerging technology in the IT Hype chart which takes approximately 5-10 years to become a mainstream technology. However, its position in the IT Hype chart has increased from year to year that shows that the adoption, maturity, and popularity increases over time. In 2014, it has reached nearly the peak as companies have started publishing success stories adopting IoT technology such as Radio-frequency identification (RFID) for logistics and wireless sensors for process monitoring. Nonetheless, many potentials of IoT are still explored at an early stage, which makes it very heterogeneous in terms of the standards and maturity of the technology.

1.1 Problem Statement

While the number connected things are increasing rapidly, developing IoT system prototypes is still a complex task. It requires developers to deal with various technological challenges such as limited computing resources, heterogeneous communication technology and data format, processing sensors and actuators' signals in real-time, storing and analyzing enormous amount of data which is produced by devices. In addition, IoT development sometimes requires developers to master different programming languages. For instance, developing software for embedded controllers requires very efficient languages such as C. However, MDD tools are often used to generate a program to increase the productivity of the developers. In the enterprise development, where powerful hardware hosts the program, computing resources are not the primary issue. However, the complexity of the applications makes software maintainability more of a concern, and therefore software readability and object orientation are highly desired. Newer programming languages such as Java and C# are designed to fulfil this purpose. Consequently, they are more often used in the enterprise development. As highly specialized developers usually required addressing these challenges, it is particularly difficult for small development teams to develop IoT systems.

Furthermore, heterogeneity could be easily found when developing IoT systems since it overlaps with many application domains that already have well-established and diverse standards. For instance, industrial automation and building automation domain rely on diverse Fieldbus networks e.g., Profibus (Bender, 1993), Modbus (Modbus, 2004b), while Ethernet and Wi-Fi have become the communication standard between PCs. Many middleware solutions have been proposed to facilitate connectivity between heterogeneous devices. Between these solutions, service oriented (SOA) middleware offers an attractive solution to support horizontal and vertical integration since it has been widely adopted by business applications (Pramudianto, Khaleel, et al., 2013). Unfortunately, many of these solutions were designed for a broad range of use cases which come with a high price of complexity and steep learning curve especially for inexperienced developers (Blackstock & Lea, 2012). The existing tools for IoT development currently focus on supporting specific group of developers

such as embedded developers and enterprise application developers. Consequently, to create a simple IoT prototype, developers are required to combine different disintegrated tools. This approach is not very intuitive and error prone. As the author interviewed developers who were 2-4 years involved in several IoT research projects, the author found that they are always confronted with a lack of standardizations, immaturity of the technology, and integrated development tools which could support their productivity.

Similar to current network resources, IoT applications typically expect to gain access to large numbers of shared resources for cost saving. E.g., RT-WIS, sensor system to monitor water in Tasmania, Australia is being shared between several organizations that perform geological research activities (Liu et al., 2010). In addition, appropriate scalable mechanisms must be put in place, allowing a real-time discovery and dynamic binding to such devices. Traditionally in computer networks, devices are discovered, according to their roles in the network and the protocols they support. For instance, the address resolution protocol (ARP) (Plummer, 1982) has been used for discovering hosts on a local area network and learning about their physical and Internet Protocol (IP) addresses. At the device and service levels, mechanisms such as the Universal Plug-and-Play (UPnP) (Jeronimo & Weast, 2003) and Semantic Web Services (Klusck et al., 2006) have been supported. Device discovery has been investigated and applied in local or mobile networks intensively (B. A. Miller et al., 2001; Jeronimo & Weast, 2003; Ahamed et al., 2006; Klusck et al., 2006; Outay et al., 2007). However, in an IoT scenario a much larger network of things must be anticipated. In this case, diverse terminologies might be adopted to describe and search for devices, which could limit the discoverability of devices.

In summary, until this moment there is still a gap of extensive guides for IoT development. Secondly, the support for developers to share and discover devices semantically, which considers the term diversity, is a nonexistence. Thirdly, there is a lack of integrated IoT development tools, which can assist them in creating IoT applications rapidly.

1.2 Research Questions

The problems described in section 1.1 has motivated this work to perform research that can answer the following research questions:

- To what extent can the internet of thing architectures be abstracted?
 - a. What are the typical requirements of IoT systems?
 - b. Are there similarities between IoT architectures and technologies across domains?
 - c. How sensors and actuators should be abstracted to enable rapid software development for the IoT?
- How can applications discover “Things”?
 - a. How could “Things” be discovered, according to their semantics (e.g., types, capabilities, utilizations)?
 - b. How to overcome the terms diversity used to describe and search for Things?
- To what extent can a Model-Driven Development approach support IoT prototyping?
 - a. What kind of abstractions should be provided to enable rapid development?
 - b. Whether a model driven tool could accelerate IoT software development?
 - c. Whether the visual notations have a good understandability?

1.3 Contributions

This work aims at addressing these research questions, firstly, by presenting an extensive study of IoT developments in several domains such as manufacturing and intelligent buildings. Additionally, it synthesizes common requirements and architectural patterns that the author uses as a theoretical ground for guiding the implementation of IoT prototyping tools.

Secondly, to facilitate developers sharing and discovering devices on the internet, this work elaborates the state-of-the-art approaches in the device semantic discovery which then is used as a reference for implementing a semantic discovery component which allows developers to find devices according to his requirements such as the capability, precision, and accuracy of the devices. The implemented discovery management goes beyond the state-of-the-art by presenting an approach to overcome the possible use of diverse terms. It considers the lexical semantics of the terms used as the device descriptions and the terms used as a keyword to search them.

Thirdly, guided by the results of the study, this work proposes an IoT development toolkit, called IoTLink. IoTLink is intended to support inexperienced developers in developing IoT prototype rapidly. As the targeted user group is inexperienced developers, this dissertation evaluates MDD approaches that have shown quite promising results, enabling rapid prototyping in various application developments such as web applications (Nunes & Schwabe, 2006; Ceri et al., 2007) and pervasive computing applications (Cetina et al., 2007; Serral et al., 2008). Unfortunately, this approach has not been widely explored for IoT development. Applying MDD, IoTLink exploits a visual domain specific modeling language similar to Mashup development tools that have been claimed simple and easy to use for non-expert users (Grammel & Storey, 2008; Jin et al., 2008). Moreover, IoTLink's models can be transformed into Java, which provides higher flexibility to be extended by more experienced developers in a further phase of the development.

The evaluation of IoTLink is performed through case studies within the prototype developments of three European projects, SEEMPubS, BEMOCOFRA, and ebbits. These projects focus on quite different scenarios, providing a wide range of requirements and constraints. In addition, controlled experiments are performed to examine the users' comprehension of relations between domain objects that are presented in diagrams and source code. In the study, the usability aspects of IoTLink were also compared to the current approach used for creating IoT prototypes, which are done through textual programming languages.

In summary, the goal of the research can be decomposed into as follows:

- Develop an in-depth theoretical framework around the IoT architecture. The theoretical framework discusses the architectural patterns and best practices that can be adopted for IoT development and deployment in the prototyping.
- Develop an MDD toolkit that allows inexperienced and experienced developers to create IoT prototypes rapidly through modeling language and code generation. The design of the IoTLink reflects the findings from the theoretical framework study.

- Develop a semantic device discovery component which goes beyond state-of-the-art by utilizing Lexical semantics to discover similar devices even when the Metadata is described with diverse terminologies.

1.4 Thesis Statement

This dissertation contends that a model-driven development supported by semantic device discovery is able to accelerate IoT software developments and allow inexperienced and experienced developers working together to develop IoT prototypes rapidly.

1.5 Assumption

This work assumes the future IoT ecosystems comprise an interaction between service providers and user applications as illustrated in Figure 2. The architecture describes several service providers run their businesses by providing innovative services enabled by IoT. The applicable owners do not need to operate their own IoT infrastructure that might cost quite significantly. Instead, they could lease IoT services that are operated by service providers. This would reduce the operational costs since the service providers may share their IoT infrastructures for several applications e.g., weather stations and satellite could be operated by a company while several websites subscribe to the service to retrieve the data. Moreover, IoT companies could provide, e.g., Real-time air quality data in the city, traffic reports, tracking goods or public transportation.

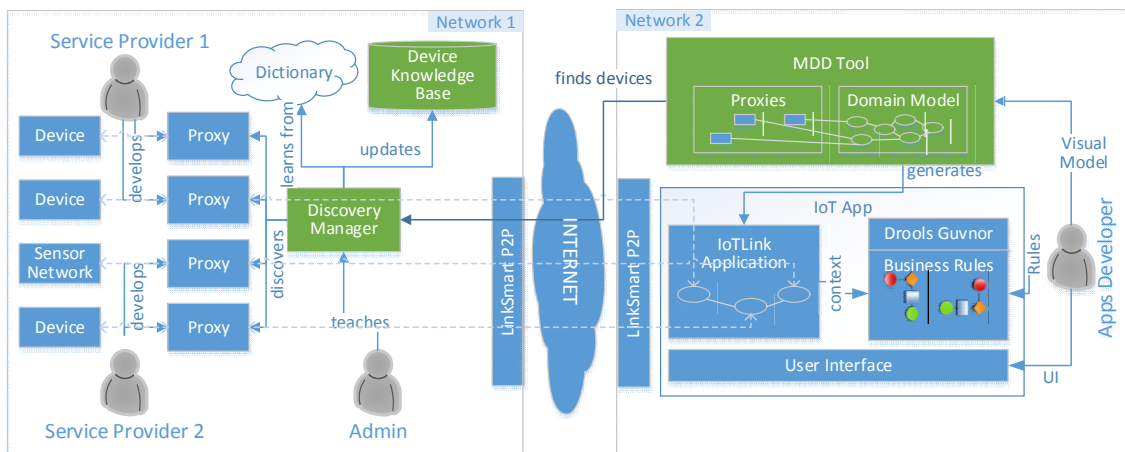


Figure 2. The envisioned prototyping toolkit for IoT

In this scenario, the application developers must be able to discover and subscribe to the appropriate services and use them to acquire contextual information for their systems. The service providers must provide uniform interfaces to access their IoT infrastructure which can be done through software proxies. The proxy should offer the intended services without requiring developers to have extensive knowledge about the device’s communication technology. To enable the discovery, the proxies must be annotated with metadata of the

devices such as the capabilities of the sensors and actuators and their quality parameters (e.g., Accuracy, availability, and costs). The information is used by the users to decide which services are appropriate for their needs.

A component called Discovery Manager was implemented, which provides a centralized entry for the users to find the appropriate IoT for their needs. The Discovery Manager acts as a discovery broker that can discover proxies in the local network using WS-Discovery² protocol as well as allow devices to register themselves. On the other hand, the application could look up the devices based on their semantic properties.

When local devices are discovered or when devices register themselves, the Discovery Manager extracts Meta information and store them in a central knowledge base. During application development, this knowledge is used by the application developers to find IoT devices based on their needs. The right side of Figure 2 depicts that the developers are able to query information about the devices that have been discovered and filter them based on certain requirements such as capabilities and functions. This work also assumes that diverse terms could be used by the service providers to describe their IoT infrastructure. Similarly, the application developers may also use diverse terms to find the appropriate IoT infrastructure. Therefore, the discovery broker must be able to overcome these challenges.

1.6 Overall Methodology



Figure 3. Research Methodology

As depicted in Figure 3, this work performed four major activities which were repeated in small iterations to reduce the uncertainty. In the first two phases, an exploratory research (Jaeger & Halliday, 1998) based on survey and literature study was applied to synthesize information on how the IoT applications are made. The obtained information builds a theoretical framework around the IoT architectures comprising the best practices and architectural patterns that can be used as a guide for building IoT prototypes. In the second phase, a confirmatory research was done for confirming the proposal of architectural patterns and best practices. They are evaluated with different stakeholders against the visionary scenarios as suggested by the “Software Architecture Analysis Method (SAAM)” (Kazman et al., 1994).

²<http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01> (Retrieved on March 5, 2014)

Based on the findings of the first two phases and the requirements of the developers, IoTLink was designed and implemented iteratively following the user-centered design approach (ISO, 2009). Moreover, its usability is evaluated qualitatively. Furthermore, the usability parameters were quantified by measuring the efficiency, effectiveness, and the satisfaction as suggested by ISO 9241-11 (ISO, 1998). To evaluate the applicability of MDD in IoT prototyping, IoTLink was evaluated through case studies in three European research projects with a different set of requirements and constraints.

1.7 Document Outline

The organization of the remainder of this document divides as follows:

- Chapter 2. The Notions of the Internet of Things, describes the various understandings of IoT including the available definitions and scope.
- Chapter 3. IoT Application Domains, describes relevant application domains that are often related to IoT technology.
- Chapter 4. State-of-the-art in IoT Architecture and Development Platforms, describes the latest advancements in providing development tools and platform for implementing IoT prototypes.
- Chapter 5. Design Concept and Technical Implementation of IoTLink, describes the implementation of IoTLink including the conceptualization of composing IoT components and its software architectures.
- Chapter 6. Sharing and Discovering IoT Semantically, describes the state-of-the-art of IoT discovery, and the IoTLink discovery component, which solves the terms diversity.
- Chapter 7. Case Studies, describes the application of IoTLink in three case studies which examine its applicability for IoT prototyping.
- Chapter 8. IoTLink Formal Evaluation, describes the controlled experiments to evaluate the usability aspects of IoTLink.
- Chapter 9. Conclusion and future work, conclude the research and present a possible future work and outlook for IoT prototyping.

Chapter 2.

The Notions of the Internet of Things

This chapter introduces the evolution of the IoT definitions from different perspectives. These definitions are used as the foundation of the research in this dissertation. Moreover, the scope of IoT as well as the overlapping fields within the computer science are discussed.

2.1 Definition

The idea of surrounding objects being interconnected and able to work together on the background supporting human activities was started by Mark Weiser, a chief scientist at Xerox PARC who has a vision for ubiquitous computing (Weiser, 1999). Ubiquitous computing envisions that the technology would be available everywhere, transparent to the users, and calm supporting users without imposing a significant mental load nor a steep learning curve. This vision requires an intelligent system that is able to sense users' contextual information and understand what their activities and intentions. This vision has driven continuous innovations behind the IoT such as connectivity, miniaturization, and intelligent information processing. As the enabler technology for ubiquitous computing, IoT should not only be concerned with connectivity between objects, but also autonomous interaction between them to serve the ultimate goal of supporting human activities.

The IoT can be viewed from different perspectives. Initially, the 'Internet of Things' was coined by Kevin Ashton. He has mentioned, "The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so" (Ashton, 2009). Since then IoT definitions have been understood from three perspectives (Atzori et al., 2010). The first perspective corresponds to the "Internet" part of IoT, which deals with various communication infrastructure between devices, systems, and the users. The second perspective corresponds to the "Things" part of the IoT, which focuses on enabling interaction between the "smart" physical objects as well as with the users. This area has produced an identification technology such as RFID for integrating physical objects into IoT. Third, the engineering aspects of distributed computing are concerned with the access to the physical things and devices, maintaining a network of things and retrieving useful information from massive and presumably inconsistent data generated by IoT. This field has produced middleware approaches, applying software oriented architecture for IoT, using semantics to discover devices, as well as mining information from IoT data.

The internet was born in the 1960's when Advanced Research Projects Agency Network (ARPANET) was introduced to enable communication between research laboratories funded by

the US Defense Advanced Research Projects Agency (DARPA). ARPANET became publicly available in the 80s and has laid the technology foundation such as TCP/IP which became the backbone of the internet as we know now. The IoT is an evolution of the current internet which is dominated by machine-human interactions such as browsing web, sending emails, and consuming media. The IoT enables embedded systems to retrieve and publish information from and into the internet automatically reducing the need of manual work that must be done by human, e.g., Organizing travel that involves different transportation and accommodations. Figure 4 shows the internet evolution from local network communication to a seamless connection between devices, according to (Perera et al., 2014).

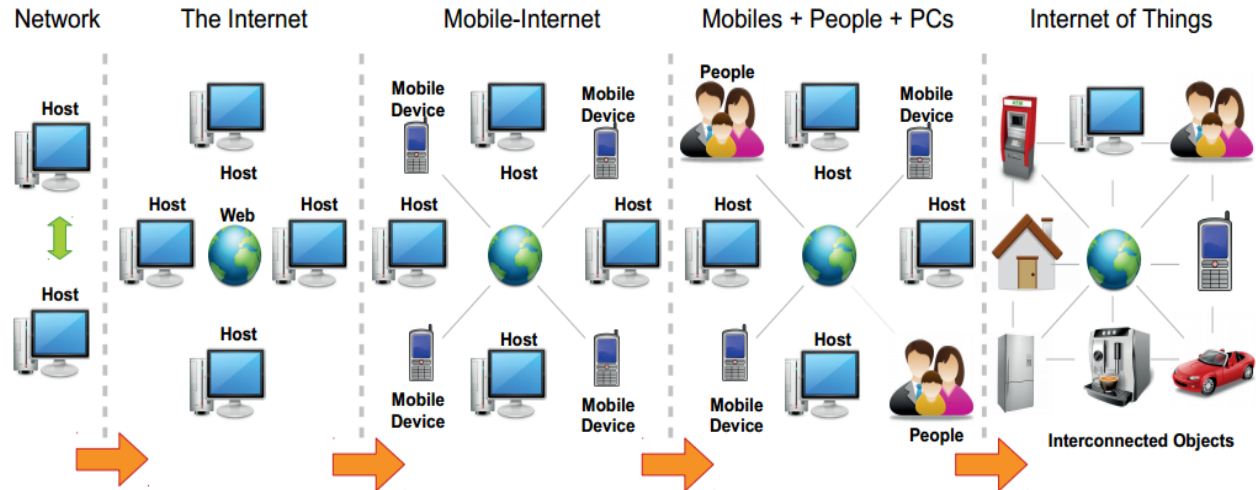


Figure 4. Distributed computing evolution to the IoT (Perera et al., 2014)

The “Things” vision initiated by the Auto-ID lab, where they investigated how information of physical objects can be retrieved. They propose to tag physical objects with RFID containing a universally unique ID. The information about the objects is stored in a centralized server which can be retrieved by passing the ID (Brock et al., 2001). However, the IoT research landscape has evolved beyond this initial definition. The IoT research community has gone even further by proposing approaches to bring physical objects, intelligent software services, and people together. Unfortunately, the lack of standardization and interoperability has made these into separate vertical “silos” or Intranets of Things (Zorzi et al., 2010). For instance, in the industrial automation, the devices on the shop floor are connected to Fieldbus networks, which must be bridged to the Ethernet networks to communicate with PCs.

Technology enablers such as embedded micro controllers, wireless technology, and semantic web are being investigated in the IoT projects like BUTLER³, ebbits⁴ and BEMOCOFRA⁵ to bring intelligent services that are not only able to present plain data to the users but also contextualized information to support human making more accurate decisions. IoT-A⁶ and IoT-I⁷ try building a unified IoT community and IoT vision in Europe.

³ <http://www.iot-butler.eu/> (Retrieved on March 5, 2014)

⁴ <http://www.ebbits-project.eu/news.php> (Retrieved on March 5, 2014)

⁵ <http://www.bemo-cofra.eu/news.php> (Retrieved on March 5, 2014)

⁶ <http://www.iot-a.eu/> (Retrieved on March 5, 2014)

⁷ <http://www.iot-i.eu/> (Retrieved on March 5, 2014)

Moreover, availability of affordable embedded micro controllers and electronic prototyping platforms such as Arduino⁸ and Raspberry Pi⁹ has furthered the growth of the IoT community. These phenomena blur the original definition and scope of IoT.

A few works that are more recent try to recapture a more up to date definition of IoT. For instance, in 2005, the international telecommunication union (ITU) released their report about the IoT. They identified four technology enablers in IoT including the RFID, sensor technologies, smart object technologies, and nanotechnology. According to ITU “the IoT will entail the connection of everyday objects and devices to all kinds of networks, e.g., Company intranets, peer-to-peer networks and even the global internet” (Peña-López, 2005).

McKinsey describes the IoT as “sensors and actuators embedded in the physical objects that are linked through wired and wireless networks, often using the same IP that connects the Internet”(Y.-K. Chen, 2012). According to the Cisco Internet Business Solutions Group (IBSG), “IoT is simply the point in time when more “things or objects” were connected to the Internet than people” (Evans, 2011a). Another paper argues that the IoT must involve intelligent, embedded in the physical objects instead of only communication capability (Y.-K. Chen, 2012)

The Internet of Things European Research Cluster (IERC) has come up with a definition to guide the European projects related to IoT research. IERC defines IoT as "A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual “things” have identities, physical attributes, and virtual personalities. They use intelligent interfaces, and are seamlessly integrated into the information network" (Vermesan et al., 2010). This definition strains not only the communication between physical and virtual world, but also demands intelligent aspects towards autonomous systems that require very little to zero maintenance.

Gubbi et al. coined a more generic definition for the Internet of Things as follows “IoT for smart environments is an interconnection of sensing and actuating devices, providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless large-scale sensing, data analytics and information representation using cutting edge ubiquitous sensing and cloud computing.” (Gubbi et al., 2013)

Because of this diversity, there exist some concerns that IoT is only a “marketing umbrella” for other research fields such as distributed computing and ubiquitous computing. Contending this belief, Uckelmann et al. claim that IoT is not synonymous to these fields, it rather relies on these approaches to achieve synergies between things on the internet scale (Uckelmann et al., 2011).

2.2 The Notion of “Things” within the IoT

“Things” in context of IoT is usually understood as physical objects that interact through a communication medium. The physical objects in this context could be electronic devices with network capabilities or any physical objects (electronic and non-electronic) without any communication ability. The first category is able to communicate with other devices with similar

⁸ <http://arduino.cc/en/Guide/Introduction> (Retrieved on March 5, 2014)

⁹ <http://www.raspberrypi.org/> (Retrieved on March 5, 2014)

communication capabilities. E.g., Smartphones, Tablets, and PCs are able to communicate through Wi-Fi. In recent years, this type of devices is increasingly immersed to various kinds of consumer electronics devices and home appliances. Manufacturers have been equipping TVs, refrigerators, and ovens with embedded computers, touch screen displays and Wi-Fi connections. General Electric has announced GE Brilliance™, ovens that can be controlled through smartphones from the internet¹⁰. LG has produced their smart appliances including washing machines, refrigerators, and ovens that can be controlled through smartphones¹¹. Moreover, wearable devices with network capabilities such as Android Wear¹² smart watches and Google Glass¹³ become more popular and affordable.

Physical objects without any communication capabilities such as products in the supermarket, livestock, building structures, humans, or legacy devices without network interface can be represented by proxies which are able to deliver information about the products as well as influence their states. The proxies may consist of devices or powerful servers with communication capabilities that are able to provide information about the objects as well as influence their states. Such devices are also called “Enabler Devices” since they enable physical objects to take part in the IoT.

The ITU has defined several device categories that can be used to classify devices used in the IoT, which is depicted in Figure 5. They classified the relation between devices and physical things according their relations to data which is summarized as follows (ITU-T, 2012):

- Data-carrying device: A data-carrying device is attached to a physical thing to connect the physical thing with the communication networks indirectly.
- Data-capturing device: A data-capturing device refers to a reader/writer device with the capability to interact with physical things. The interaction can happen indirectly via data-carrying devices, or directly via data carriers attached to the physical things. In the first case, the data-capturing device reads the information on a data-carrying device and can optionally also write information given by the communication networks on the data-carrying device.
- Sensing and actuating device: A sensing and actuating device may detect or measure information related to the surrounding environment and convert it into digital electronic signals. It may also convert digital electronic signals from the information networks into operations. Generally, sensing and actuating devices form local networks communicate with each other using wired or wireless communication technologies and use gateways to connect to the communication networks.
- General device: A general device has embedded processing and communication capabilities and may communicate with the communication networks via wired or wireless technologies. General devices include equipment and appliances for different IoT application domains, such as PCs, industrial machines, smart phones, and home electrical appliances.

The first and second categories are relevant for attaching and retrieving information to and from physical objects. RFID and barcodes have been used to attach information to products and goods. The data that can be encoded is limited and can be as simple as identification number or a web address. This information can be used to look up further information about the object from a

¹⁰ <http://www.geappliances.com/connected-home-smart-appliances/> (Retrieved on July 28, 2014)

¹¹ <http://www.lg.com/us/discover/smarthing/thinq> (Retrieved on July 28, 2014)

¹² <http://www.android.com/wear/>

¹³ <http://www.google.com/glass>

centralized information system. The data-capturing device could be based on the radio technology to read RFID tags, or optical sensors which could read barcodes.

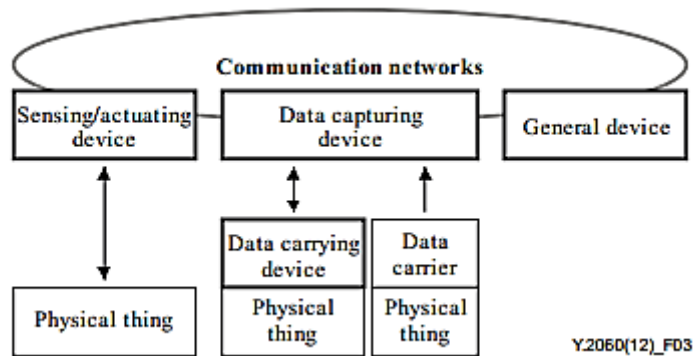


Figure 5. Types of devices and their relationship with physical things (ITU-T, 2012).

Sensing devices and sensor technology in general is an essential IoT enabler that is used for acquiring contextual information about physical objects. Sensing technology has gone through a rapid development particularly since the micro-electro-mechanical system (MEMS) was invented. It enables the development of intelligent and compact sensors from one millimeter up to 20 micrometer in size, including a microprocessor (Gardner et al., 2001). This level of miniaturization has allowed the development of compact wireless sensor nodes that can be installed in places that are hard to be reached by conventional wired sensors. The advancement in wireless sensor and actuator network (WSAN) has influenced IoT development significantly. Not only gives a birth to the IoT term when the RFID was being developed, but it has also revolutionized how contextual information about physical objects are acquired. For instance, in logistic domain, WSAN enables tracking of goods' location (J. Kim et al., 2008), which would not have been possible to be done with wired technology.

2.3 Context and Scope of IoT

Identifying the context and scope of IoT, the development of similar research fields must be reviewed. However, it is practically not possible to draw clear separations between these topics since less mature fields tend to change dynamically. Several terms have been used to describe research fields that overlap with IoT such as, machine-to-machine (M2M), cyber-physical system (CPS), and wireless sensor network (WSN). Chen et al. tried to describe how these fields are related, but not identical. As depicted in Figure 6, they summarize the semantic of IoT, CPS, M2M and WSN as the following. WSN, M2M, and CPS are part of IoT in which WSN provides a basic scenario of IoT that relies on wireless sensing and actuation. WSN supplements the M2M goal since M2M covers a more generic communication between machines that can be done through wired or wireless medium. M2M currently provides the main pattern of IoT describing a world where machines could communicate with each other seamlessly. However, CPS provides an evolution to the M2M vision in which intelligent information processing is an essential part to achieve autonomous subsystems that is able to communicate with other intelligent subsystems (M. Chen et al., 2012). However, their claim was not backed up by any strong reasoning and extensive literature studies. In practice many CPS literatures focus on more localized

communications between real-time embedded systems with high degrees of automation and control loops (Jianhua et al., 2011). Currently, the integration of automated real-time devices and internet has raised a controversy particularly on keeping the balance between the degrees of uncertainty when communicating through the internet and the real-time requirement of safety-critical systems (Koubâa & Andersson, 2009). The author believes that some degree of separation between deterministic real-time system and the communication through the internet will still exist. However, through a careful design some degree of communication to the internet could still be established without interfering the real-time requirements of the CPS systems. Many electric cars have allowed their owners to monitor the charging process, control the climate, and start the engine from a mobile app^{14,15}. This interaction can be done when the system designers carefully isolate the system modules to work independently regardless the delay caused by other modules as suggested by approaches for mixed critical systems (Kumar N.G. et al., 2013).

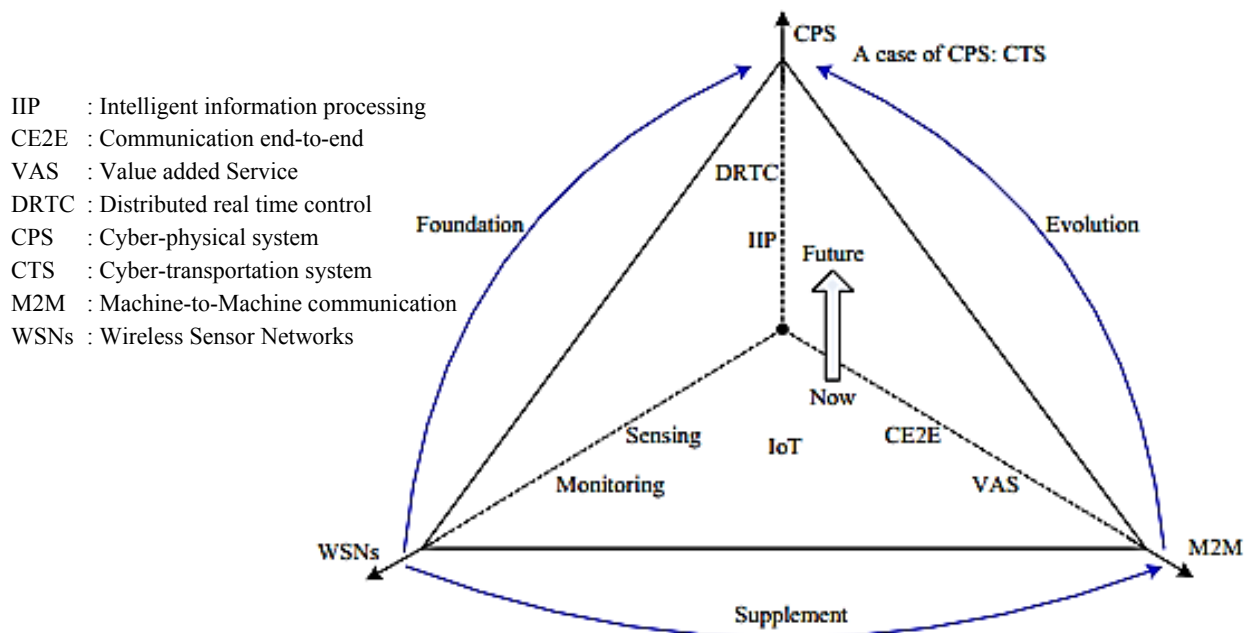


Figure 6. The relation between IoT and related fields such as M2M, CPS, and WSN (M. Chen et al., 2012)

While there has not been yet an agreement on an IoT definition, many works describe overlapping scenarios and requirements for the internet of things. Analyzing the requirements and scenario will provide us with a better overview of the scope and context of IoT. The aforementioned definitions try to give a common understanding of the main components that forms the IoT. In practice, IoT applications demand a broader set of functional and non-functional requirements. For instance, security, trust, and privacy raise a major concern when connecting millions of objects into the same network e.g., whether or not we are fully in control of our personal information, whether data and information that we obtained is trustworthy.

¹⁴ <http://www.leaflinkapp.com/> (September on August 14, 2014)

¹⁵ <http://9to5mac.com/2014/08/20/tesla-to-allow-iphone-to-start-model-s-without-keyfobs-with-v6-0-update/> (Retrieved on September 14, 2014)

Based on the results of the IoT projects and several expert workshops, the European commission has identified several enabling technologies. These technologies are categorized into (Vermesan et al., 2013):

- Technologies that enable “things” to acquire contextual information,
- Technologies that enable “things” to process contextual information, and
- Technologies to improve security and privacy.

These three challenges are the main research directions, which have been and will be funded by the European commission as one of the leading force that determine research directions in Europe. The European commission has set a target to invest 51M € within 6 years for the IoT research and development. Furthermore, they aim to increase the number of connected objects in Europe to 25 billion by 2020 (E.1, 2014).

Context acquisition has been investigated quite extensively within the ubiquitous computing and context awareness field. Zimmermann, in his dissertation, presented a survey of approaches for acquiring contextual information through sensors (A. Zimmermann, 2007). Sensors can be seen in a broader meaning, which include not only sensor devices, but also input from the users. The complexity of context acquisition requires developers to deal with several problems such as building a communication with heterogeneous sensor technology and interpreting the sensor readings into a meaningful sense. These difficulties have motivated the development of middleware (Choi et al., 2005; Gu et al., 2005; Henricksen et al., 2005) which aims at supporting developers for integrating sensors, communicating with them and interpret the values. Several works use ontology to model context aware applications (Gu et al., 2004; Qin et al., 2007). They argue that an ontology provides a broad expressiveness to express relations between entities.

2.3.1 IoT related and enabling technologies

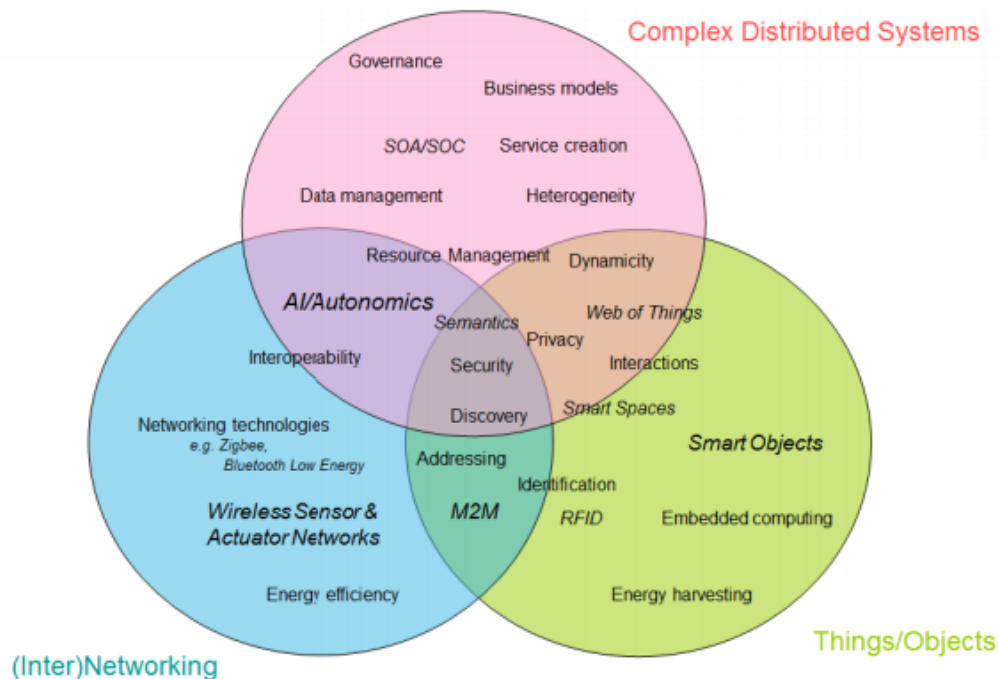


Figure 7. Technologies related to IoT research and development (Lee et al., 2013).

Atzori et al. performed a survey of technologies that are often related to the IoT (Atzori et al., 2010). This survey was extended by Lee et al. as illustrated in Figure 7 (Lee et al., 2013). They categorized the relevant technologies, according to three overlapping perspectives that are understood as the IoT core technologies, including “Thing”, “Network”, and “Semantic”. Furthermore, Lee et al. extended the semantic vision to a broader view, which involves technologies for maintaining complex distributed data and services such as Service Oriented Architecture, SOA Governance, and Semantic that focuses on extracting useful information from data generated by the IoT.

Investigating the technology that often linked with IoT, the author found that several technologies have gained more adoption than others. This can be caused by many factors such as the simplicity of the technology, the support from the major players, the availability and maturity of the reusable components and tools. The available publications and reusable components indicate the following technologies are closely related to the IoT developments:

- Web of things, which is popularized by Guinard et al. (Duquennoy et al., 2009; Guinard et al., 2010a; Ning et al., 2013; Taylor et al., 2013). Some of the Web of things approaches also include end-user development platform through Mashup developments (Guinard et al., 2010b).
- M2M is concerned with communication between machines. Currently there exist two specifications including ETSI M2M that is proposed by European Telecommunications Standards Institute (ETSI, 2011; Guang Lu et al., 2012) and oneM2M (Emmerson, 2010; Swetina et al., 2014). Moreover Eclipse foundation, one of the biggest open source contributors, is working on providing an implementation of MQ Telemetry Transport (MQTT) that is a lightweight publish-subscribe messaging protocol, CoAP that is an IETF standard protocol targeting resource-constrained devices, and OMA LWM2M (Lightweight M2M) that is a standard device management protocol from the Open Mobile Alliance, and ETSI M2M.
- Low powered wireless sensor network is concerned with developing wireless sensor and actuator based on IEEE 802.15.4 (Jose A Gutierrez et al., 2001; Jose A Gutierrez et al., 2004). e.g., ETSI has proposed 6LoWPAN (Mulligan, 2007; Hui & Thubert, 2010) which uses IPv6 for sensor addressing. ZigBee and WirelessHART have been developed based on IEEE 802.15.4 for enabling wireless sensor in building automation and industrial automation.
- IoT Middleware, which focuses on abstracting different communication technology, providing reusable components for managing IoT and their services e.g., (Jammes et al., 2005; Hadim & Mohamed, 2006; Jahn et al., 2010; S. Bandyopadhyay et al., 2011b; M Eisenhauer et al., 2011; Acquaviva, Blaso, Dalmaso, Del Giudice, et al., 2012; Pramudianto, Khaleel, et al., 2013)
- Semantic interoperability investigates semantic representations such as ontologies to describe IoT systems, which enable applications making sense sensor data. (Qin et al., 2007; Katasonov et al., 2008; Hachem et al., 2011)

2.4 Conclusion

After investigating diverse definitions that exist for IoT, the author concludes that IoT definition must explain the “Thing” and “Internet” aspects of IoT. A clear separation between what would and would not be considered as a “Thing” in the IoT context is needed. The author believes that to answer this question, the application domain and the specific implementation play a very important role. Since in reality, developers and system architects must have strong reasons to

include physical objects as part of the IoT such as business cases and government regulations. These factors decide the granularity of physical objects that must be represented in the virtual world. For instance, although it is possible from the technological point of view to represent each banana as a virtual object, from the business perspective it might be more beneficial to represent a batch of banana as a physical object.

Secondly, the “Internet” aspect of IoT should clarify how these “Things” communicate at a very basic definition. The definition could be advanced by explaining how “Things” cooperate autonomously to reach a particular goal. The IoT definition should consider the fundamental view how physical objects are represented in the virtual world in order to provide added values for our life.

Taking these aspects into consideration, the author’s view on IoT is rather technical. It can be described as follows:

IoT is a “vision” of the world in which, physical objects could seamlessly communicate with other physical or virtual objects through electronic media. The objects that do not have communication capabilities or have incompatible communication capabilities could be represented by virtual objects that act as their proxies. The proxies reflect their actual states, contain their information, and able to interact with other virtual entities on the behalf of the physical objects that they represent. The proxies are responsible for ensuring interoperability between things and therefore must provide translation services for the incompatible technologies. These proxies are created with certain goals. Therefore, they may not expose all possible information about the physical objects nor have the ability to bridge all possible services that the physical objects may have. Virtual objects may be created to expose only relevant information and able to represent a set of functions to fulfill certain goals. In the opposite, a proxy may represent a composition of physical objects as a virtual object, which sometimes required to encapsulate the complexity of several physical objects.

Moreover, it is quite difficult to summarize a very broad technology domain only by examining the definitions. Therefore, to understand the context of IoT, this work investigates the technologies and implementations within specific domains that are often related to IoT such as RFID, Web of Things, IoT Middleware, wireless sensor and actuator network. In addition, this work discusses the implementation of IoT within various application domains as well as its integrations with legacy systems.

Chapter 3.

IoT Application Domains and Related Technology

This chapter describes four application domains in which IoT is seen as a significant technology that could improve the flow of information across organizational structure. Additionally, these application domains present diverse challenges which should be considered in IoT system design. For instance, industrial automation involves complex and distributed legacy devices, building and home automation require some degree of human interaction, and smart grid requires a large scale network deployment. This chapter also views the development platform used for these application domains as well as the accepted communication standards.

3.1 Industrial Automation

Industrial automation enables goods being mass produced with lower costs and consistent quality. Industrial automation systems consist of devices and subsystems that are designed to work in deterministic environments. This means that, when designing the system, variations are minimized and must be predictable. A general architecture for industrial automation is illustrated in Figure 8. It shows a classical distributed control system (DCS) that relies on a hierarchical network of distributed programmable logic controllers (PLC). PLCs, sensors and actuators are wired together and communicate through industrial Fieldbus networks. Fieldbus networks dominate the communication in the industrial automation domain since they are designed to support real-time communication and work deterministically.

In the early days of industrial automation, various proprietary technologies were used by vendors supplying components for manufacturing machines. Integrating these heterogeneous components required massive efforts and costs (Thomesse, 2005). Translating between two different protocols required additional gateways which must be installed and configured by experts. Since the 80s, many efforts to standardize, communication networks on the shop floor were initiated. Today, many operational factories have adopted the Fieldbus standard such as Profibus (Tovar & Vasques, 1999), Modbus (Modbus, 2004b) or an industrial Ethernet standard such as Profinet which has gained popularity to support better interoperability with computer networks (Jasperneite & Feld, 2005).

3.1.1 Interoperability

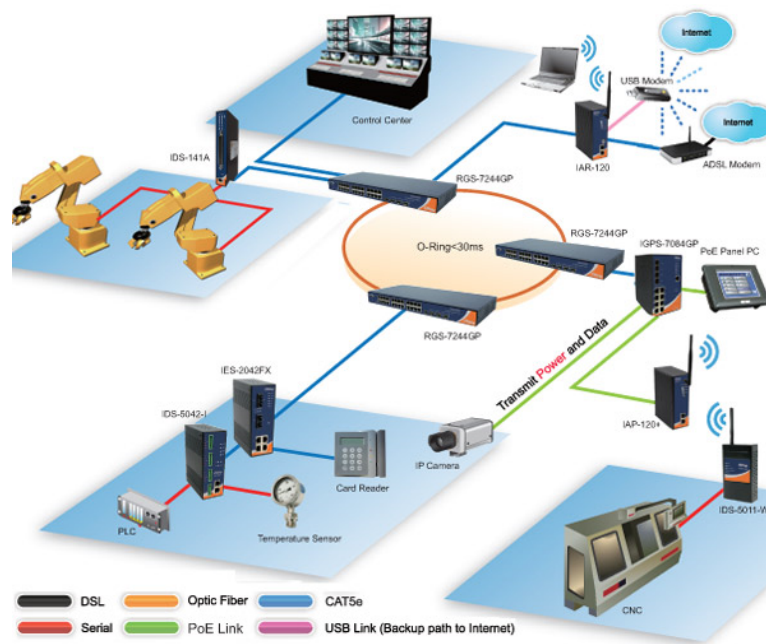


Figure 8. Distributed control architecture in factory automation¹⁶.

The competition in the manufacturing business has forced decision makers to be promptly responsive to the market conditions. This means that, they must be able to analyze the necessary data and information in a very timely manner and take decisions upon them. Making data instantly available requires a seamless vertical and horizontal interoperability from the shop floor into the enterprise system. In the last decade, much research has been done towards interoperability between industrial automation and the ICT ecosystem upon which the enterprise applications are built. As depicted in Figure 8, diverse devices and systems used in factory automation poses a real challenge to achieve interoperability between them.

Addressing the horizontal interoperability in manufacturing, Object Linking and Embedding (OLE) for Process Control (OPC) was created by the OPC foundation that is jointly funded by industrial automation vendors (OPCFoundation, 2014). Since then OPC has become a de facto standard in industrial automation (Zheng & Nakagawa, 2002). OPC provides technical specifications of the software interface that can be implemented by software vendors. The initial specifications consist of the OPC Data Access Specification (OPC-DA) that elaborates how real-time data produced by heterogeneous devices could be read by applications while OPC Historical Data Access (OPC HDA) specifies how to retrieve archived historical data from devices or applications.

¹⁶<http://www.anewtech.net/solutions-industrial-automation-industrial-managed-unmanaged-ethernet-switch-device-server-industrial-media-converter.php> (Retrieved on March 30, 2014)

As illustrated in Figure 9, OPC follows a client-server architecture in which the server establishes communication with devices using the native device drivers and encapsulate these various drivers with a uniform application-programming interface (API) for the OPC clients.

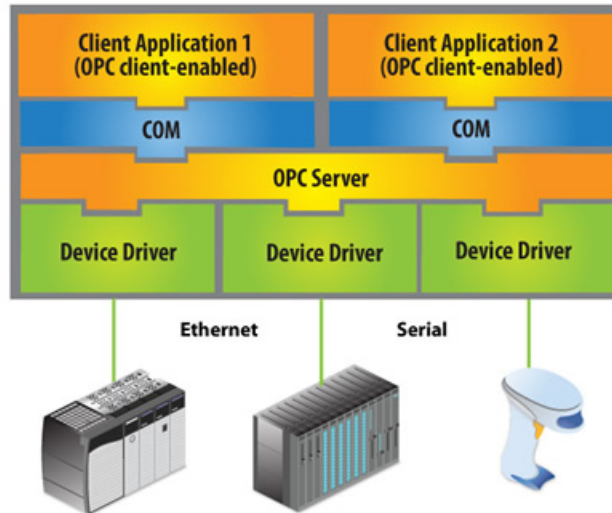


Figure 9. OPC Architecture¹⁷

The latest version of the specification, the OPC Unified Architecture (OPC-UA), unifies previously separate modules such as Data Access, Alarm & Events, and Historical Data Access. It decouples OPC from Microsoft DCOM/COM and based on a service oriented architecture that allows vertical interoperability on a different level of the factory through Web Services (Schleipen, 2008). OPC-UA provides an address space that represents objects in a standard manner.

Achieving vertical interoperability requires business and manufacturing processes to exchange interoperable information, ensuring products are built at the right time when demands from the market exist. However, achieving vertical interoperability is more difficult than horizontal interoperability since it involves multiple systems supplied by various vendors such as manufacturing execution systems (MES), product lifecycle management (PLM), enterprise resource planning systems (ERP), and supply chain management systems (SCM) (Brandl, 2002). ISA 95 / IEC 62264 standard tries to close this gap by providing a standard integration model for allowing interoperability between various systems from the shop floor to the enterprise systems exchanging the required information (Sauter, 2007). As depicted in Figure 10, the specification manages different integration roles from the shop floor into business systems which consists of five parts including:

- Part 1: Models and Terminology, describes a set of terminology, concepts and models for integration of control systems and enterprise systems based on Purdue Reference model for computer-integrated manufacturing (CIM) defined in ISO 15704.
- Part 2: Object model attributes, describes the exchanged information in sufficient detail to allow compliant interfaces to be defined.

¹⁷ http://www.kepware.com/Products/Typical_Applications/UCON/UCON_Typical_Applications.asp (Retrieved on March 30, 2014)

- Part 3: Activity models of manufacturing operation management, describes the activities associated with manufacturing operations which usually managed by an MES.
- Part 4: Object models and attributes of manufacturing operations management, describe the information to be exchanged for different MES activities.
- Part 5: Business to Manufacturing transactions, describes the possible transactions between business and manufacturing systems as well as the information to be exchanged. The models covered in this part including the personnel, equipment, maintenance, material, process segment, production capability, product definition, production schedule, and performance models.

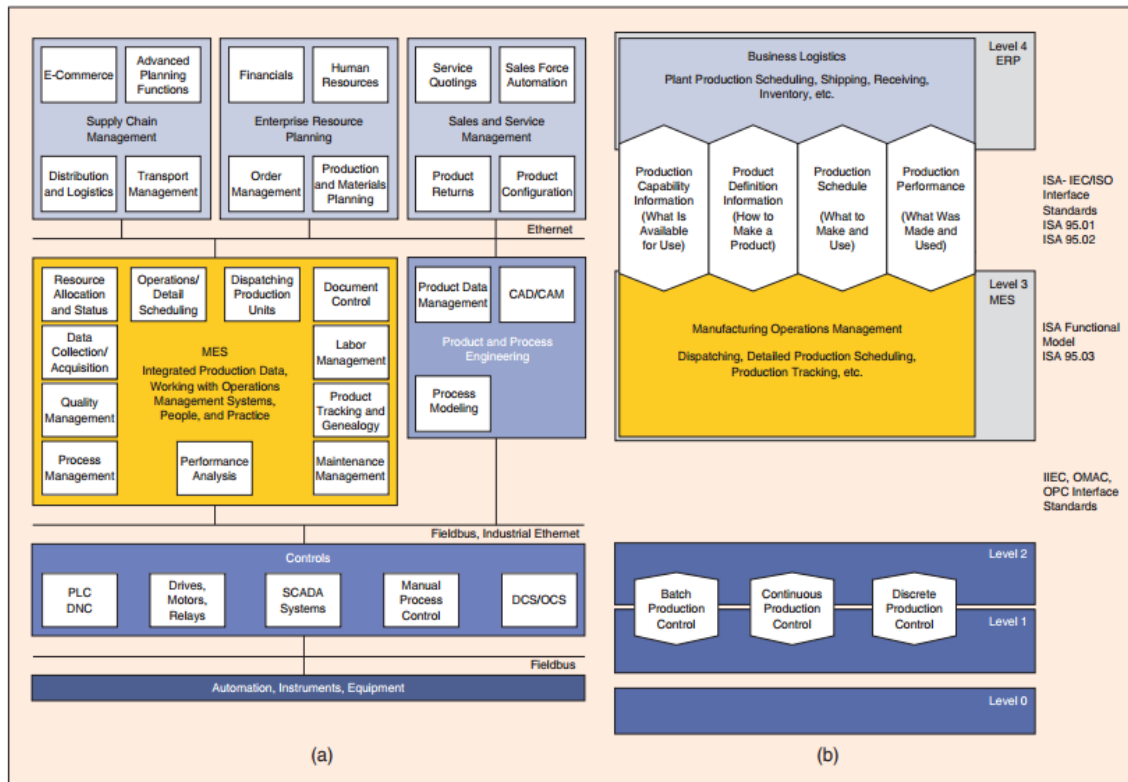


Figure 10. (a) Plant information model according to MESA (International, 1997) and (b) hierarchical enterprise model according to ISA-95 and the scope of the standards (Sauter, 2007)

ISA-95 provides a high-level guideline, which allows a high degree of freedom for the implementation. This may lead to incompatible implementations. The World Batch Forum (WBF) provides an XML schema that follows ISA-95 specification which is named Business to Machine Mark-up Language (B2MML) (He et al., 2012). The XML schema provides a serialization of objects and their attributes as specified in ISA95 standard. B2MML specifies data format that can be transferred through Web Services in a service oriented architecture. The existing ERP and MES on the market use B2MML to transfer the production schedule from ERP to MES and the manufacturing performance information from MES to ERP which enables a faster transfer of production order and the manufacturing key KPIs (Emerson et al., 2007).

As the emerging IIOT technology, such as 6LoWPAN based sensors, becomes commonly available, several works have proposed to introduce them into the existing industrial automation systems (Pramudianto, Khaleel, et al., 2013). In this case, OPC and ISA-95 could play a significant role to allow communication between the legacy industrial automation systems which still rely on various Fieldbus networks and emerging IIOT technologies.

3.1.2 Industrial WSAI

Wireless sensors are very attractive, particularly for the current industries. They enable sensor deployment in places that are hard to be reached by wired sensors or where cables could easily be damaged because of vibrations and mechanical movements. For industrial use, WSAI must be designed to operate reliably in harsh environments, which may have unpredictable electromagnetic interferences caused by the reflection of wireless signals by any moving mechanical parts. Moreover, many industrial applications have a stricter time constraint such that the wireless solution must be able to guarantee the packet delivery in a specific time frame.

Many industries have shown a significant interest to the IEEE 802.15.4 (Jose A Gutierrez et al., 2001) standard that was introduced to enable low-rate wireless personal area networks. Since then the IEEE 802.15.4 standard has been developed into diverse full wireless stacks and has started to be adopted on the industrial shop floor to monitor physical parameters (e.g., vibration, temperature, pressure, power quality) which are critical to preserve the equipment's time to failure and ensure the quality of the manufacturing processes.

Several wireless standards based on IEEE 802.15.4 are proposed by different industrial organizations such as ISA100.11a (Committee, 2009), WirelessHART (Song et al., 2008), WIA (N. Wang et al., 2006), and ZigBee (Z. Alliance, 2009). Most of these standards feature self-organization and self-healing which minimize the required effort for maintaining the network. Some of them, such as ZigBee, support mesh and multi-hop networks that is useful to extend the range of the network as well as reliability by utilizing transmission on multiple links.

WirelessHART has gained popularity since it is designed to work in harsh industrial environments. WirelessHART is an extension of the traditional HART (Highway Addressable Remote Transducer) protocol that was an early implementation of Fieldbus. WirelessHART devices are categorized into field devices, adaptors, handhelds, gateways, and network managers. The gateway is in charge of bringing the communication between the wireless nodes and the process plant. The field devices are connected to the process plant while handheld devices are used by the workers to supervise and perform control to the process plant. A network manager is installed on a gateway to perform the network management functions such as configuring the network, schedule and maintain communications between devices.

WirelessHART maintains two types of routing (Jianping et al., 2008). First, graph routing whose paths are created explicitly by the network manager and downloaded to each node. Sending a packet must be identified with a destination ID. Since all nodes know how the topology looks like, they are able to forward the package to their neighbor until reaching the destination. Second, the source routing that is used to perform network diagnostic functions.

The route of the packet must be defined by the sender by including an ordered list of devices along the path in the header of the packet.

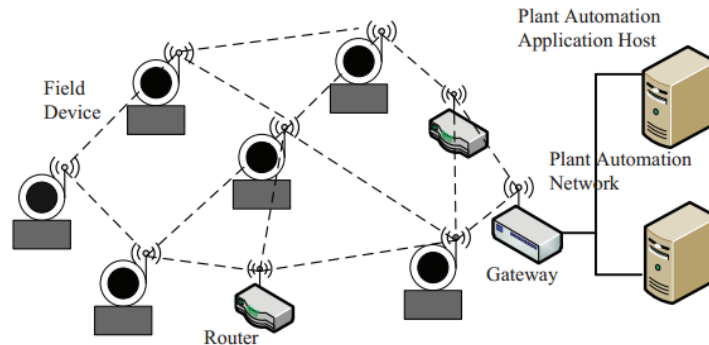


Figure 11. Typical WirelessHART deployment architecture (Gao et al., 2013).

Another approach to establishing network connections to low-powered sensor nodes is by taking advantage of the network stack used as the foundation of the internet which is TCP/IP. The IETF has proposed to use IPv6 and UDP for enabling communication with WSN. They proposed a lightweight version of IPv6 packet, called 6LoWPAN (Hui & Thubert, 2010). Mulligan claims that using IP down to the WSN flattens the naming and addressing hierarchy and therefore simplifies the communication model (Mulligan, 2007). This also reduces the need of protocol translations that must be done in gateways. In addition, the developers that are already familiar with the IP communication model are able to develop the applications without having to invest much time for learning how it works. This also opens the possibility to modify and adopt the existing communication protocols such as HTTP, UDP, Representational state transfer (REST), as well as routing approaches such as AODV (Perkins et al., 2003) for WSN. An example of this approach includes CoAP (Z Shelby et al., 2012) which is a lightweight version of REST. CoAP allows interacting with the sensor nodes through HTTP GET and PUT messages that are piggybacked on a UDP protocol.

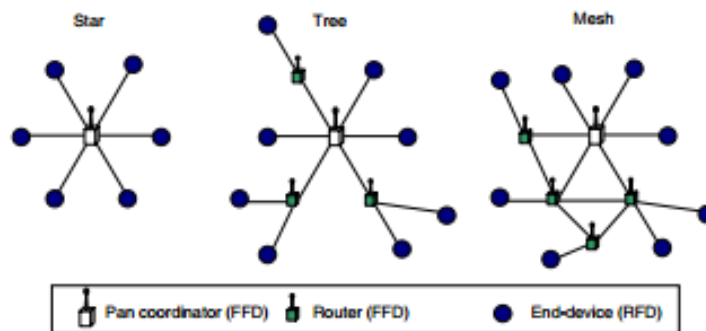


Figure 12. Overview of ZigBee network(Baronti et al., 2007)

Furthermore, ZigBee (Z. Alliance, 2009) is one of the widely used network protocol based on IEEE 802.15.4. ZigBee. It categorizes devices into full function device (FDD), and reduced function device (RFD). FDD is able to act as a network coordinator and talk to other devices. On the other hand, RFD is limited to a star topology since it cannot become a coordinator and

only able to talk to the coordinator. The possible network topology is illustrated in Figure 12 which shows that to build a star-mesh network several FFDs are required and having only one FFD only allows a star network to be built.

3.2 Product Traceability

Besides horizontal and vertical interoperability, traceability of products along the production chain characterizes the third type of interoperability. It is also known as temporal or longitudinal interoperability (Sauter, 2005). In this context, IoT can be used to collect, aggregate, and present information about the products throughout their lifecycle to the interested parties. Product traceability is quite crucial for localizing production errors and recall the affected products. For instance, car and aircraft parts are tagged with barcodes which can be traced back to the suppliers. In the case of faulty production, the car and aircraft manufacturers are required to trace the faulty components very quickly and recall the affected vehicles to replace the faulty parts before they endanger human life. Similarly, in the food production domain, being able to trace the defective products and accelerate the recall process could help save human life. In Europe, the traceability of live stocks is mandated by the European commission Council Directive 92/102/EEC (Comission, 2003). In 2004 the EC Regulation 21/2004 (Comission, 2003) requires that goats and sheep can be identified electronically using low frequency RFID that has been standardized in ISO 11784 (Code structure) and ISO 11785 (transponder activation & data transfer) (Kampers et al., 1999).

Technologies such as RFID and Barcode are already widely used for identification and tracking of assets throughout the supply chain network. Each product or a batch of products requires a universally unique ID of the product, which can be used by different actors in the supply chain to retrieve the necessary information from the other actors in the chain. Several standards for goods identification exist. For instance, EPCglobal regulates a standard architecture, interface, and identification scheme for RFID. Alternatively, barcodes are used ubiquitously to identify trade items, particularly by retailers (Becker, 2012). Most Barcode IDs in North America, UK, Australia, and Europe follow the Universal Product Code (UPC) scheme which is standardized by GS1¹⁸.

Healthcare and pharmaceutical industries have begun tracking medicines from the warehouse to the patients (Barchetti et al., 2010; Hay, 2014). Several hospitals have fully automated the process of picking the medicine from the storage and delivering them to the patients to prevent loss and human error that could endanger their patients' life (Chapuis et al., 2010).

3.2.1 RFID for Product Tagging

RFID tags can be distinguished based on the identification format, power source, and the frequency where it operates. Based on the Id format, different tags exist that are able to contain either 64, 96, or 128 bits of data. Based on the power source, there exist passive tags that must be powered by the reader to transmit any data and active tags that are powered by batteries and able to send signals independently from the readers. Based on the operating frequency, the tags vary from low frequency that operates in a range of 120–150 kHz, High

¹⁸ <http://www.gs1.org/> (Retrieved on Oct 5, 2014)

frequency, which operates in a range of 3 to 30 MHz with a range between 10cm-1m, to ultra-high frequency (UHF) tags that operate in the frequency band of 300MHz - 3GHz. Although the frequency used differs in different countries, the most common frequency band used is at 13.56 MHz for the HF tags, and 900-915 MHz for the UHF tags which follow the UHF Gen2 standard (Roberti, 2004).

The frequency and power source of the tags determine the range of the transmission and the resilience of the signals within the operating environment. Choosing tags to use depends on the application requirements. For instance, the low-frequency (LF) tags only support single read and up to 8 cm range that provides a suitable solution for scenarios such as wireless keys and electronic payment. The UHF tags support multiple reads and up to two meter range, which is good for identifying batches of products passing through a gate, for this purpose exist RFID gate readers. UHF tags cost very low due to simpler manufacturing cost compared to the LF tags. However, the cost of the reader might be higher due to higher complexity of the technology (Solution, 2011). Several UHF can be read up to 12m distance with a faster data rate, making it suitable for applications such as automatic toll collection system or asset management for heavy products.

In livestock farms, not all RFID tags could be used because livestock are able to attenuate radio signals significantly. The RFID tags for livestock initially used LF tags. However, UHF tags are being explored for tagging livestock recently since the range it offers could be used for localizing the livestock within the farm. This opens the possibility to study the movement pattern of the animals to identify stress and illness of the animals. RFID Tags can be attached to the ear (Figure 13), neck or brisket area.



Figure 13. RFID tag attached to the ear of a pig (Wasserman, 2009)

3.3 Building and Home Automation

Building and home automation are two areas where IoT could improve the interoperability between devices. Many technologies used for building automation are similar to the industrial automation domain. Commercial Building Management systems (BMS) such as Siemens Desigo¹⁹ rely on a network of industrial controllers that are connected to sensors and actuators for controlling lighting, HVAC (heating, ventilation, and air conditioning), and safety systems such as fire detection systems or security systems. Many of the existing BMS still rely on competing bus standards such as BACnet (Haakenstad, 1999), KNX (Association, 2004), LONWorks (Hur et al., 2006). These networks connect sensors, actuators, and embedded controllers that are programmed to automate the building management strategy defined by the utility manager through a management application. These networks are connected to a gateway that allows a communication to the applications that run on Ethernet networks, e.g., A human interface device. The applications that run on human interface devices allow the facility manager to define schedules, perform monitoring, and to control the devices in the building. Newer BMSs even have a web-based or mobile applications that could access the connected sensors and actuators from the internet as depicted in Figure 14.

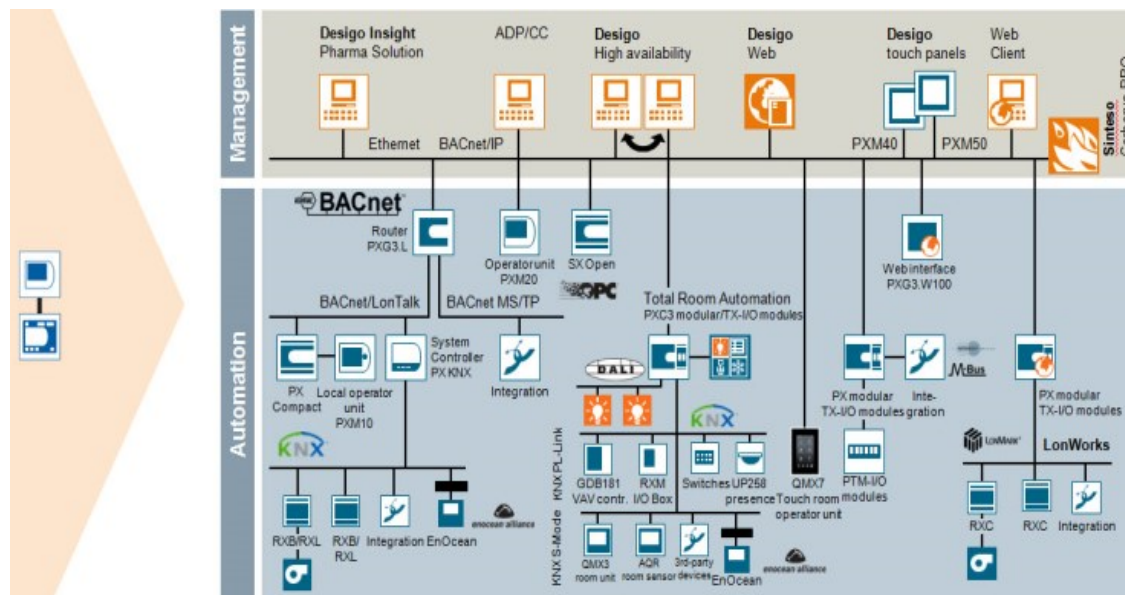


Figure 14. An example of Siemens Desigo consisting of various network standards, including KNX, BACnet, and LonWorks²⁰.

In the recent years, building automation started to adopt wireless sensors and actuators (WSAN). It offers a promising solution since it requires a minimal retrofitting work, which is very desirable in cases such as historical or preserved buildings (Acquaviva, Blaso, Dalmaso, Giudice, et al., 2012). WSAN eliminates costs related to cabling and enables a simpler

¹⁹http://www.hqs.sbt.siemens.com/gip/general/dlc/data/assets/hq/Desigo-building-automation---flexible-and-energy-efficient_A6V10227660_hq-en.pdf (Retrieved on July 30, 2014)

²⁰<http://www.siemens.de/buildingtechnologies/de/de/gebaeudeautomation-hlk/gebaeudeautomationssystem-designo/system-designo/topologie-designo/systemtopologien/Seiten/systemtopologien.aspx> (Retrieved on February 11, 2014)

installation. Battery powered WSN may require some maintenance effort for replacing the batteries, especially when the number of the nodes is quite high. However, there exist WSN nodes, which could harvest the ambient energy such as solar, thermal, wind, and kinetic enabling them to work autonomously with almost zero maintenance (Vullers et al., 2010).

Protocol	Zigbee	Z-Wave (Zensys Corp)	EnOcean	UWB	Bluetooth	Insteon (SmartLabs, Inc)	Wavenis (Coronis System)	Wi-Fi	6LoWPAN
IEEE Standard	802.15.4	-	-	802.15.3a	802.15.1	-	-	802.11a/b/g	802.15.4
Frequency Band	2.4 GHz, 915 MHz, 868 MHz	868/908MHz, 2.4 GHz (400 series only)	868 MHz	3.1-10.6 GHz	2.4 GHz	904 MHz	433 /868/ 915	2.4 GHz; 5 GHz	2.4 GHz, 915 MHz, 868 MHz
Data Rate	20/40/250 K/s	9.6Kbps/40K bps, 200 kb/s	125 kbit/s	110 Mb/s	1 Mb/s	38.4 K/s	4.8/19.2/100	54 Mb/s	20/40/250 K/s
Modulation	BPSK/BPSK / QPSK	FSK /GFSK	ASK	BPSK, QPSK	GFSK	FSK	GFSK/PSK	B/QPSK, UOFDM, QAM	BPSK/BPSK / QPSK
Spreading	DSSS	No	No	DS-UWB, MB-OFDM	FHSS	No	FHSS	DSSS, CCK, OFDM	DSSS
Communication Range(m)	10-100	30 (in) 100 (out)	30 (in) 300(out)	10	10	45 (out)	200 (in) 1000 (out)	100	10-100
Security	AES	AES-128	Basic	AES	E0 Stream AES-128	Rolling codes, public-key	3 DES 128AES	RC4 Stream / AES Block	AES
Error Control/ Reliability	16-bit CRC, ACK, CSMA-CA	8-bit CRC, ACK, CSMA-CA	-	32-bit CRC CSMA - CA	16 -bit CRC	8-bit checksums	BCH (32,21) FEC	32-bit CRC	16-bit CRC, CSMA-CA, ACK
Network Size	64000	232	2 ³²	8	8	256	NA	2007	2 ⁶⁴
Internet connection	Gateway Required	Gateway Required	Gateway Required	Gateway Required	Gateway Required	Gateway Required	Gateway Required	Gateway Required	Gateway NOT required
Logistic	Standard	Proprietary	Proprietary	Standard	Standard	Proprietary	Proprietary	Standard	Standard

Figure 15. Comparison of wireless technology used in home and building automation (Rathnayaka et al., 2011).

The manufacturers in building automation have developed proprietary WSN. However, in the recent years, the number of standards related to low-rate wireless personal area networks (LR-WPANs) has grown rapidly such as the IEEE 802.15.4 (J. A. Gutierrez, 2004) derivative standards including ZigBee (Z. Alliance, 2009), Z-Wave, and 6LoWPAN. EnOcean, an energy efficient wireless protocol. The latest media and home office devices such as smart TV, wireless speakers and printers usually support Bluetooth and Wi-Fi that can communicate with computers and mobile devices. As depicted in Figure 15, these standards have different properties such as the operating frequency band, data rate, communication range, and the adoption of open standards. These properties affect the development and operating costs, maintenance efforts and the quality of the communications that must be considered when designing IoT systems. For instance, when transmitting numeric sensor data such as temperature readings from battery powered devices over a wireless medium the system designer must find the most power efficient solution such as IEEE 802.15.4 based protocols such as ZigBee, Z-Wave, EnOcean, and 6LowPAN, which are suitable for transferring data up to 250 kbps. The designer should also consider the density of the network and the range when choosing the appropriate protocol, e.g.; ZigBee network is able to accommodate 64000

nodes while Z-Wave is only able to accommodate 232 nodes. Other factors such as openness of the protocol, which may influence licensing cost and interoperability with other systems could also influence the decision on which technology to be used.

In conclusion, different features, system requirements, and cost optimization are the contributing factors to the heterogeneity of IoT technology, which complicates the software developments. The developers must deal with different software interfaces and different programming models provided by the gateways. Therefore, providing an abstraction to these different technologies on the software level are urgently required.

Integrating the existing home and building automation devices into the current Internet is rather challenging. The internet relies on TCP/IP network which is optimized to transfer significantly larger data compared to the data being transferred on the Fieldbus or WSA. When TCP/IP is used for the control networks, it imposes unnecessary overhead, which could reduce the efficiency of a real-time control network (Jung et al., 2012). Therefore, gateways are normally used to bridge the data networks and the control networks to keep a clear separation between them to preserve control networks working deterministically in a real-time environment while assuring the data networks transmitting large data reliably.

Acquaviva et al. investigated the integration of the existing BMS with the emerging wireless networks such as ZigBee and EnOcean²¹ (Acquaviva, Blaso, Dalmasso, Del Giudice, et al., 2012). They keep each different network isolated and used gateways to maintain the necessary communication between them and the applications on the TCP/IP network. Their solution allows applications to retrieve the energy consumption data and perform control to the lighting and HVAC to optimize the energy consumptions.

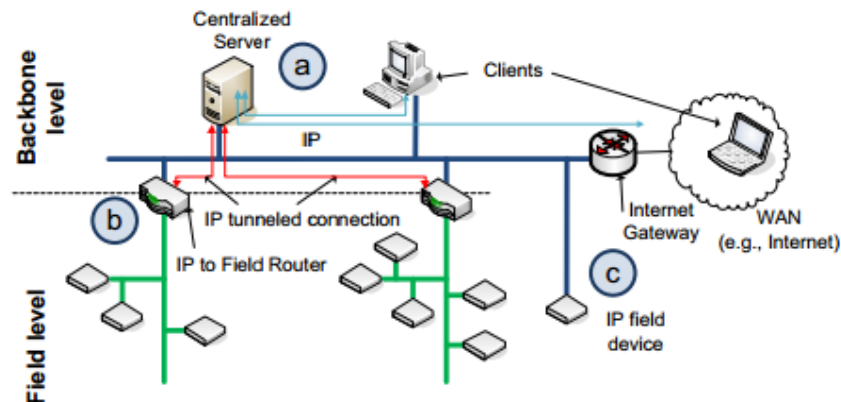


Figure 16. Possible solutions to integration existing BMS to IoT (Jung et al., 2012).

Jung et al. (Jung et al., 2012) identify there exist three possibilities to integrate the BMS to the IoT scenario which can be done through:

1. A centralized server that encapsulates various Fieldbus networks and provides a uniform access through an IP network.

²¹<http://www.enocean.com/en/home/> (Retrieved on August 10, 2014)

2. Field to IP gateway/router that acts as a proxy to the communication from and to the internet.
3. Equip field devices with an IP interface that can be accessed from the internet. This eliminates the requirement of a gateway.

These approaches conform to the author's understanding of IoT that physical objects with a compatible communication medium can be directly integrated into IoT while the devices that do not have a compatible communication medium must be represented by proxies running on a server or gateway.

Home and building automation are often related to energy efficiency. Being able to sense contextual information, whether the devices are required by the users or business processes in the corresponding space could be used to optimize energy consumption e.g., By turning off the lighting when they are not required or by reducing the power needed by the HVAC system when the space is not currently occupied. Moreover, environmental conditions such as outdoor temperature could be used as a parameter in defining HVAC control strategies.

Intelligent homes and buildings may even communicate with the electricity grid to balance the electricity load in a district, which could help avoiding power outages. Many research scenarios have envisioned that sustainable homes and buildings equipped with renewable power generators such as photovoltaic panels actively supply electricity to the electrical grid (Ghatikar, 2010; Kiliccote et al., 2011). Having a seamless communication between the electrical grid and the buildings in the district allow a better management for the power demand and supplies and perform a more accurate forecast of the electricity demands. GREENCOM²² project is conducting research that allows the utility providers to intervene the electricity demands to reduce consumption peaks by providing an incentive when the users allow them to control their home appliances remotely. For instance, the utility provider may reduce 1-2 degrees of the heater's temperature, or allocate schedules to the washing machines on every home, so they do not run at the same time.

3.4 Smart Grid

Smart Grid is an effort to make our power grid smarter by allowing communication between smart meters and the electrical grid. The potential of Smart Grid has been recognized in all over the world. The communication between the actors allows them to negotiate power demand and supply in the network in order to create a more balanced distribution of consumptions and avoids unpredictable peaks that may overload the whole power grid. The conventional power grids are not very efficient since there is a huge gap between electricity demands on the peak hours and off peak hours. The utility providers must invest in the required power generators for handling the highest peak, which only last for several hours in order to ensure that continuous power supply to the consumers. Moreover, the conventional power industry has only a little to none of storage capacity since the cost of power storage is quite high (H. Chen et al., 2009). This means the generated power must be directly used to stabilize the power network.

Enabling communication between consumers, distributors, and producers could help reducing the difference between on and off peaks since the consumers and providers could negotiate

²² <http://www.greencom-project.eu/project-description.html> (Accessed on August 8, 2014)

the supply and demand and the providers could estimate the load on the power grid more accurately based on the data collected from the consumers. Therefore, smart grid could increase the efficiency of the power network and reduce the need of infrastructure investments.

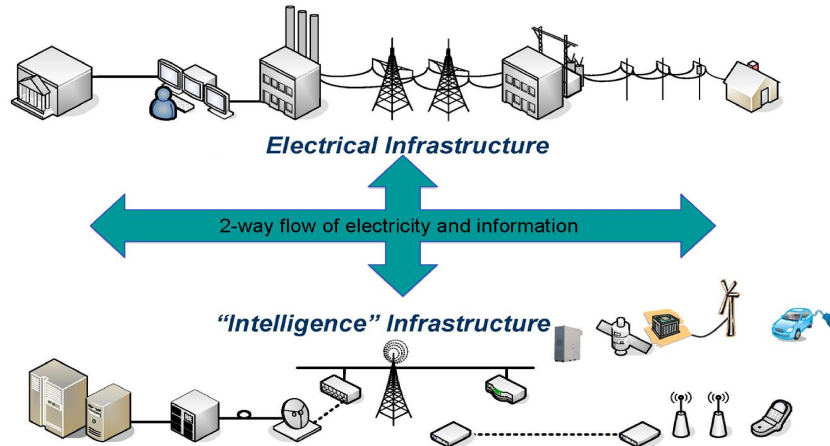


Figure 17. Smart Grid vision²³

Moreover, smart grid also envisions that power generations do not need to be centralized anymore. Smart Grid allows power to flow bidirectional from distributors to the consumers and from the consumers to the distributors. These consumers who have the capability to produce power from renewable energy sources and supply this power back to the grid are called prosumers (Silva et al., 2012). The power distributors act as brokers that receive power from prosumers and power generators and distribute them back to the consumers. Consequently, this will ease the load of the power plants.

3.4.1 Communication in Smart Grid

Communication in smart grid varies from country to country, and some of them do not regulate any communication standard between the smart meters and the utility providers. The communication could be divided into two. First is the bidirectional communication between Distribution System Operators (DSOs) and consumers through the smart meters and secondly the communication inside the consumer premises, which is influenced by home automation and smart appliance manufacturers.

In order to gain profit from smart grid, the DSOs must be able to analyze and predicts the energy load on their network. Therefore, they are required to collect data from hundred thousands of smart meters frequently. Enabling this requirement, the communication between smart meters and DSOs must work on a low latency and be able to handle a burst of messages from the smart meters. A real-world analysis of power meter traffic was done from BC Hydro installation involving 1.9 million meters (90% residential and 10% commercial) that are served by about 1,700 collectors (Wenpeng et al., 2013). The average upload traffic per

²³ <http://www.nist.gov/itl/antd/emntg/smartgrid.cfm>

day is about 3,185 bytes and 272 bytes/day for the download. The data from each meter is aggregated at an aggregator installed on each neighborhood area network (NAN) which then uploaded to the DSO through a satellite link (WAN). Figure 18 shows the monthly amount of data uploaded by the aggregators to the central server through a satellite link. It shows that the server must be able to handle several hundred megabytes for each aggregator monthly.

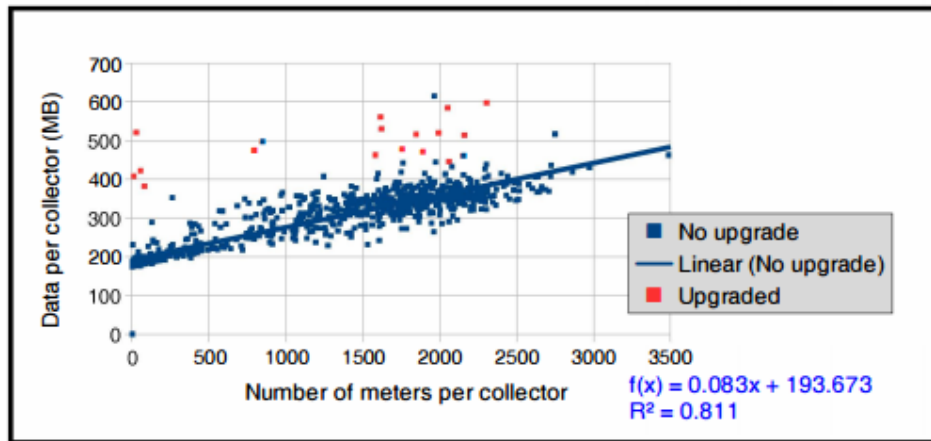


Figure 18. An example of monthly smart meter traffics sent to the DSO by the aggregators (Wenpeng et al., 2013).

Several techniques can be used to facilitate the communication between smart meters on the customers' sites and the DSOs. For instance, cellular network, power line or dedicated wire line. The power line communication (PLC) seems to be the optimal solution since they do not need to invest cabling infrastructure anymore. The PLC can be used for communication in high to low voltage network. However, PLC has some problems. First, attaining a low error rate over a long distance through PLC is challenging since the channel over power line is quite noisy (Aalamifar et al., 2012). Second, the cost of the PLC modems is still too expensive for penetrating mass market. Third, they present an electromagnetic compatibility (EMC) issues (Galli et al., 2011) since there exist several standards that interfere each other when they are used in the same medium. The PLC technologies can be categorized into three classes. First, the Ultra Narrow Band (UNB) which operates on 0.3-3kHz and has a data rate of approx. 100bps. Second, the Narrow band (NB) technology, which operates at 3-500 kHz band and has a bandwidth of few kilobits up to 500kbps. Third, the broadband technology, which operates in 1.8-250 MHz and a bandwidth of several megabits to several hundred megabits per second.

Alternatively, Wireless technology provides a viable solution since there is no need of having a huge initial investment for installing the cable infrastructure. There are various wireless technologies that could be used for enabling communication in various part of smart grid networks (depicted in Figure 19).

Wireless Technology	Data Rate	Approx. Coverage	Potential Smart Grid Applications
Wireless LAN	1-54Mbps	100m	distribution protection and automation
WiMAX	70Mbps	48Km	Wireless Automatic Meter Reading (WMAR)
Cellular	60-240Kbps	10-50km	SCADA and monitoring for remote distribution
ZigBee	20-250Kbps	10-100m	Direct load control of home appliances
MobileFi	20Mbps	Vehicular Std.	communication for PEVs and remote monitoring
Digital Microwave	155Mbps	60 km	transfer trip (point-to-point)
Bluetooth	721Kbps	1-100m	local online monitoring applications

Figure 19. Comparison of wireless technologies for smart grid (Parikh et al., 2010)

For instance, Wi-Fi, ZigBee, Bluetooth are suitable for the communication in houses and buildings since their range is up to 100m. While WiMax, Digital microwave and cellular networks are suitable for the communication between DSOs and consumers. Between the three, cellular network shows the most promising approach since most residential and business areas are already covered with cellular service providers. Cellular networks in the recent years have developed rapidly and allow high bandwidth data transfer from 21Mbit up to 300Mbit through 3G and LTE networks.

In the context of IoT, many scholars and practitioners proposed to apply the ETSI M2M (ETSI, 2011) architecture for enabling the communication between the smart meters and the utility providers over cellular networks. In addition, the home automation networks play significant roles enabling the vision of intelligent, smart grid where communication not only exist between the utility provider with the smart meters but also up to the device level. IoT technologies such as 6LoWPAN standard could provide a more standardized IP-based communication. Alternatively, gateways could be used to expose the communication between the power grids to the field devices through an IP network.

3.5 Conclusion and Common Requirements towards IoT

Having investigated the application domains of IoT, the author concludes that the technology used in different domains may overlap but also could vary greatly. The diversity is most likely influenced by the functional requirements, costs, and the driving forces in that domain. For instance, the communication technology is influenced by specific requirements such as the bandwidth, transmission rate, and reliability. The driving force such as vendors, regulatory bodies also influence the technology in the domain. E.g., WirelessHART is quite popular in industrial automation, while ZigBee and Z-Wave dominate the home automation domain. The radio frequency of WSAAN diverse from country to country depending on the regulations. The adoption of the technology is also accelerated when the standardization or regulatory bodies have release recommendations.

In addition, the scale of IoT varies greatly. It can involve few devices as can be found in home automation, few thousand such as in commercial buildings and industrial automation and few

hundreds thousands such as in the smart grid scenarios. The scale of the network may also include home and local area networks, closed wide area networks, and internet scale network.

Analyzing these application domains, the following high-level requirements for IIOT systems can be summarized as follows:

- There is a need to define a common reference architecture that provides an abstraction for heterogeneous communication technologies and architectural patterns that can be used as a guide when integrating different technologies. In a further step, the common abstract architecture must be defined more detail within the application domains to capture architectural commonality in each application domain.
- Support for legacy systems and domain specific technology.
Bringing IIOT technology into a specific application domain requires an integration of it with the existing systems that use domain specific standards or proprietary technology. IIOT must consider ways to integrate these various technologies.
- Integrating IIOT into existing domain should respect domain-specific requirements and constraints.
Sometimes these technologies were designed to address specific requirements in that domain e.g., In industrial automation, real-time and deterministic behavior are hard requirements which ensure that the behavior of the whole system is predictable. Therefore, the integration of IIOT technologies must not interfere any domain specific requirements and constraints to ensure the system runs as expected.
- Support vertical, horizontal, and longitudinal integration.
Nowadays, information must be able to be exchanged between different systems used within departments in organizations known as horizontal integration, between departments known as vertical integration, and between organizations known as longitudinal integration. Integration must ensure that information could be delivered between these units seamlessly regardless of the technology used in the unit e.g., The sales department could place a production order which is decomposed into smaller processes and forwarded to the automated production line automatically.
- Device Resource heterogeneity.
When integrating devices into IIOT one should also consider that the resources might be very scarce on some devices e.g., Transmission rate, power supply, transmission range, computational power, and the amount of memory while on other devices, computing resources might not be of a concern. Therefore, communication and cooperation between these devices must be done in a way that the resource-constrained devices are not overwhelmed by the communication protocols.
- Maintainability of devices
When designing IIOT, one should consider approaches to minimize the maintenance efforts required by different components. For instance, wireless sensors that are installed in places that are hard to reach must be able to operate without maintenance such as replacing batteries or configuring the network for a very long time.
- The users and other components must be able to address the thing uniquely.
Since addressing schemes the scope of IIOT could vary, there is a need for encapsulate these schemes and provide a uniform addressing scheme that can be used to address specific devices and physical objects.
- Different type of developers must be supported.

IIOT development involves a broad range of technologies that requires different experts such as embedded developers, enterprise software engineers, electrical / automation engineers. Some systems, even address end-users with little to none programming experiences which require another programming paradigm than professional engineers.

- **Sharing IIOT resources**

There are several reasons why IIOT resources should be shared such as the space limitation, the cost of acquiring and maintaining IIOT, the lack of expertise required to operate the resources (Gluhak et al., 2011). These reasons motivate sharing IIOT resources between applications and inter-organizations. Sharing IIOT resources for business and research purposes is able to maximize the profit by minimizing the operational costs, as well as accelerate the progresses in research.
- **Discovery of devices over a wide area network or Internet**

When a large scale of a distributed system is required, such as for the Smart Grid and Smart City scenario, devices might be shared between diverse applications through a wide area network / internet. These applications and developers must be able to discover these shared devices in order to use them.
- **Collect, process and store large amount of data and information generated by IIOT**

IIOT systems can be developed for collecting data which need to be analyzed at some nodes in the network and stored. These nodes must be able to receive data and process as well as storing them.
- **Support mobility of IIOT**

IIOT should support sensing of mobile objects that move from a location to other locations. This requires a dynamic association between the sensors and the object being observed. Additionally, IIOT must also support sensor that moves around and sense several objects along the path. This could be done to reduce the costs of having many sensors fixed to a location. This type of use case also requires dynamic association between the sensors and the object which is currently being observed.
- **Support human involvement**

Although IIOT development is going in the direction of autonomous systems where user interventions could be minimal, IIOT should foresee that human users may interact with the systems to obtain the results that they would like to have. Therefore, IIOT should consider human errors or inconsistent instructions from the users.
- **Support a massive size of wireless network nodes and other devices**

Some applications require a massive number of nodes being used to observe physical events. For instance, to detect fire in forests, thousands of nodes may be deployed which should work together to inform us when a fire is detected. The scale of the nodes causes additional requirements for managing these devices with minimal human intervention such as plug-and-play configuration and fault management.
- **Security and privacy**

The current IIOT system must be able to ensure that data and information can only be accessed by the authorized parties.
- **System performance of different components in the network may not be influenced by another component. E.g., Different performance requirements between computer networks and Fieldbus devices must be preserved when integrating these two different networks.**
- **Reliability of messages**

In certain domains, the reliability of messages is strictly required. For instance, the emergency stop button on heavy machines must stop all mechanical parts instantly when it

is pressed. The order of messages could also be important, e.g., for the systems that rely on state-machines.

These high-level requirements show different aspects that system designers should consider when building IoT systems. System designers must select and prioritize the requirements that are relevant to the goal of the systems. For instance, for building IoTLink, the author chooses to prioritize requirements that provide the primary functions of IoT without over complicating the development efforts so that developers could produce the functional prototypes rapidly. When required, the developers could still extend them with additional features. The author concludes that the highest priority requirements for IoT includes:

- Establishing communications between devices and applications,
- Facilitating the addressability of devices
- Providing an abstraction to heterogeneous devices, according to the reference architecture
- Enabling sharing and discovery of IoT

Because of this reason, in the Chapter 4, the author presents the relevant the state-of-the-art in IoT architectures in order to summarize the required reference architecture. Moreover, different IoT development approaches are elaborated to inspire the design concept of IoTLink.

Chapter 4.

State-of-the-art in IoT

Architecture and Development

Platforms

This chapter describes the state-of-the-art in IoT developments, including the EPCglobal architecture which initiates the use of the “Internet of Things” term. In addition, this chapter discusses the further IoT architectural patterns and the architecture reference model proposed by IoT-A a project that was funded to standardize IoT architecture. In addition, various development platforms and approaches are presented such as middleware that provide an abstraction for different IoT technologies, development platform that is intended for professional developers as well as end users. These technologies provide an overview that is useful as consideration for designing IoTLink.

4.1 IoT Architectures

One of the initial IoT architecture was defined by the Auto-ID Labs when they tried to find a solution for enabling product traceability in supply chain management (Bose & Pal, 2005). They started the EPCglobal standard and proposed architecture for identifying and retrieving information about physical objects tagged with RFID (Johnson et al., 2005) (Figure 20). Their architecture requires a centralized service registry, called Object Naming Service (ONS) that manages the address of the services that have information about the tagged products. When, an application needs to retrieve the information about a product, it has to obtain the address of the responsible service from the ONS. Then it could retrieve the information directly from the corresponding service.

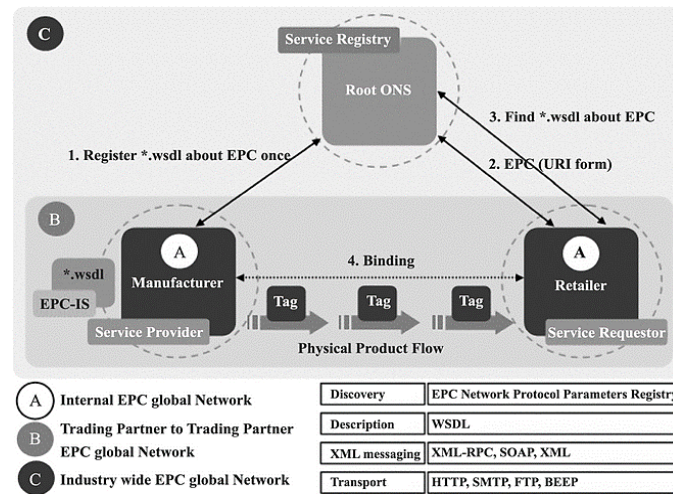


Figure 20. EPCglobal Architecture(Shih et al., 2005)

According to a survey, many works suggested that IoT should not only focus on getting information, but also aiming at smart objects that are able to think, react, and cooperate autonomously (Atzori et al., 2010). This perspective obviously raises two fundamental research questions. First, how physical objects can be represented in the virtual world and how the smart objects could communicate and cooperate.

Answering the first question, Serbanati et al. define the characteristics of resources, as a digital representation of physical entities, to have the following fundamental property: “they are digital entities that are unequivocally associated with the physical entity they represent. Each resource must have only one ID that identifies the represented object. The association between the resource and the physical entity must be established automatically” (Serbanati et al., 2011). The concept of a smart object is not entirely new and has been explored other computer science field, for instance, pervasive computing, context awareness, and augmented reality. Siegemund et al. presented an approach that enables communication between everyday objects tagged with Bluetooth nodes with a handheld device to retrieve information about the objects (Streitz et al., 2005). Rukzio et al. present an approach enabling interaction between smart phones and posters which are tagged with QR code (Rukzio et al., 2004). Kranz et al. present an approach to reflect the state of the physical objects within the virtual world, e.g., the position of a knife when it is used for cutting, a cutting board that senses the weight, a sitting cushion that is able to sense the sitting position (Kranz et al., 2010).

From an engineering perspective, “Things” can be represented as smart objects in different level. On the physical world, sensors are required to sense the context of the objects and actuators are required to affect the state of the objects. On the software layer, the sensor values are received as raw data that might need to be processed to determine the actual state of the physical objects. When using an object-oriented paradigm, the physical objects can be modeled as software objects. When stored in a relational database, the physical objects might be represented as tables and their state at a point in time may be stored as a row of data in the respective table. On the network, when representing “Things” in a service oriented architecture, they can be represented as Web Services (Guinard, Trifa, et al., 2010). When

exposing them through the web protocols, they can be presented as web resources (Guinard et al., 2010a).

The IoT-A consortium (Nettsträter et al, 2012) proposes a generic domain model containing abstract concepts involved in IoT systems generally (depicted in Figure 21). The architecture describes a domain model that contains the interaction between entities in an IoT system. The *User - Physical Entity* interaction is the core relationship in this model. Both sides are decomposed into more detail subclasses, including devices (blue rectangles), software artifacts, services, and resources (green rectangles) or human users (yellow rectangles). The IoT-A model provides quite simple concepts and relations that can be extended for representing any IoT systems. The domain model was built based on iterative requirement engineering done through expert workshops and study of previous EU funded IoT projects such as Sensei²⁴, Aspire²⁵, and Hydra²⁶ (Pastor et al., 2011). In addition, they evaluated the architecture against future IoT scenarios, including diverse domains such as eHealth, transportation/logistic, retail, smart city, smart home, and smart factory. From this perspective, the IoT-A domain model provides a suitable reference architecture that can be extended to develop IoTLink, targeted in this work.

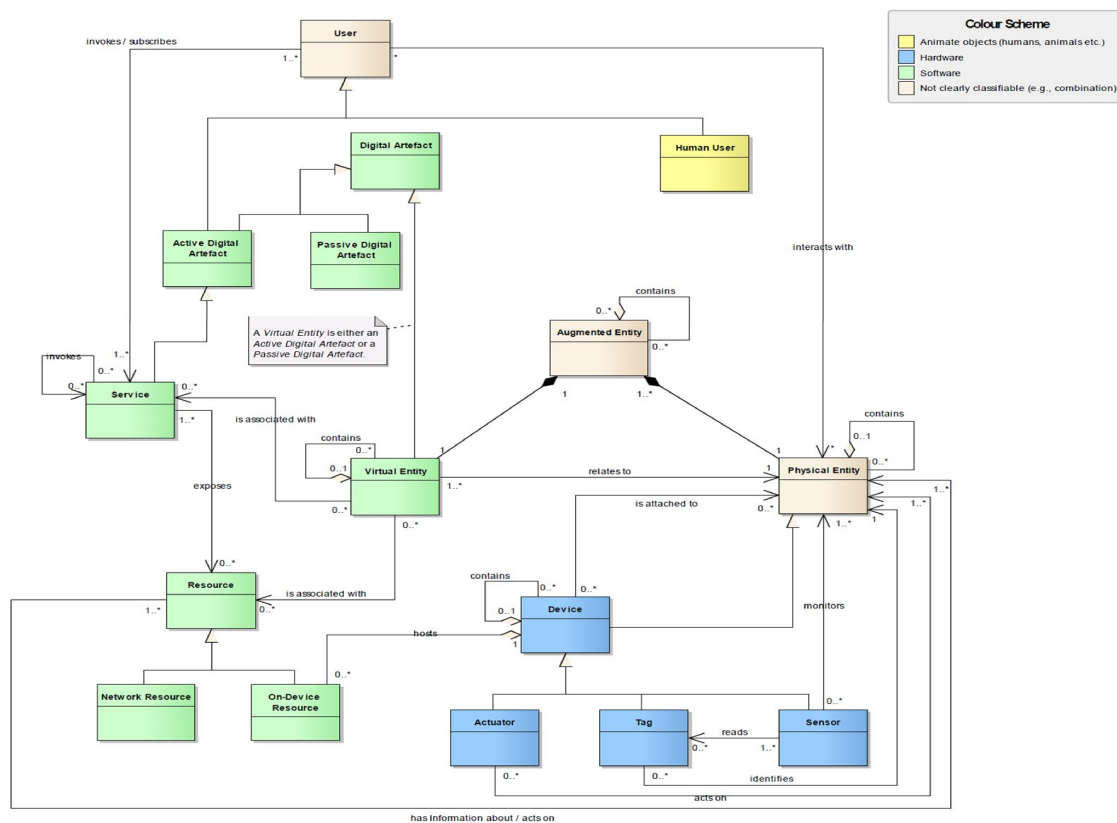


Figure 21. The IoT-A Architecture Reference Model (ARM)

²⁴ <http://www.sensei-project.eu/> (Retrieved on Oct 21, 2014)

²⁵ <http://www.fp7-aspire.eu/> (Retrieved on Oct 21, 2014)

²⁶ <https://linksmart.eu/redmine> (Retrieved on Oct 21, 2014)

4.1.1 Communication between Things and the Internet

Communication between physical things can happen in different levels. At the most granular level, electronic components such as FPGA and microcontrollers are integrated and communicating through high-speed on-board bus networks such as FPGA I/O which is able to transfer data up to 1.6 gigabits per seconds (Tyhach et al., 2005) and PCI express which is used for communication between PC peripherals (Anderson et al., 2004). On the local area networks, most PCs and consumer electronics communicate through TCP/IP based networks through wired or wireless network. However, in the embedded and industrial IT, Fieldbus networks are frequently used since they are designed to operate deterministically and in real-time environments. Examples of Fieldbus networks are Modbus (Modbus, 2004a) for building automation and CANBus (Semiconductors, 1996) for vehicles. In addition, wireless standards such as Wi-Fi and Personal Area Networks such as Bluetooth, ZigBee are used more frequently to facilitate the interconnections between mobile devices and home appliances. Despite the heterogeneous network protocols on different levels, TCP/IP based communication has become a de facto standard on the Internet, which makes all communication across the internet is possible. On the internet, various applications utilize different protocols built on TCP/IP such as HTTP, which is used for to transmit web traffics, SMTP and IMAP which are used to exchange emails traffics.

OSI (Open Source Interconnection) 7 Layer Model

Layer	Application/Example	Central Device/ Protocols	DOD4 Model
Application (7) Serves as the window for users and application processes to access the network services.	End User layer Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management	User Applications SMTP	G A T E W A Y Process
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	Syntax layer encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • Character Set Translation	JPEG/ASCII EBDIC/TIFF/GIF PICT	
Session (5) Allows session establishment between processes running on different stations.	Synch & send to ports (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.	Logical Ports RPC/SQ/NFS NetBIOS names	
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	TCP Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	F I L T E R I N G PACKET	Host to Host
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	Packets ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		R O U T E R S IP/IPX/ICMP
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	Frames ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgement • Frame delimiting • Frame error checking • Media access control	Switch Bridge WAP PPP/SLIP	Can be used on all layers Network
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	Physical structure Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts	Hub Land Based Layers	

Figure 22. The seven OSI layers²⁷

The communication between computer networks has been designed following the 7 OSI layer (H. Zimmermann, 1980; Day & Zimmermann, 1983) which aims at providing a standard open model and decoupling solutions on every layer of computer network. Since then this model has been responsible for modularizing the development of computer networks, including the Internet as we know today. The OSI layer design enables communication protocols in each

²⁷ <http://www.escotal.com/osilayer.html> (Retrieved on Oct 21, 2014)

layer being developed independently. The modularization makes it possible to exchange protocols in different layers without having to modify the whole network stack. For instance, TCP/IP and the protocols above it are able to work with both wired Ethernet and 802.11 Wi-Fi (IEEE, 2010).

The IoT development requires a model that fits the existing Internet architecture while addressing its unique characteristic such as a massive number of interconnected Things. As a consequence, the IoT architectures must be able to handle heterogeneity in many different areas, e.g., communication protocols, data format, computing resources.

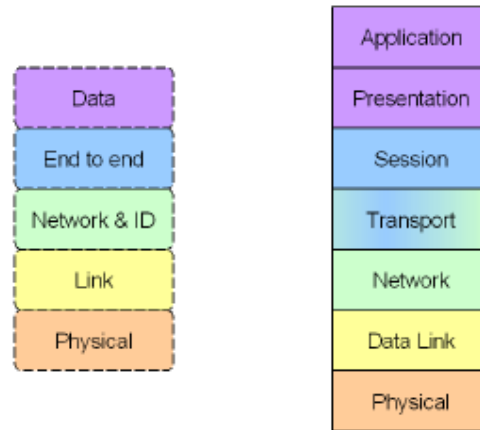


Figure 23. IoT communication layers according to IoT-A (IoT-A, 2013).

There exists diverse IoT architectures which are relevant to integrate IoT into the existing internet infrastructure. However, IoT-A's architecture shows the most generic approach based on an extensive literature study and workshops. Their architecture is also meant to be the IoT reference architecture. They propose a five-layer communication model (depicted in Figure 23), which strains the interoperability aspect. Similar to the OSI model, the lowest layer is concerned with the physical aspect of the network. The physical layer does not enforce any specific technology and is responsible to support interoperability between devices with different physical capabilities e.g., enabling low-power radio transceiver such as ZigBee to communicate with a smartphone that has a Wi-Fi or 3G through a gateway. The second layer is concerned with heterogeneous communication technologies, providing abstractions and standardized capabilities for the upper layers. The network and ID provide similar network capabilities as the corresponding layer in the OSI model. In addition, it provides an identification scheme, which enables managing IoT globally and able to deal with different network technologies in order to make heterogeneous systems addressable from one another regardless of the technologies they use. E.g., a temperature sensor running on a ZigBee node must be able to send the values to a smartphone, and the smartphone must be able to request the temperature of the environment to the sensor. The end-to-end aspect focuses on reliability, transport issues, translation functionalities, proxies or gateways support and parameter configuration for cross network communications. Additionally, this layer handles some application aspects related to communication such as congestion controls and acknowledgements that are required by the applications. The data layer is concerned with modeling data exchanged between two or more actors. The data aspect needs to include a

structured data description that can be translated by different techniques such as compression (e.g., XML to EXI), decompression (e.g., CoAP to HTTP), and mapping (e.g., IPv4 to IPv6). Moreover, this layer must consider constrained devices (e.g., translating binary data from sensor nodes in XML format for better readability on more powerful nodes).

4.1.1.1 Communication Interoperability for different networks

The IoT tries to achieve an integration of heterogeneous devices, which involve different type of networks. Typically, sensor network protocols are more constrained than PC networks in terms of data rate and availability since they need to consider devices with limited computing power as well as power sources. For instance, ZigBee (Z. Alliance, 2009), which is based on IEEE 802.15.4, is only able to transfer 250 Kbps (Gang Lu et al., 2004) while IEEE 802.11ac Wi-Fi is able to transfer 433-867 Mbps (Ong et al., 2011).

Communication between different network technologies can be enabled by an entity that perform a translation service. IoT-A recognizes that there are two different possible configurations for translation service between networks (IoT-A, 2013). First, a gateway may translate two or more protocols on the same layer in a network stack across different communication media by extracting the content, exchanging the originating header into the header that can be understood by the destination. For instance, the left side of Figure 24 illustrates a gateway that translates IP packages, travelling through Ethernet into Wi-Fi. The right side of Figure 24 shows the second gateway that translates packages on several layers. For instance, IP to 6LoWPAN, TCP into UDP, and HTTP into CoAP messages.

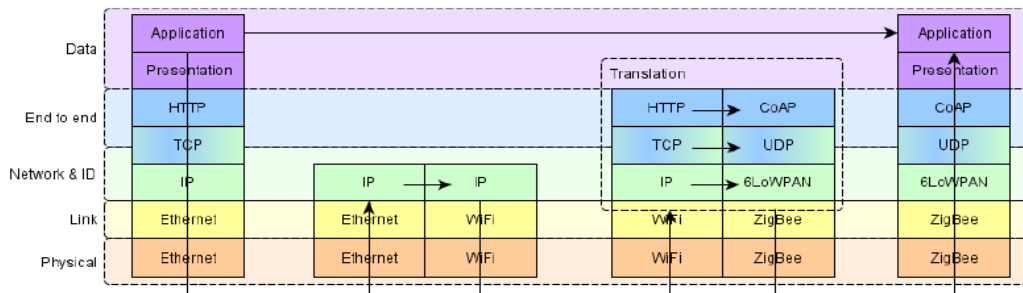


Figure 24. Gateway translating the lower layer protocols (IoT-A, 2013).

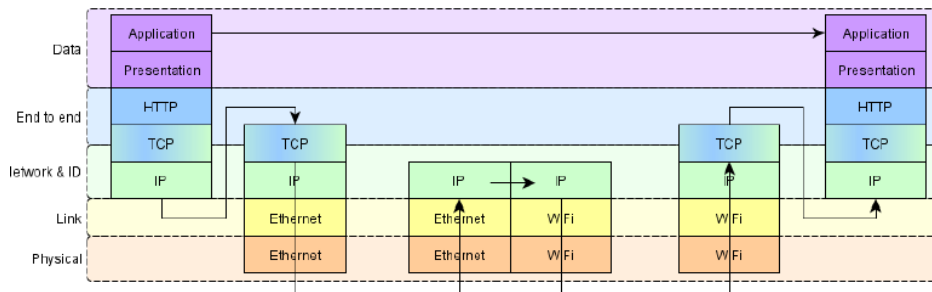


Figure 25. A Gateway that translates the lower layer protocols by piggy backing the content into a protocol in a different layer (IoT-A, 2013).

The second configuration uses virtual configurations as the composition of two or more protocol stacks one on top of the other. Figure 25 illustrates a gateway that piggybacks IP messages into TCP messages. Before the messages reach the destination, they must be extracted from the TCP messages by a gateway then sent as an IP message to the destination.

4.1.1.2 M2M Architecture

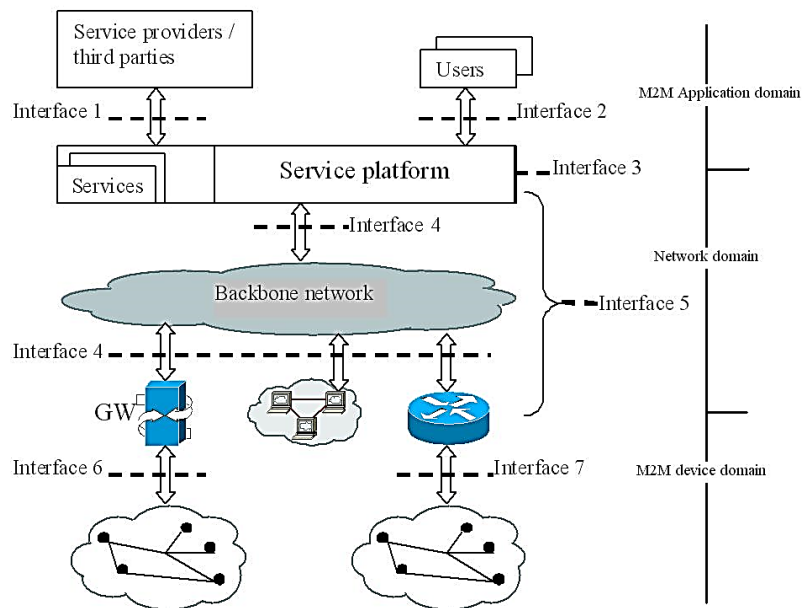


Figure 26. M2M Network Architecture

Machine-to-Machine (M2M) communication has existed since the birth of distributed computing and industrial automation. However, back then many proprietary solutions were used which resulted in heterogeneous silos of proprietary networks. Today, these heterogeneous technologies are able to communicate with each other to some extents using network bridges and middleware that are able to translate a protocol into another. Nonetheless translating these protocols requires enormous efforts and some features might be lost during the translation.

In the recent years, many organizations realize that open standards are quite important to allow a seamless cooperation between machines without having to deal with the aforementioned problems. The ETSI proposes an architecture for enabling M2M communication. This architecture provides a set of specifications describing an interoperable communication between service providers, electronic appliances, and industrial machines particularly in Europe. It specifies on open interfaces between components depicted in Figure 26, which includes:

- Interface 1 specifies the interface for application developers and service providers.
- Interface 2 exposes XML Web Services for customers.
- Interface 3 is specified for remote and inter-operator cooperation (load management, reduced traffic capacity, etc.)
- Interface 4 is the standard interface towards the backbone network.

- Interface 5 provides a range of supports for connecting Objects (COs) over the Interface 4 (Internet).
- Interface 6 specifies a communication with the application specific or proprietary protocols.
- Interface 7 specifies communication to low-power devices using ETSI defined protocols such as 6lowPAN (Mulligan, 2007) and CoAP (Niyato et al., 2011; Zach Shelby et al., 2013).

M2M architecture provides a suitable model to enable a standardized smart meter and smart grid infrastructure in Europe (Fan et al., 2010; Fan et al., 2012)

4.1.1.3 Publish-Subscribe communication

In addition to the M2M architecture, there exist several proposals of message-based communications based on publish-subscribe communication pattern. Publish-subscribe pattern is an additional communication pattern which is useful to push the data from the transmitter to the receiver. Publish-subscribe can be implemented through point-to-point communication architecture where the subscriber and publisher communicate directly. In a local network where connections are reliable and can be kept open continuously without straining the network, point-to-point architecture could be applied. However, communication through the internet could be unreliable and keeping the connections between things continuously opened could put the internet under a lot of stress.

Alternatively, publish-subscribe pattern could be implemented through mediation of a central server known as a broker. The broker maintains a list of subscribers and their interest on specific topics. When the publisher publishes an event, the broker notifies the interested subscribers. Figure 27 shows an example of two clients subscribing to an event. When the service publishes the event, the broker finds the two clients to be interested in the corresponding topic. Therefore, it notifies the clients.

Using a broker is very useful when the publisher and the subscriber cannot be always connected to the same network at the same time. The broker that is always on can be used to cache the published messages and forward them to the subscribers when they are connected. Because of this reason, broker architecture provides a promising solution for the IoT communication which sometimes may not be reliable.

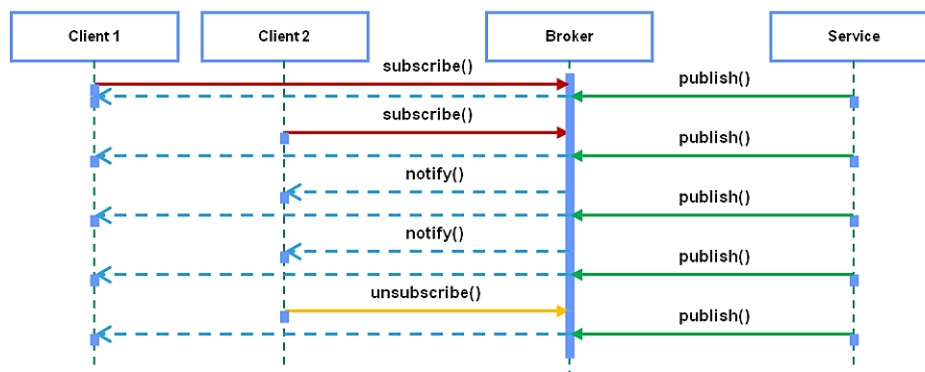


Figure 27. Publish-subscribe pattern involving two clients and a service (IoT-A, 2013)

Another advantage of using publish-subscribe with broker for IoT is that it decouples the communication interface between the subscribers and publishers. By providing the protocol

specification, subscribers and publishers can be implemented with diverse programming languages and deployed on diverse hardware platform. Moreover, few of these messaging protocols are designed to work on resource-constrained devices such as battery-operated sensor/actuator nodes that must operate under low bandwidth constraints. This enables IoT applications from the edge to the core of the network to use the same messaging protocols.

There exist several publish-subscribe brokers which have been proposed to enable M2M and IoT communications such as MQTT (Locke, 2010) that is designed as extremely lightweight publish-subscribe protocol and Advanced Message Queuing Protocol (AMQP) (Vinoski, 2006) which is widely used for passing messages between banking applications.

MQTT has gained a lot of support from the IoT community since it is quite simple to understand, the protocol is quite slim, and can be used for resource-constrained devices (S. Bandyopadhyay & Bhattacharyya, 2013) as well as unreliable connections.

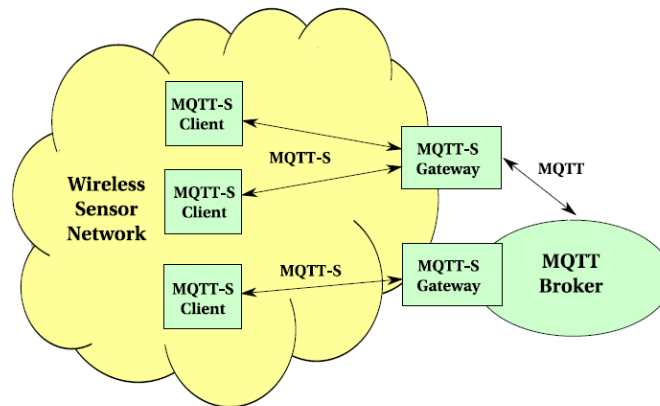


Figure 28. MQTT-S Architecture to enable communications between WSNs (Hunkeler et al., 2008)

MQTT is able to communicate through TCP/IP and has been implemented to work with other network protocols such as ZigBee (Stanford-Clark & Truong, 2013) and WebSocket. Several MQTT brokers and the client libraries are available in various programming languages.

4.1.1.4 Poll based communication

Polling is a communication pattern, which can be used as an addition to the push based communication, which is enabled by publish-subscribe pattern. Polling can be achieved by providing passive services that can be called. Polling based communication is useful when the clients only retrieve the status of IoT when they need them, and they do not need to know the state changes instantly as soon as the states have changed. Poll based communications can also be used to retrieve the status of the IoT when its clients are initialized.

Poll based communication can be enabled by Web Service technology such as Simple Object Access protocol (SOAP)²⁸ based Web Services (known as WS-* standard) as well as RESTful services (Fielding, 2000a) which provide network-accessible endpoints for accessing software

²⁸ <http://www.w3.org/TR/soap12-part1/#intro> (Retrieved on May 17, 2014)

components (Gottschalk et al., 2002). In IoT context, standard such as the Device Profile for Web Services²⁹ (DPWS) and CoAP (Z Shelby et al., 2012) have been designed to enable Web Services and RESTful services on resource-constrained devices.

4.1.2 Functional view of IoT architecture

ITU has proposed an architecture model describing the functional view of IoT, which is illustrated in Figure 29. This view has been adopted and extended by IoT research works such as the ARM done by IoT-A³⁰. The architecture suggests four layers including application layer on the topmost, followed by a service and application support layer, network layer, and device layer. In addition, it includes two layers that span vertically, management and security, since they are cross functional and can be used in the four main layers. The application layers comprise applications that use IoT resources. The service support and application support layer provides a generic support that can be used by various IoT applications, e.g., data storage and processing. In addition, it provides support that is specific to application requirements. The network layer provides two capabilities, including networking capabilities such as access and transport control functions, authentication, authorization, and accounting (AAA). And secondly, transport capabilities that facilitate the transmission of IoT data and information as well as transporting data related to network management functions.

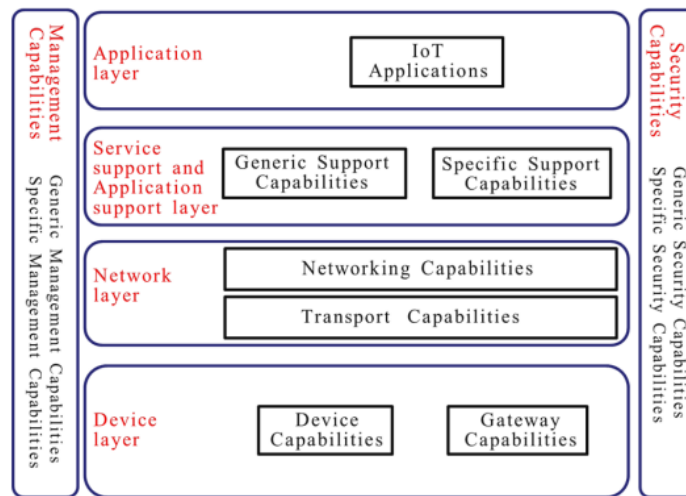


Figure 29. IoT reference model (ITU-T, 2012)

Device layer is divided into two capabilities, including device capabilities and gateway capabilities. Device capabilities include first, a direct interaction between the network and the IoT devices which does not require any gateway to send and receive data. Second, indirect interaction between IoT devices and the network which requires a gateway to send and receive data. Third, Ad-hoc networking which could be constructed dynamically in scenarios where scalability and quick deployment are required. In addition to device capabilities, the

²⁹ <http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01> (Retrieved on May 17, 2014)

³⁰ <http://www.iot-a.eu/arm/d1.3/view> (Retrieved on May 17, 2014)

device layer also includes gateway capabilities which integrate devices with different communication means such as wired, wireless and different network protocols e.g., ZigBee, Bluetooth, Controller Area Network (CAN). The gateway must provide protocol conversion to allow devices having different network protocols and data format communicating at the semantic level. The gateway may facilitate the connection to the internet through various technologies, including wireless such as 2G, 3G, LTE, and Satellite, or cabled connections such as DSL, TV Cable, and analog land line.

The Management capabilities may include generic management capabilities that apply to all IoT applications such as fault, configuration, accounting, performance, and security (FCAPS) managements as well as remote device, diagnostic, software update, traffic management. In addition, it could include application specific management functions, e.g., smart grid load management.

The security capabilities may also include generic such as authentications, authorization, privacy protection, and audits at different layers. Moreover, application specific capabilities may be utilized, e.g., ePayment, eHealth, SmartGrid may impose different security requirements, e.g., level of data encryptions and restrictions on storing data.

IoT-A has extended this functional view of IoT based on further studies and expert workshops. The result incorporates several concepts that have been introduced by other works such as “Virtual Entity” which is meant as a representation of physical things within the information world. Moreover, it contains extensive functional components, which can be summarized as follows: The device, application, management, and security layers are identical to the corresponding layers described by ITU architecture. However, the functional components in the middle focus on a broader scope than ITU. It introduces Service Organization, IoT Process Management, Virtual Entity, and IoT Service layers.

The service organization is concerned with composing and orchestrating services between other components as well as services offered by the physical devices. Service-composition requires a dynamic resolution of complex services based on the availability of the services as well as authorization of the users. Service orchestration finds and resolves the suitable services upon requests by the users. The choreography provides a broker function that allows publish-subscribe communication between services.

The IoT Process Management takes care the interaction between Things and business processes of the tools necessary to include the IoT within the business process model to the execution of the model.

The Virtual Entity provides an abstraction for the physical objects that simplify the interaction with the applications since they do not have to deal with different technology. It manages the associations to the physical objects. Additionally, it also provides functionality to discover, and retrieve information about the Virtual Entities.

The IoT Service is responsible for providing functionalities for discovery, look-up, and name resolution of IoT Services. IoT services could deliver information about physical devices such as sensor and actuator. Additionally, it could be used to control devices. IoT Service resolution is responsible for providing functions for managing service description, as well as allowing application to find services based on their capabilities and quality parameters.

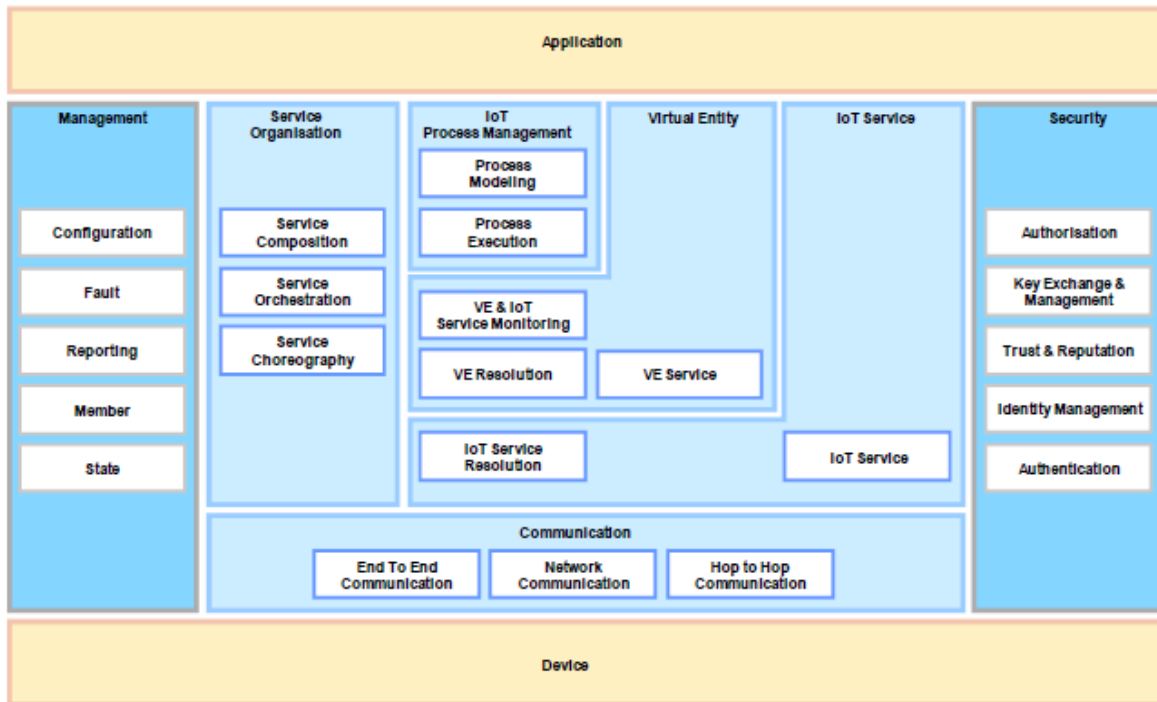


Figure 30. IoT Reference Functional View (IoT-A, 2013)

The communication layer is responsible for abstracting the communication technology. It should support routing, maintaining QoS, and queues for hop-to-hop communication such as mesh network build on 802.15.4. The network functions are responsible for the resolution of ID and network addresses, network protocol translation, routing, and QoS on the network level. The end-to-end functions are responsible for transmitting a message with the required protocol translations and pass this context between gateways.

4.1.3 Middleware for IoT

Based on the functional models, various types of IoT middleware have been proposed. Middlewares have been used in various application domains to facilitate communication between heterogeneous systems. For instance, in the industrial automation, OPC is used to bridge communication between the device on the shop floor with the supervisory control and data acquisition (SCADA) (Zheng & Nakagawa, 2002).

IoT middlewares proposed different IoT architectures. Some works followed a layered architecture to mimic the 7 OSI layer, for instance, the five-layer architecture depicted in the Figure 31 (D. Bandyopadhyay & Sen, 2011). It places the internet in the middle as the main communication media. The edge layer manages devices such as embedded systems, sensors, actuators, and ID tags. The access gateway layer manages the bridge for different communication technologies to the internet protocols. The main task of this layer is performing a routing optimization, bridging the different communication protocols to the internet protocols (e.g., TCP/IP), and forwarding data from the edge nodes to the other end

across the internet. The middleware layer provides generic interfaces for the applications to communicate with the IoT.

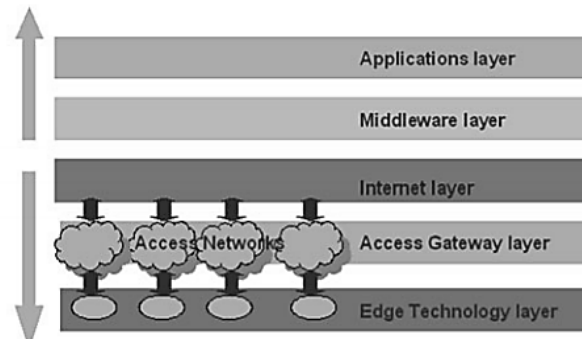


Figure 31. Generic Layered Architecture for IoT (D. Bandyopadhyay & Sen, 2011)

In addition, many approaches have been used for abstracting IoT devices as well as the programming model. Figure 32 shows the classification of approaches taken in designing a middleware used for wireless sensor networks (Hadim & Mohamed, 2006). It shows that the current WSN middleware abstract sensor networks, e.g., as a database which can be queried using SQL-like language or the application may communicate with the nodes through messages.

Interestingly, SOA middleware has been proposed quite often to support the integration of between “Things” and the legacy systems while providing interoperable Web Services for the applications to access (Jammes & Smit, 2005; de Souza et al., 2008; Markus Eisenhauer et al., 2009; Spiess et al., 2009).

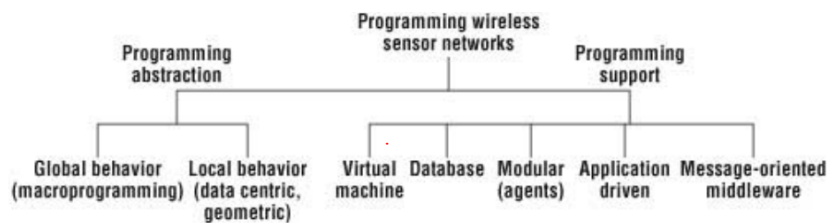


Figure 32. Classification of middleware approaches for wireless sensor network (Hadim & Mohamed, 2006)

SOA middleware for IoT exposes device capabilities as Web Services that can be accessed from any programming language that supports HTTP and XML. It introduces a service management layer whose task is to deal with service discovery, execution monitoring, and configuration. For the discovery purposes, a service registry such as Universal Description, Discovery and Integration (UDDI)³¹ is usually used. Supporting the applications, some of the SOA middlewares provide a Service-Composition. Service-Composition allows developers to create an executable workflow indicating the order and conditions to execute the services. The workflow could be expressed in Web Service composition languages such as Web Services

³¹ <http://uddi.xml.org/> (Retrieved on June 20, 2013)

Business Process Execution Language (WSBPEL)³² which then fed to a processing engine which executes the Web Service calls accordingly.

Although Web Services have been widely adopted within the enterprise environment, its reliance on XML format made it difficult to penetrate IoT scenarios, particularly when dealing with embedded system with limited computing resources. There exist more efficient data formats that can be used for embedded system communication such as JSON (Crockford, 2006) or binary XML (W. Lu et al., 2006).

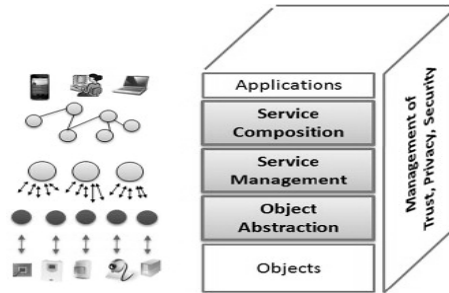


Figure 33. SOA-based architecture for the IoT middleware (Atzori et al., 2010)

4.1.3.1 LinkSmart Middleware

LinkSmart is a service-oriented middleware that facilitates IoT communications through Web Services. LinkSmart represents real-world objects by software proxies running on physical devices or gateways that are able to run an OSGi Framework³³. The proxies communicate with the physical devices through heterogeneous communication protocols. On the other end, the proxies provide Web Services that encapsulate these various communication protocols for the applications.

³² https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (Retrieved on June 20, 2014)

³³ <http://www.osgi.org/Technology/WhatIsOSGi> (Retrieved on June 20, 2014)

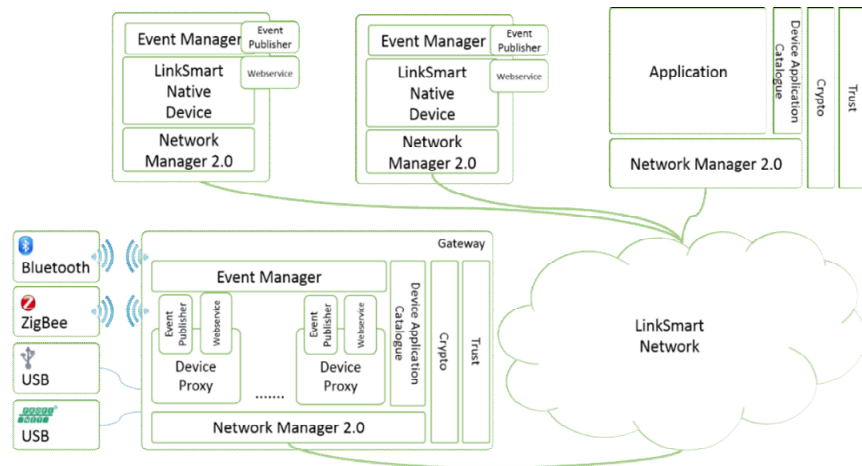


Figure 34. LinkSmart Architecture³⁴

Additionally, LinkSmart makes all Web Service calls to remote proxies appear as local Web-Service calls since the applications have to consume Web Services through a component called Network Manager with a local host address (Milagro et al., 2009). When the network manager receives a Web Service call from the applications, it is responsible for forwarding the calls through an overlay P2P network to the actual service providers as well as the responses from the remote services to the local applications as depicted in Figure 34. Moreover, the Network Manager can be used as a service registry where the proxies could register their services, and the applications could discover the ID of the services of the proxies. Every Network Manager keeps a list of services that have registered to it and exchange the list with other Network managers that are connected to the same P2P network. This makes the network manager to act as a decentralized service registry for the LinkSmart applications.

Enabling publish-subscribe communication pattern, LinkSmart provides an event broker that works based on Web Services. The event subscribers may subscribe to the events with particular topics. The subscriber must provide a Web Service callback method that conforms to the LinkSmart event subscriber interface. The event broker relays events from the event publishers to the corresponding event subscribers by calling the callback Web Service. This is useful for the input and sensor devices to notify the applications when new data are available.

LinkSmart does not offer a structured approach for IoT application development as it assumes that the application could simply consume the Web Service provided by the proxies. Alternatively, application developers could take advantage of the existing service orchestration approaches to orchestrate services offered by the proxies. However, the existing service orchestration frameworks such as Business Process Model and Notation (BPMN) (White, 2004) are designed to orchestrate high-level software services that have different characteristics than physical devices. For instance, many embedded devices operate in a more time critical environment. Moreover, sensor data usually contain measurement noise and

³⁴ https://linksmart.eu/redmine/projects/linksmart-opensource/wiki/LinkSmart_Architecture_2x (Retrieved on Oct 20, 2014)

without a proper context, they provide less meaningful data. Therefore, complex calculations sometimes must be done on the sensor data to provide a contextualized information, which is more meaningful than the raw sensor data.

4.1.4 Semantic Discovery

Being able to share IoT resources through the internet sometimes are required to reduce the cost of the systems. There exist already several sensor systems that are shared between different research organizations worldwide, e.g., Fluxnet³⁵, an international network of sensors distributed over 35 countries. Fluxnet is built to enable different meteorological applications, conducting micro-meteorological measurements. Each application may use a subset of the sensors and combine them with other sensor systems such as weather satellite. For addressing such a scenario, diverse distributed applications must be able to discover the shared devices, understand what they do, and know how to work with them. Being able to discover devices on the network level has been explored for multimedia devices and office appliances with technologies such as DLNA (J.-T. Kim et al., 2007), Bonjour³⁶, WS-Discovery (Modi & Kemp, 2009) however, these approaches were designed for local area networks and cannot be applied directly in the IoT scenarios since it covers broader heterogeneous networks and a range of devices. A conventional approach such as broadcasts and network advertisements without intelligent modifications would not be scalable for IoT. Understanding what the devices do, and how to work with them is essential in order to find the appropriate devices that are able to perform the necessary functions. These requirements have initiated the term semantic discovery which enables the applications understanding what the services do and how to access them.

In the context of Web Services, semantic technology has been investigated for finding services not only based on unique ids but also based on their “semantic” in terms of functionalities and capabilities (Mokhtar et al., 2008; R.-C. Wang et al., 2009). These services are published with their metadata that describe the capabilities of the services as well as other parameters that can be matched against the service requests. For instance, MIDAS (Middleware for Intelligent Discovery of context Aware Services) uses the service metadata and match them with the service requests send by the application to identify whether the service’s capability is semantically related the requested capability (Toninelli et al., 2005). Similarly, semantic discovery has been proposed for not only for discovering devices based on their physical qualities, but also to allow the applications to query for devices that fulfill the quality requirements, such as the accuracy and precision of the device (Kostelník et al., 2009; Compton et al., 2012).

Semantic description of devices has not been standardized and varied depending on the application requirements. For instance, when the quality and the trustworthiness of data obtained from sensors are of concern, the semantic description could contain the sensing quality such as accuracy, precision, drift, sensitivity, selectivity, measurement range, detection limit, response time, frequency and latency. Over the past several years there exist approaches how to model metadata of sensor systems that can be automatically processed by

³⁵ <http://fluxnet.ornl.gov/> (Retrieved on June 20, 2014)

³⁶ <http://www.apple.com/support/bonjour/> (Retrieved on June 20, 2014)

the users' application. Sensor Model Language (SensorML) (Botts & Robin, 2007) specifies XML schemas that can be used to create an XML model of observations by sensor systems. The schemas include an extensive set of concepts from the data type, observable events, aggregation processes, geographic locations, and sensor classifications.

Other approaches use ontology to model sensor systems since ontology offers some degree of expressiveness for describing relationships between entities in the application domain. Moreover, RDF based ontology provides a machine-readable information and allow developers to reuse the available tools to store, extract information, and perform reasoning. Semantic Sensor Network (SSN) Ontology is an example of the information model proposed by W3C Semantic Sensor Network Incubator Group (SSN XG). The SSN Ontology is designed to be used by independent application services, accessing the sensor systems as depicted in Figure 35. The ontology provides a schema that includes information such as the world phenomena that it measures, the physical devices and their functions, the measurement processes and observation (Compton et al., 2012). The SSN ontology should allow to be extended by relevant ontology such as measurement and unit ontology when required. The SSN XG provides an overview of the various ontologies for describing sensor systems (Lefort et al., 2011). This study evaluated 17 ontologies against several factors including:

- The initial purpose of the ontology
- Whether it is actively maintained
- Documentation
- The range of subject-matter
- Level of sophistication
- Adoption rate
- Best & worst features

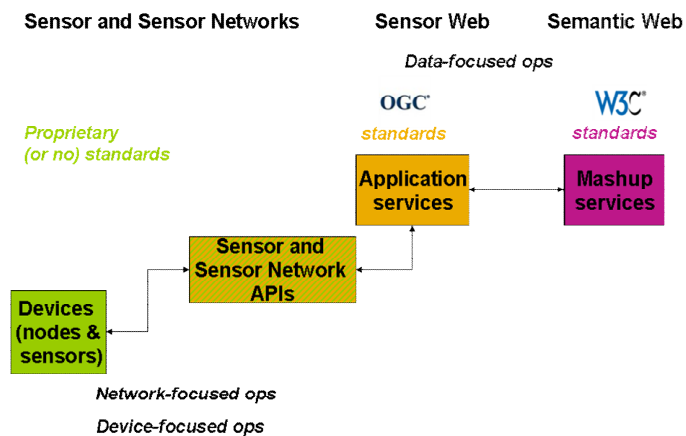


Figure 35. Relations between device, application, and Mashup services as envisioned by the W3C (Lefort et al., 2011)

Based on their study, the SSN ontology is built by extending the CSIRO ontology (Holger & Compton, 2009) since it is the only one that supports a proper composition with external ontology. The design of SSN ontology avoids providing a hierarchy of sensors which is mostly influenced by the application domain.

In the IoT context, it is quite challenging to enforce many parties to adopt one standard. Thus, this dissertation supports realistic scenarios where non-standardized vocabularies could be used to describe or search for devices by different developers and system integrators all over the world. With the existing approaches, identical devices described with different vocabularies cannot be discovered. Therefore, this dissertation argues that the semantic discovery should be able to understand various synonymous terms used to describe devices. This approach would increase the probability to find identical devices across the internet.

4.2 Model Driven Development

One of the challenges in software in general, including IoT development is filling the gap between the application domain and the technology used to implement the solutions of the problem (France & Rumpe, 2007). When creating a solution to the problem domain, developers are required to understand and map two different worlds. First is the problem domain space, which deals with concepts, information flows, behavior, and processes within the scope of the application domain. Then they must map their understanding of the problem domain to the computer science domain, which requires knowledge of programming languages, memory and storage management, as well as communication technology. Furthermore, these engineering challenges are only exacerbated by market's demands for more and more system functionalities that increase drastically from year to year. For instance, a recent survey done by Frost & Sullivan (Sullivan, 2013) reveals that Fighter Jets F-22 Raptor comprise about 1.7 million lines of software code (MLOC). The newer F-35, which was planned to have eight MLOC, has gone completely off the projection and ended up with approximately 24 MLOC. The software that runs the Boeing 787 is almost 7 MLOC (without entertainment system), which triples the Boeing 777 (Charette, 2009). The survey predicted that with the current rate of complexity growth, premium cars will require 200-300 MLOC in the near future overtaking the complexity of current fighter jets and passenger aircrafts.

This level of complexity motivates the industrialization of software development. Many discussions compared the software industry to mass production of goods (B. J. Cox, 1990; B. Cox, 1995). However, Greenfield contended these comparisons since the software industry moves at a very rapid pace and requires new features to be frequently implemented. Thus, the software industry requires creativity and strong analytical skills which cannot be solved with approaches used in mass productions (Greenfield & Short, 2003). Greenfield also argues that we need to distinguish *the economies of scale* from *the economies of scope*. The economies of scale try to reduce the cost of the product by mass replicating a single design while the economies of scope reduces cost as a result of reusing the same styles, patterns, processes.

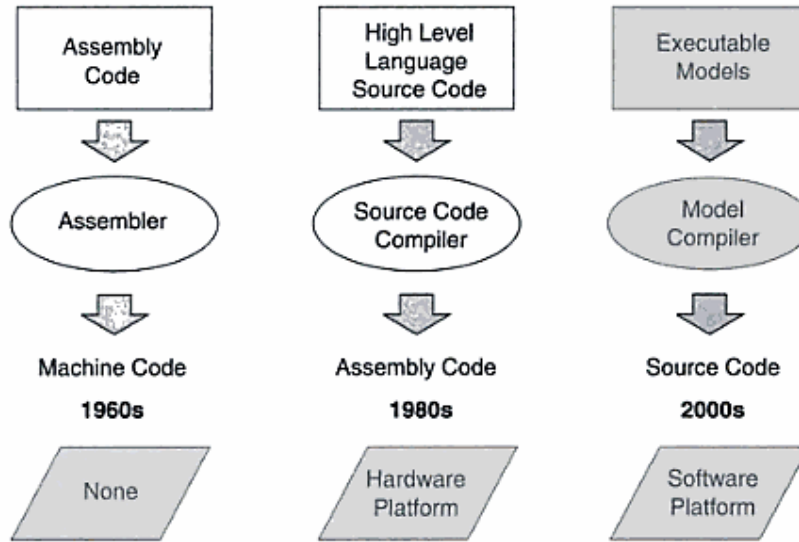


Figure 36. Elevating abstractions in software development (Mellor, 2004)

MDD offers an approach to industrialize software developments to a certain extent. MDD intends to bridge the gap between problem space and the technological implementation through systematic transformations of problem-domain level abstraction to more detail technology implementation (France & Rumpe, 2007). Models and modeling techniques have been used to reduce the risks of engineering in a different field of productions, e.g., electronic, machineries, and building constructions. On the other hand, models are often used only as a documentation tool, which offers a less value than what the MDD envisions. The MDD's vision could be compared to the evolution of the 1st and 2nd generation of programming languages that are closer to the hardware platform to the 3rd generation that was introduced to provide a higher abstraction level as illustrated in Figure 36. Nowadays, the 3rd-generation language compilers are able to transform high-level languages into machine executable codes very efficiently. Through several years of evolution, these compilers were optimized and now could even produce code that are more efficient than what a programmer could produce when he writes the machine code carelessly. In addition, this abstraction allows more complex software to be developed and managed than the software written using low-level programming languages.

The system model can be expressed in different levels of abstraction. For instance, assembly language was introduced to provide a higher abstraction over numerical machine codes, 3rd generation programming languages such as Java and C++ provide a higher abstraction over assembly, and modeling languages such as unified modeling language (UML) elevate the abstraction over the 3rd generation programming languages. Software models aim at reducing the system development complexity by separating the domain related design from the computing related design in different level of abstractions. Ideally, the computing related artifacts are automatically generated by MDD tools similar to what the compilers do with the programming languages.

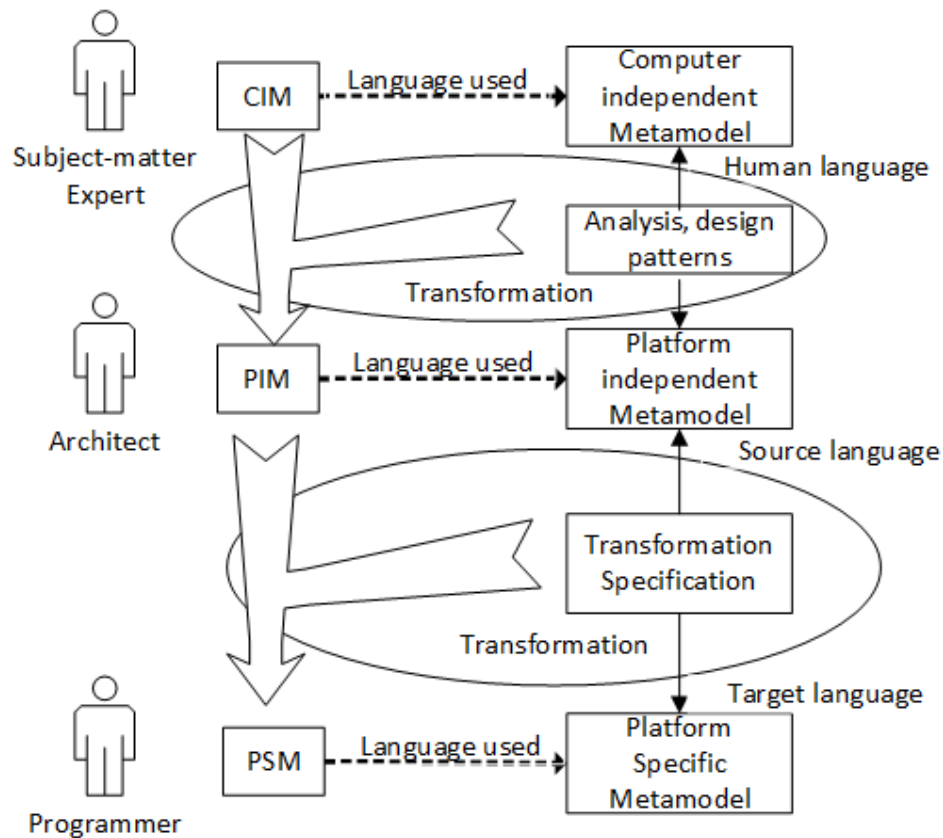


Figure 37. The transformation from the CIM, PIM and PSM [adapted from (M. OMG, 2003)].

There exist diverse forms of modeling languages such as graphical notations, hierarchical trees, and textual languages with a standardized vocabulary. The Object Management Group (OMG) has played a significant role in establishing standardized notations. For instance, the UML has gained popularity to define software models, and the system modeling language (SysML) has started to be used for modeling embedded system.

The OMG has specified a view for applying MDD in system developments. The framework was introduced with the name of Model Driven Architecture (MDA) (M. OMG, 2003). MDA recommends that systems should be defined in three levels of viewpoints, including the computation independent model (CIM), Platform-Independent Model (PIM), and platform-specific model (PSM). CIM reflects the system's structure, functions, and behavior in a language that is natural to the problem domain. CIM bridges the gap between the subject-matter experts who understand the problem domain and the software engineers who are experts in defining a technical design and an implementation. PIM extends CIM by including computational terms, but independent of any specific implementation technology. At this level, the problem domain is modeled using modeling languages such as UML, which is independent of any development platform. PSM describes the system concerning specific technological platform. At this level, the PSM contains detailed system instructions, e.g., in a specific programming language or database management system. MDA still considers the

available programming languages to be platform specific regardless whether they can be executed on different hardware or not such as Java. MDA envisions that PIM would provide a generic solution that will survive rapid technological evolutions. In addition, to increase productivity PSM may be generated from PIM based on transformation instructions. MDA is built upon standards established by OMG including the UML³⁷ that can be used to define PIM, the Meta-Object Facility (MOF)³⁸, which can be done to define metadata for a model, the XML Metadata Interchange (XMI)³⁹ to enable model exchanged between modeling tools, and the Common Warehouse Metamodel (CWM)⁴⁰.

4.2.1 Embedded System Software Modeling

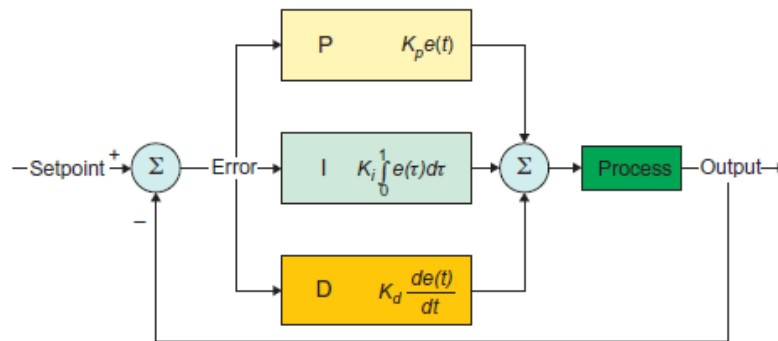


Figure 38. PID control algorithm represented in a graphical software view (Gretlein, 2013).

System models are widely used for embedded system developments. MDD approach is well supported by the availability of system modeling tools that also provide an offline simulation and validation, allowing developers to find design and logic flaws throughout the development cycle. In addition, developers' mistakes and overheads could be reduced through automatic generation of code (Gretlein, 2013). System modeling has become a natural way to define an embedded system behavior. For instance, algorithms to be implemented on proportional-integral-derivative controller (PID controller) (Willis, 1999) is usually represented in a graphical software view as illustrated in Figure 38.

In system developments for factory automation, PLC are used to control sensors and actuators in electromechanical processes that are part of a subsystem. Unlike multipurpose personal computers that are designed to be programmed by computer scientist, PLCs are designed to be programmed by electrical engineers who are used to work with electrical wiring diagrams. For making software development feels natural for electrical engineers, several domain-specific languages have been proposed since the 70s as Figure 39 shows. Between these languages, the most influential standards are IEC 61131 and IEC 61499 which nowadays are widely used by automation and electrical engineers (Frey & Litz, 2000).

³⁷ <http://www.omg.org/spec/UML/2.4.1/>(Retrieved on June 28, 2014)

³⁸ <http://www.omg.org/mof/>(Retrieved on June 28, 2014)

³⁹ <http://www.omg.org/spec/XMI/2.4.2/>(Retrieved on June 28, 2014)

⁴⁰ <http://www.omg.org/spec/CWM/>(Retrieved on June 28, 2014)

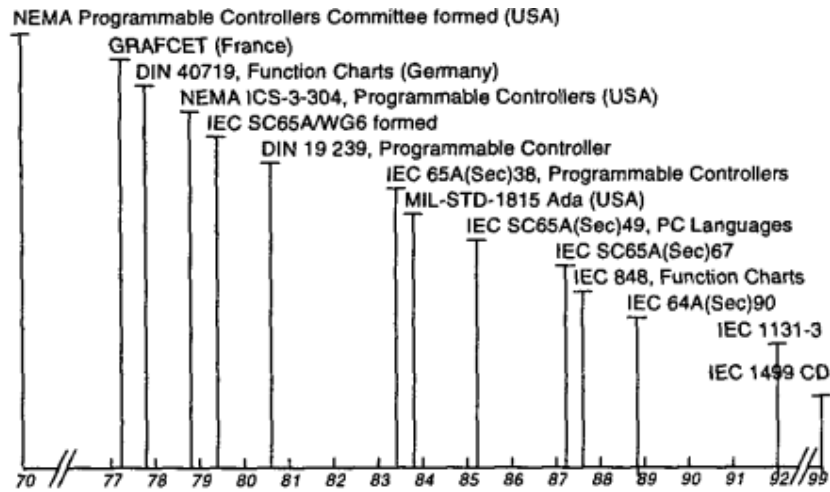


Figure 39. The evolution of programming languages for PLCs (Frey & Litz, 2000)

The IEC 61131 part 3 describes Ladder diagram and Functional block diagram, which are graphical programming languages. Moreover, it contains two textual languages, including Structured Text and Instruction List (John & Tiegelkamp, 2010). Ladder logic enables electrical engineers to design the application logic as if they design electrical wiring on circuit boards. When the software is defined and uploaded to a PLC, it runs in a loop and scans the application logic from the top left down to the bottom right. The language is very logic driven and very suitable for modeling discrete signals. On the other hand, PLCs handle analog signals as an integer counter. To provide an abstraction to a complex application logic, it relies on Function Block concept. The blocks contain isolated application logics that can be connected to other blocks creating more complex functions. The output of a function block can be connected to the input of another function block. The function block may perform functions such as transforming the input data into the required information or triggering appropriate actuations.

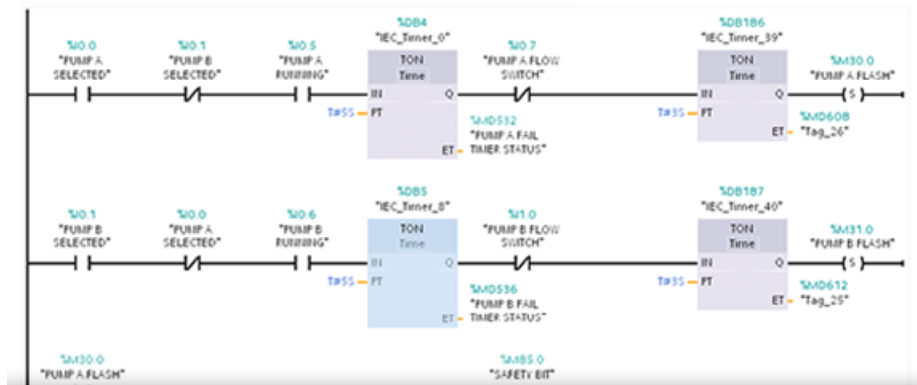


Figure 40. An example of ladder logic used for programming Siemens PLCs

MDD tools that could provide a simulation environment is very beneficial and cost efficient for performing testing and validations of mission and safety-critical systems particularly when they involve mechanical parts. This reduces complexity of the development as well as to

ensure that software defects could be found in the simulation rapidly to prevent damages to the mechanical parts as well as hazards to the human who interact with the system. Automotive and aviation control systems are examples of safety-critical systems, which are often built by performing modeling and simulation extensively during the tests, e.g., testing the control unit of an engine or break components.

4.2.2 MDD drawbacks

Software and system models only able to express the system partially since they hide the complexity of the implementation details. While this is an advantage for defining systems whose characteristics are definable by the available abstraction, it becomes a drawback when the required details cannot be expressed with the abstraction provided. In addition, there is always a tradeoff between raising the level of abstraction and over simplifying the solution to the level that it is not useful anymore to solve the problem (Hailpern & Tarr, 2006). This can be seen, for instance, when the architecture of the system is defined in a very abstract way, it does not help the developers to understand how the system should be implemented.

When models on different abstraction levels are used for documenting software code, they create multiple representations of the system. Several literatures have pointed out that keeping consistencies between software models are indeed problematic (Hailpern & Tarr, 2006). Often it is hard to maintain these representations in sync, particularly when they are done manually without a proper management tools. This creates a problem that the models in different abstraction levels must be correlated and traceable. When changes are necessary to be done at any level of the abstraction, these changes must be propagated to the other level and consistency between these models must be managed carefully. In addition, most of the roundtrip engineering provided by current tools do not seem to be able to provide an optimal solution. This problem has been faced by the MDD community since the first time. Synchronizing two languages at different abstraction levels could not be done easily since some details are lost when a more detailed language is transformed into the more abstract language.

Moreover, using code generators to generate programming language from abstract models may present a classic problem caused by automated systems. Automation might cause the experts to lose their expertise because they rely on the automation and thus over the time they become out of practice. Consequently, they may not have a sufficient knowledge or confidence to manipulate the code manually for obtaining the results which are not foreseen by the code generator. The lack of confidence and expertise may also influence their ability to fix problems when they arise.

4.2.3 Mashup development

A similar approach to MDD is Mashup development. It is initially used as a way of building web applications rapidly by aggregating different data sources on the web by using a graphical development interface (Grammel & Storey, 2010). Yahoo! Pipes⁴¹ and DERI Pipes⁴²

⁴¹ <https://pipes.yahoo.com/pipes/> (Retrieved on Oct 15, 2014)

⁴² <http://pipes.deri.org/> (Retrieved on Oct 15, 2014)

are examples of Mashup development platforms that allow non-expert developers to compose services by linking the input, output, transformation components.

Since the abstraction level of Mashup components is usually quite high and reveals less technical details, it is less flexible than conventional programming languages (Grammel & Storey, 2008). However, it is done so intentionally to ensure the simplicity of the application development. Because of this reason, Mashup development is generally aimed at the end users with minimal development experiences instead of expert developers. To illustrate how a Mashup development is done, Figure 41 shows a Yahoo! Pipe development environment which enables users to connect modules retrieving data from diverse sources (e.g., RSS, Web Service, user input) then transform the data into a desired information.

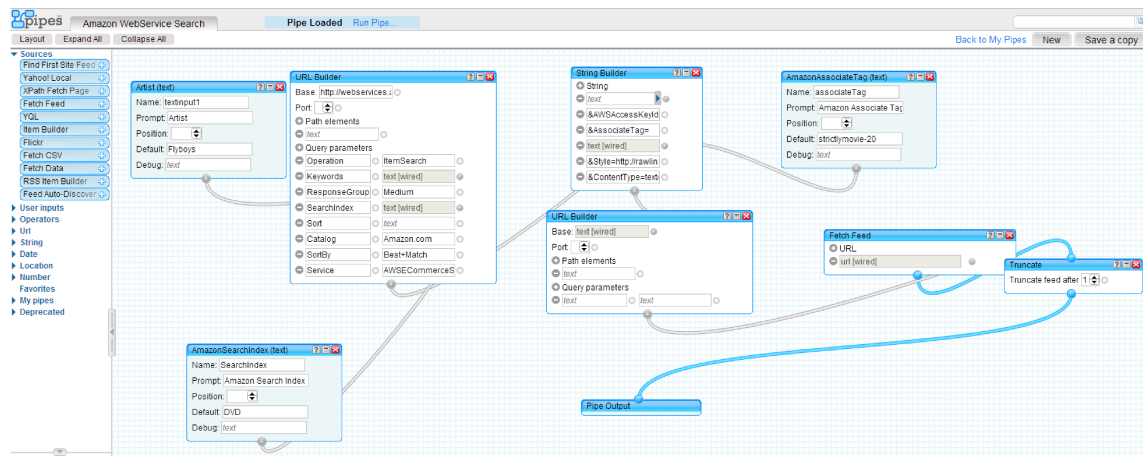


Figure 41. An example of a Yahoo! Pipes service for querying the Amazon Web Service.

The output of each module can be used as input to another module given that they have a compatible data type otherwise data transformation components must be introduced. A study that evaluated Yahoo! Pipes' acceptance showed a good acceptance and fast learning curve (Yue, 2010). In the enterprise application development, Mashup development has been investigated to involve business users, who do not have extensive programming experience, to create and share their applications. This approach is proposed to reduce a bottleneck on the IT department which has to implement different business requirements (Cherbakov et al., 2007; Maximilien et al., 2008).

There exists a similar approach as Mashup development, but focuses for IoT application called Node-RED⁴³. It is an open source project created by IBM that can be deployed on a local computer as well as small computing platform such as Raspberry PI (www.raspberrypi.org). Their approach relies purely on data flow abstraction similar to flow based programming (Morrison, 2010).

4.2.4 Web of Things

In the IoT field, Mashup development was investigated to simplify the development of "Web of Things" (WoT). The WoT approach believes that interconnectivity between Things should

⁴³ www.nodered.org (Retrieved on August 15, 2014)

be facilitated by Web 2.0 technologies to achieve a common standard and breaking the barrier presented by the current “silos” of networks (Guinard et al., 2009).

Guinard et al. propose using RESTful services to integrate physical objects by abstracting them as HTTP resources (Guinard & Trifa, 2009). The physical devices that are represented by RESTful resources can be combined together with other virtual resources in a Mashup application. A prototype of Mashup development environment is proposed to wire the available components which then executed as a service on the server (Guinard et al., 2011). Although they acknowledge that the discovery of devices is required in IoT, they have not yet presented a concrete proposal to solve the problem. Stirbu proposes a similar approach of using a RESTful service to represent diverse physical sensor and actuator networks that he called SAN islands (Stirbu, 2008). He elaborated the discovery of things by utilizing a Resource Repository, which maintains context information of the sensors and actuators such as their capabilities. The Resource Repository can be accessed through an Atom feed (Nottingham & Sayre, 2005).

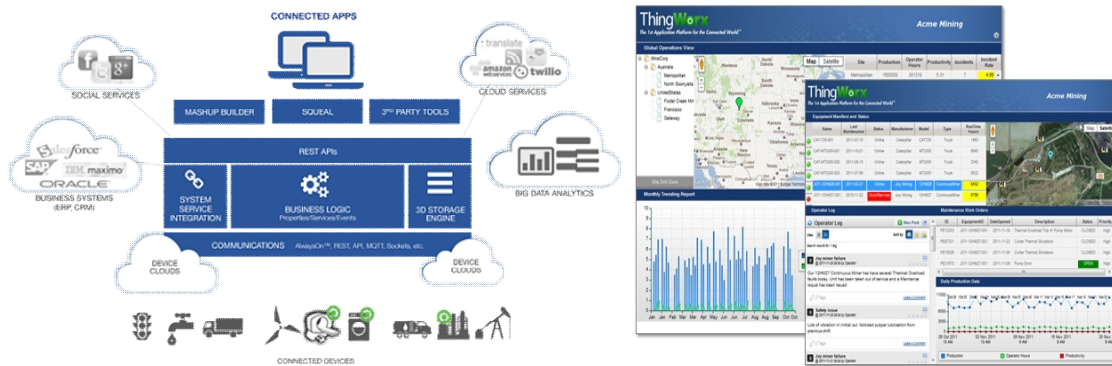


Figure 42. ThingWorx, a centralized platform for analyzing data from IoT⁴⁴

Another work presents a unique possibility to share Things through social network platforms such as Facebook and Twitter (Guinard, Fischer, et al., 2010). They describe an approach that enables physical objects sharing data by posting messages to their profile or newsfeeds. Additionally, their approach allows the users to manage filters to the data to be published and an access control to the data, based on the social structure.

Commercially, there exist web or cloud oriented platforms for aggregating sensor streams such as Xively⁴⁵, Open Sense⁴⁶, and ThingWorx⁴⁷. These platforms allow developers to connect different sensor streams to their cloud based data storage and aim at providing programmable data analytics. Furthermore, they enable developers to create business intelligence-like dashboard combining different information, e.g., GPS location and a map or pollution level in different parts of the city. These platforms also allow developers to share their data to other parties that might be interested. Some of these providers provide Mashup development tool allowing developers to combine sensor streams into data analytics or

⁴⁴ <http://www.thingworx.com/platform/#how-it-works> (Retrieved on June 25, 2014)

⁴⁵ <https://xively.com/> (Retrieved on June 25, 2014)

⁴⁶ <http://open.sen.se/> (Retrieved on June 25, 2014)

⁴⁷ <http://www.thingworx.com/platform/> (Retrieved on June 25, 2014)

visualization components rapidly. However, only a few provides support for integrating heterogeneous devices into the platform, e.g.; Xively provides libraries in several languages that can be used to connect devices into the platform. Examples of other products that provides an open source and nonprofit platform for sharing and visualization of the data produced by devices and services are Fluxstream⁴⁸ and ThinkSpeak⁴⁹.

4.3 Summary and Conclusion

IoT architecture initially only presents a way to include physical objects into IoT with tagging technology. However, this has progressed covering a wider definition of IoT including the wireless sensor and actuator network, as well as smart devices. Many works have proposed high-level architectures comprise diverse elements required for IoT. However, these proposals were too diverse and have failed to provide a common ground and guide for the future IoT developments. IoT-A was an effort by many organizations and the European commission to create a uniform understanding of IoT architectures. They have studied the previous works and concluded the IoT architecture as the ARM. IoT-A provides an IoT domain model that describes how the physical objects, virtual entities, and technological components are related which is useful to provide a uniform understanding and standard terminology. Moreover, it describes the links between physical devices, different communication patterns, as well as device abstractions that can be done on the IoT service level and virtual entity level.

However, it does not guarantee any standardization on the implementation level, which is reasonable considering that IoT-A aims at providing an architecture reference that applies for different application domains with a different set of requirements and technology dominance. The ARM must be extended in different application domains to provide a standard approach with a sufficient level of details that are useful for application development in that domain. Moreover, support from big market players are required to popularize the architecture as well as its adoption.

In the area of development platform, different approaches have been proposed to provide an abstraction for the IoT. In the industrial and business area, service oriented architecture has gained significant interests since it has been adopted by many enterprise systems. However, SOAP based Web Services require too much overhead for transporting sensor data, particularly by resource-constrained devices. Therefore, gateway and software proxies are used to provide translation services between the lightweight communication protocols into Web Services. In addition, technology such as RESTful services and CoAP have gained significant interest from the Web of Things community, since they are simple to use and require less computing power and bandwidth that is suitable for resource-constrained devices. Publish-subscribe communication pattern is also required in IoT to ensure the sensor values reach the destination instantly and to reduce the traffics that would have been caused by relying solely on polling technique. MQTT shows a promising approach for enabling communication between small devices since it is simple to use and imposes a very little communication overhead.

⁴⁸ <https://fluxstream.org/> (Retrieved on June 25, 2014)

⁴⁹ <https://thingspeak.com/>

The current middleware approaches often offer comprehensive features such as device abstraction, discovery, addressing, and security. So far, there has not been any approach that is accepted as the standard middleware for IoT. The author believes that this fragmentation will still exist in the future since it is not possible to provide a solution for all IoT problems in different domains. Many of current middleware approaches target professional developers with extensive experiences in electronics, communication technologies, embedded system, and data processing. For instance, in the industrial automation ladder logic is used to program industrial controller since most automation engineers have an electrical engineering background.

Studying the software developments in embedded systems reveals that the level of software complexity increases dramatically from time to time, which makes complex software system harder to develop and maintain. This problem also affects IoT developments. The rapid increase of middleware's features which results in a higher complexity, a steeper learning curve, and more prone to human errors particularly for inexperienced developers. Consequently, there is a new trend of simplifying the development platform, particularly for less experienced developers and end users.

MDD tries to solve this problem by raising the level of abstraction that the developers work to reduce the complexity of the software artifacts and the developers' efforts (Hailpern & Tarr, 2006). Another promising approach that often used for end-user development is Mashup developments. Both mashup and MDD usually rely on visual modeling languages and aim at increasing the level of abstraction which the developers work with in order to reduce the developers' effort and the complexity of the software artifacts (Hailpern & Tarr, 2006). Mashup development usually based on flow-based programming (FBP) (Morrison, 2010) while model driven focuses on achieving software model to represent the problem domain. While model-driven approach usually generates 3rd-generation programming languages from the high-level models, Mashup development environment provides an executable model. An advantage of generating 3rd-generation language is the possibility to extend the results when the modeling language does not provide sufficient details to define the system behavior. However, working in different abstraction level may cause consistency issue which is one of the major problems in MDD. For instance, Node-RED is an IoT development tool that adopts Mashup development. It relies on the flow based modeling to link different components that are required by IoT applications. However, Node-RED does not have any separation of concern between the components. It considers that all components are at the same level and can be connected to all other components. Without a clear separation, the diagram could be overwhelming and hard to understand for inexperienced developers.

As the number of connected things increases rapidly, the author believes that in the future, inexperienced developers and end users will contribute significantly to IoT developments and therefore must be supported by appropriate tools that are able to encapsulate the complexity of IoT technology. Moreover, the tools must be quite simple and easy to understand as well as conforming to IoT-A reference architecture model as an effort in standardizing IoT architecture. The author argues that applying the MDD approach and Mashup development could help the inexperienced developers building IoT solutions faster compared to development tools that rely on textual programming languages.

Filling the gap this work proposes a simplified five-layer architecture for IoT systems that enables rapid IoT developments using a model-driven approach. In addition, an MDD tool called IoTLink is designed and implemented based on the proposed architecture. The design and implementation of IoTLink and the architecture are elaborated in Chapter 5.

Chapter 5.

Design Concept and Technical Implementation of IoTLink

Based on the findings discussed in Chapter 4, the author believes that enabling rapid IoT prototyping for inexperienced developers could be achieved when the developers are equipped with a development tool that is able to encapsulate heterogeneous physical devices and create the representations of physical things visually by combining the advantages of Mashup development and MDD. Hence, this work proposes IoTLink, a development tool that is based on the MDD approach. MDD aims at bringing system developments close to the application domain by increasing the level of abstraction and reduce the complexity of the software artifacts (Hailpern & Tarr, 2006). Additionally, MDD decouples the required domain knowledge and specific IoT technology needed to implement the solution. Decoupling these allows the knowledge about the domain to be engineered by domain experts while the technology experts focus on addressing the technological implementation (Pramudianto, Rusmita, et al., 2013).

Designing IoTLink, this work follows a user-centered design (UCD) methodology which is recommended by the ISO 9241-210 for designing and developing an interactive system (ISO, 2009). UCD is a design approach that grounds all design decisions based on the people using the product. UCD was chosen since the focus of the development tool is to support inexperienced developers, and therefore the usability and the acceptance of IoTLink is the main focus of this work. Applying UCD in an iterative development reduces the risks that IoTLink would not have addressed the developers' requirements since it was refined gradually based on the users' feedback. Additionally, since the users are involved in the design phase, they feel that IoTLink is tailored to their needs and therefore increases the chance for IoTLink of being accepted.

ISO 9241-210 describes six principles for conducting user-centered design including:

1. The design is based upon an explicit understanding of users, tasks and environments.
2. Users are involved throughout design and development.
3. The design is driven and refined by user-centered evaluation.
4. The process is iterative.
5. The design addresses the whole user experience.
6. The design team includes multidisciplinary skills and perspectives.

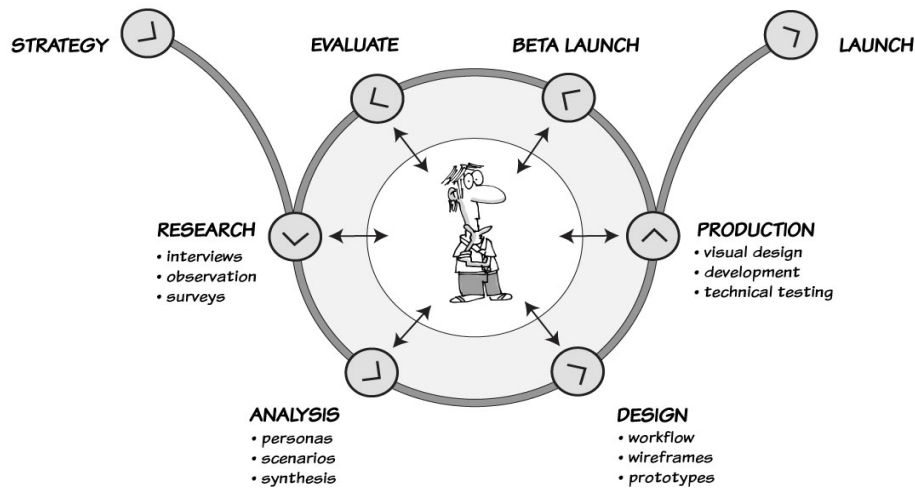


Figure 43. User-centered design process⁵⁰

As depicted in Figure 43, UCD applies an iterative approach in which the users of the system are involved in every step of the process by evaluating the design as early as possible. Initially, the user requirements are collected by different techniques such as interviews, survey, and workshop. Scenario and personas that describe the desired system were generated and validated by the users. Then in the initial phase of the design, low fidelity prototypes are used, for instance, by using paper prototypes or wireframes which save time and efforts while the degree of uncertainty is still high. When the low fidelity prototypes are matured, high fidelity prototype is generated and validated by the users. These steps are done to every part of the system and repeated until the whole functionalities of the system is implemented.

5.1 Requirement elicitation

The author organized four workshops with software developers, automation engineers, electrical engineers, and project managers to collect information about the problems they have faced within IoT projects. These workshops were done in conjunction of two master theses works under the author supervision, and the other two workshops were done during the requirement elicitation of BEMOCOFRA and IMPReSS⁵¹ project.

The first two workshops were attended by 8 and 11 participants, including research associates and students working at Fraunhofer FIT. The participants have various experiences in developing IoT applications ranging from none to expert. The main purpose of the workshop is to explore the problems of developing IoT prototypes particularly for inexperienced developers. Additionally, in the workshop, several questions such as the tools they used to develop IoT applications, what would help them developing IoT prototype rapidly was discussed.

⁵⁰ <http://uxmastery.com/resources/process/>(Retrieved on July 1, 2014)

⁵¹ <http://www.impressproject.eu/news.php> (Retrieved on August 13, 2014)

The other two workshops were attended by 7 and 8 project partners, including software engineers, automation and electrical engineers, and project managers. In the second workshop, the requirements are collected, and the initial idea of providing an MDD tool was discussed.

5.1.1 Scenario and Personas

A scenario was developed since the first workshop and extended throughout the next three workshops. Moreover, the personas were created to represent different actors to be addressed by IoTLink. The scenario that was defined to help defining the requirements is depicted in.

Table 1. Scenario defined to summarize typical problems faced by the actors

As the utility costs increase every year all over the world, Acme Corp that has large buildings all over the world decides to optimize the energy consumption gradually. Before deciding on the system development, various approaches should be investigated by the R&D team then the best possible approach should be advised to the local branches. To keep the cost of the system on the local branch low, local components should be used. In the latter phase, these buildings should be connected to a central system, which could analyze the energy data, create forecasts, and recommend possible optimization.

As an initial step, the prototypes should be developed and evaluated in their R&D branch in Germany that has quite significant resources of software and electrical engineers. They should come up with different system prototypes to monitor energy consumptions using various combinations of sensors and actuators. Unfortunately, the senior engineers in this branch are already assigned to higher priority projects. Mr. Jones, who leads the project only could get his hands on an electrical engineer, Tom and Jerry, a junior software engineer. Both do not have extensive experiences in integrating heterogeneous sensors and actuators technologies. Fortunately, the senior software engineers have experimented with a development toolkit, named IoTLink, that is meant to bridge the gap between software engineers with computer science background and automation engineers with industrial automation and electrical background. IoTLink is intended to enable cooperation between these two groups of engineers to integrate heterogeneous sensors and actuators rapidly. IoTLink exploits a visual language which automation and electrical engineers are accustomed to. From the graphical model, a Java code can be generated and extended by the software engineers who have the more experience in Java programming languages.

Kerry, one of the senior software engineer that developed IoTLink has agreed to guide the junior engineers in this project. However, Kerry only has a limited time since he is involved in three other projects at the moment. Kerry sends an instruction to Tom and Jerry one of the junior software developer assigned to the project received an instruction from Kerry to download IoTLink and tried it out. They open IoTLink and select a new prototype project. They read the one-page instruction very briefly, which describes the main idea, and the instruction how to use IoTLink. They could easily understand that they need to create a domain model and defined the objects to represent physical objects such as the rooms and appliances using a visual language. Then they need to connect the necessary sensors and actuators that can be used to sense the state of the rooms and appliances.

To measure the energy consumptions of the appliances, Tom who has experience in power

metering advises Jerry to use Plugwise which is a wireless smart-plug based on ZigBee technology. After the new project is opened, Jerry and Tom see a list of supported data sources and devices that he can use to sense the properties of the objects.

They create a domain model then defined the objects for representing the rooms and appliances such as printers and coffee brewer in the room. They connect these appliances to Plugwise components visually to measure the power consumptions and to be able to switch them remotely. They then choose a database connection for recording the power consumptions. Jerry adds a REST interface that enables him to retrieve this information easily through a mobile interface that he needs to develop.

As Tom and Jerry want to compare the energy consumption of two rooms, which are located in a different climate condition, he informs their colleague in Singapore branch to connect several devices in his office to any power sensor available locally and make them discoverable through the internet. His colleague connects them to a power strip, which has sensors to measure the energy consumptions. He developed a proxy and described them with his own terms. As soon as the devices in Singapore are connected, Tom and Jerry could easily discover them through the Discovery Manager by searching “smart plugs”. Tom and Jerry include these devices in his model and generate the prototype Java code. Jerry then develops a mobile application that accesses the REST interface generated by IoTLink to monitor the devices in these two different location. The power consumptions are also recorded in the database to be analyzed later.

Tom and Jerry show the visual model to Mr. Jones and explain how the devices are connected, and how the physical objects are monitored. Tom, Jerry and Mr. Jones feel for the first time they could work together using the same language without having problem understanding what the other was doing.

Kerry was able to check on the results and very satisfied since the solutions produced by Tom and Jerry conform to the IoT reference architecture model, and the quality of the code is quite good since they are consistent and well documented as a result of automatic code generation.

The personas were defined based on the typical actors’ background and problems on a template that is partially illustrated in Figure 44. There were several personas that the author identified. First, a project manager, Mr. Jones, who has a little to none programming experiences. But he needs to interact with the developers, monitor the progress, and understand the solutions that are proposed by the developers on a higher abstraction level. His constraint is often related to the limited time and workforce that he could get to achieve the intended results. He often gets several inexperienced developers and an experienced developer who do not work full time on the project.

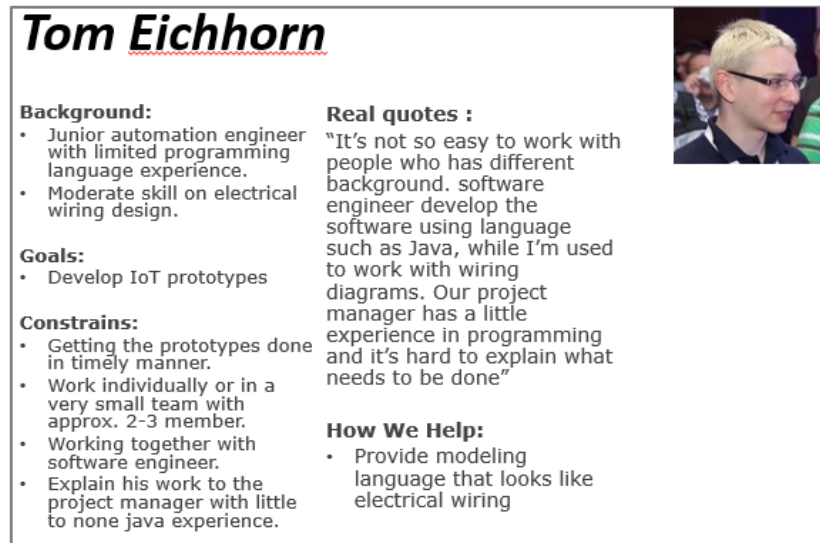


Figure 44. Example of a persona card used during the design of IoTLink

Another persona that the author identified was Tom, an automation engineer who has more experience in electrical wiring. His main constraint was that he often has limited time to develop prototypes. Additionally, he has to work with software engineers who sometimes explain their solutions by showing Java programming language which he has only a little experience with. Sometimes he has to work with devices remotely, and it was difficult for him to talk to these devices which have different protocols and capabilities.

The third persona is Jerry, a junior software developer who has experience in programming languages, but do not have extensive experiences working with sensor devices, network communications, and electrical wiring. His main constraint is to be able to work with electrical engineers integrating heterogeneous devices into his applications, being able to explain his solutions to his project manager.

The fourth persona is Kerry, a senior software engineer who has extensive knowledge on programming languages related to device and application developments. His main constraint was that he was often assigned to several projects at the same time. Therefore, he needs to delegate the detail programming tasks to his colleagues who have much less experience than him. He also needs to pay attention that the software architecture design conforms to the existing standards and internal convention in order to make sure that their solutions are manageable and can be extended easily in the future.

After analyzing the scenario and personas, several high-level requirements can be extracted which are presented in Table 2.

Table 2. The requirements of the actors involved in the aimed scenario.

Actor	User Requirements
Project Manager	<ul style="list-style-type: none"> • Able to see the solution on a high-level abstraction. • Produce prototypes with small and inexperienced team in a

	timely manner.
Senior Software Engineer	<ul style="list-style-type: none"> • Delegate programming tasks to the junior engineers. • Making sure the solutions conform to the available standard. • Making sure that the code quality is consistent and well commented. • Use less time to guide his junior engineers. • Provide reusable components for inexperienced developers.
Junior Automation Engineer	<ul style="list-style-type: none"> • Define solutions using language which is natural for him as an electrician. • Being able to work together with the software engineer on the same abstraction level. • Ability to find devices shared on the internet. • Ability to communicate with local and remote devices. • Ability to process and fuse sensor data.
Junior Software Engineer	<ul style="list-style-type: none"> • Able to integrate heterogeneous devices without having an extensive knowledge of the communication network. • Define a representation of physical objects based on object-oriented principles. • Able to automate some of the programming tasks and generate the necessary codes. • Able to extend the code using Java. • Able to share a level of abstraction with the electrical engineers and project manager.

5.1.2 IoTLink Conceptual Design

Considering the requirements and scenarios, IoTLink must be designed not only to enable rapid, but also to accommodate cooperation between users with different background, including software and electrical engineers, as well as project managers with little to extensive knowledge of programming language or communication technology. There exist many approaches in providing an abstraction of physical objects. Nonetheless, as discussed in section 4.1, the most comprehensive efforts to standardize IoT architecture is done by IoT-A through their ARM (IoT-A, 2013). Thus, the conceptual design of IoTLink is inspired by the ARM domain model which has been simplified as depicted in Figure 45.

The domain model describes that the IoT comprises physical objects, which are the concern of the application domain (e.g., room, window). Each physical object may have physical qualities (e.g.; a room has temperature, and a window has open or close state). These physical qualities can be measured by sensors (e.g., the temperature of the room could be measured by several thermometers and the state of the window could be measured by contact sensors). Each physical object within the application domain must be uniquely identifiable and therefore must have a unique ID. Additionally, physical objects may offer services that can be used to access information about themselves or perform actions that influence the environment including their own physical qualities. Some of these actions require actuators (e.g., a motor to close the window automatically).

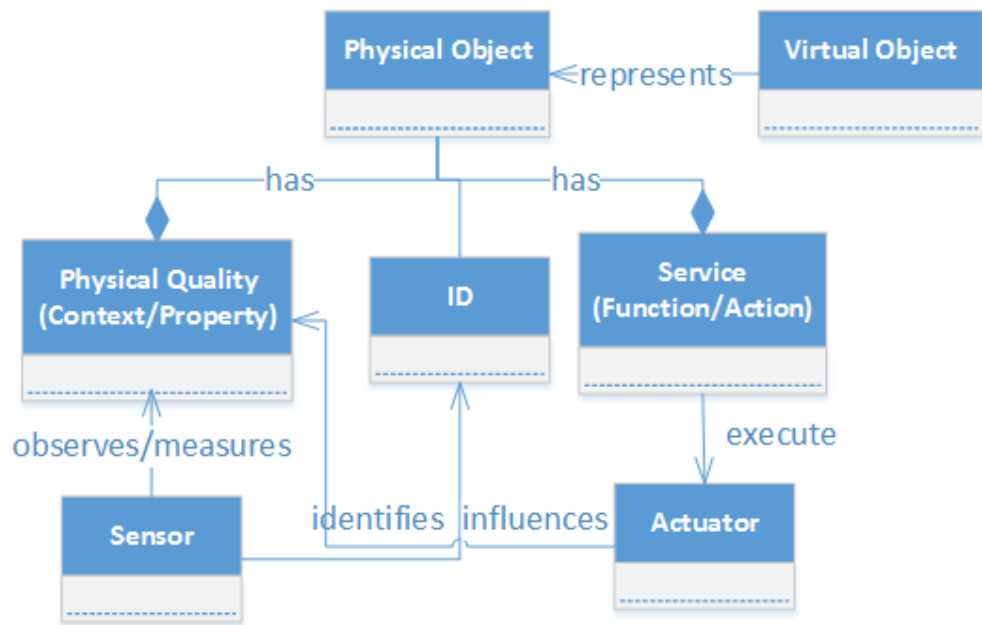


Figure 45. Internet of Things Metamodel simplified from (IoT-A, 2013).

In this context, devices that are sensors should be excluded from the domain objects since they only have a supporting role in determining the physical qualities or properties of the physical objects which are the concern of the problem domain. Similarly, actuators could also be considered as supporting devices that allow domain objects to affect the state of the environment.

In reality, the relationship between sensors and physical objects is not always straightforward. Sensor readings do not always represent the actual state of the physical objects being observed. This is caused by the physical and technological limitations, which produce noise in the observation data. In this case, the sensor data sometimes must be processed first to compensate the measurement noises.

In addition, sensor limitations may also produce a partial view of the physical events. In this case, several sensors or different types of sensors are required to infer the whole physical events, being observed. For instance, to measure emotion of a user, several bio-readings such as respiration rate, heart rates, skin conductance may be collected and through intelligent

algorithms, the system could conclude the stress level (Haag et al., 2004). Thus, within IoTLink, the author must introduce a mechanism which can be used to pre-process and fuse sensor data until the final results can represent the actual state of the physical objects being observed.

The physical objects foreseen in the IoT-A model could include non-electronic objects and electronic devices that may or may not have ability to communicate with other physical objects or the objects within the virtual world. This could be caused by the physical objects not having any communication component or incompatible communication protocols. Therefore, to enable communication between these physical and virtual objects, software proxies that are able to represent these objects must be employed.

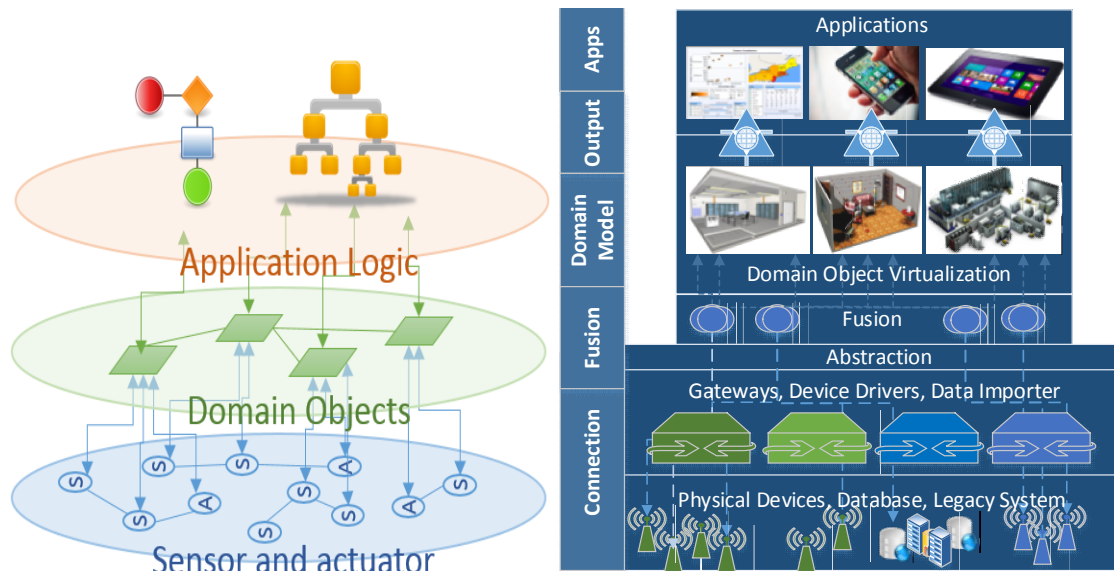


Figure 46. Abstraction levels of IoT, which hides sensors and actuators within domain objects mapped to the IoT layered architecture (Pramudianto, Rusmita, et al., 2013).

As shown in Figure 46 on the left side, abstracting sensors and actuators through domain objects encapsulates the complexity of individual device. This approach allows the application logic to be defined using a high-level abstraction that only deals with the concern of the problem domain. Consequently, technology changes could be decoupled with the application logic and vice versa. Based on this conception, the author designed a logical layered architecture as depicted in Figure 46.

The architecture anticipates a separation of work between domain modeling and technological implementation such as implementing connections to heterogeneous sensors and actuators. This separation allows domain experts to use their knowledge for designing the domain model as well as the concrete objects as they perceive the problem domain. On the other hand, the technology experts could focus on technological implementations on the other layer such as providing heterogeneous connection components, output components, and sensor fusion algorithms.

5.1.2.1 Connection Layer

The first layer at the bottom is responsible for establishing connections to heterogeneous data sources. It represents a collection of data sources, including physical devices and other subsystems that communicate through gateways or software proxies. This layer is responsible for providing a uniform interface and data format for accessing the data by the components on other layers.

When accessing heterogeneous data sources, diverse communication techniques and data format must be considered. As IoT-A describes, there are two types of communication (section 4.1.1). First, the passive data sources only provide software interfaces that must be pulled by any interested component. Second, the active data sources that are able to raise events and push data to the interested parties. To address these two communication approaches, IoTLink must allow users to subscribe to events and pull data repetitively.

Communication protocols and data format have more variations that must be generalized in order to make heterogeneous technology transparent for the components on the other layer. As a solution, IoTLink abstract them as connection objects that must be implemented with specific communication protocols, including IoT standard protocols such as Web Services, MQTT and device specific protocols. Moreover, for handling diverse data format, IoTLink abstract them as sensor observation objects that can be accessed uniformly by the other component in the other layers. For extracting the required data from different data format, it uses configurable data parsers, e.g., for extracting specific data from XML, XML Path Language (XPath) could be used. The result is then converted as simple data types (e.g., integer, double, string, bytes).

5.1.2.2 Sensor Fusion Layer

As explained previously, sensor limitations may result only in a partial view of the physical events or noisy measurements that must be compensated to retrieve the actual state of the objects being observed. Therefore, this layer is introduced providing the necessary processes to extract data that represent the actual physical events as close as possible.

Sensor fusion algorithms can only be generalized to a certain extent since they depend on the input data types and the desired output. The data delivered by the sensors could range from analog data, digital pulse, and numeric data to higher dimensional data such as videos and images. As a consequence, the sensor fusion algorithms also have a broad range from simple arithmetic operations up to processing images and videos.

Complex Event Processing (CEP)⁵² tries to provide different abstractions for sensor event processing. These CEP engines can be configured to aggregate, fuse, and extract information from streams of events using domain specific language designed specifically for fusing time series data streams. CEP engines are not only used for processing sensor events, but also in network security to identify intrusion attempts (Ficco & Romano, 2011), in the banking and financial service to identify e.g., trends in the stock market, credit card fraud (Adi et al., 2006).

⁵² <http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/complex-event-processing-088095.html> (Retrieved on August 1, 2014)

Data oriented CEP such as Esper⁵³ considers sensor streams as data collections that can be queried using query languages similar to a database. Alternatively, network oriented CEP engines such as Storm⁵⁴ enable parallel processing of the sensor data through a network of sensor fusion processes.

IoTLink abstract sensor fusion algorithms through sensor fusion objects that must be implemented with specific algorithms. The sensor fusion objects must consider intensive processes that could block the whole system, therefore, the developers must have the option to execute them as background processes.

In order to keep a separation of concern between sensor fusion processes, each process should be designed to execute a specific task. To combine different processes, the output of each process can be an input to another process. This approach allows developers to create a network of event processing similar to the concept that is introduced by Storm.

5.1.2.3 Virtual Object Layer

The third layer contains virtual objects that represent the physical objects that are the concerns of the problem domain. Thus, in this architecture, physical sensors and actuators are only seen as supporting devices that should be transparent to the applications. This layer should contain the semantic representations between physical objects. The structure of each virtual object follows the Metamodel in Figure 45, which shows that each virtual object may have properties, unique ID, and services representing its physical domain object counterpart.

Virtual objects can be categorized into stationary physical objects with fixed relations to sensors and actuators (e.g., rooms that have temperature sensors installed on the wall). The second category is for the objects that move from one location to another and do not have fixed relations with sensors and actuators that can be used to determine their contextual information e.g., rooms occupants whose temperature sensation can be determined by temperature sensor used to measure the room temperature. This type of relationship was discussed by Zimmermann as a shared context (A. Zimmermann, 2007).

Typical examples of static objects include building structures, production lines, heavy equipment which are observed by sensors dedicated to them. Typical examples of dynamic objects are items being manufactured in conveyor belts, products sold in refrigerated shelves, and building occupants. These objects can be observed by the sensors when they are in the sensing coverage of the sensors. BEMOCOFRA project presents a real-world use case where moving object is used to model car parts that are manufactured in a production line. The sensors are attached to the manufacturing cells. However, as an item enters a cell, some sensors in that cell are used to measure the condition of the item being manufactured and these readings must be correlated to the corresponding item during this period. As the item leaves the cell when the process is finished, another item enters the cell. Thus, the correlation of the sensors in that cell must be changed to the new item that just entered the cell.

Each virtual object may contain properties to represent its state, e.g.; a room could have a temperature which is measured by a temperature sensor. Nonetheless, the correlation to the

⁵³ <http://esper.codehaus.org/> (Retrieved on August 1, 2014)

⁵⁴ <http://storm.incubator.apache.org/> (Retrieved on August 1, 2014)

sensors must be defined by the concrete objects in the main canvas. The object may also contain functions that represent services offered by the physical objects. This mapping is used by the code generator to route the values of the sensors to the object being observed.

Modeling virtual objects could be done in different ways, e.g., object-oriented programming or modeling. However, IoTLink focuses on using model driven approach using a visual modeling language in order to support actors with different background. Following the MDD approach, the virtual objects should be defined using a modeling language based on Platform-Independent Metamodel (PIM) which is derived from IoT-A Metamodel shown in Figure 45. Using a generic modeling language, such as UML could overwhelm users with little technical background. In addition, UML poses significant overhead for modeling small system prototypes. Thus, the author proposes a simplified domain-specific modeling language that illustrates the relations between IoT components. When defining the domain model, each object must have a class that defines its structure. Similar to the object-oriented paradigm, classes are useful for abstracting the objects and maintaining structure changes across a large number of objects. The properties of the classes could have simple data types as well as a complex property type of another class. Classes could also have functions with a return type and parameters that may have simple or complex types. Thus, IoTLink must provide a visual tool for developers to define classes with the structure similar to object-oriented. Moreover, the developers should be able to link the properties of each virtual object to the representations of sensors that observe them in the physical world or to sensor fusion modules that fuse sensor data. Therefore, the modeling language should allow the users to create classes and their instances to represent the concrete virtual objects and link the properties to their type and the sensors that observe them.

5.1.2.4 Output Layer

The output layer is responsible for exposing the virtual objects to the application logic which can be done in two ways. First when the application logic and the virtual objects are implemented within the same application, the virtual objects could be done as software objects and may interact with the application logic that access them. To increase the flexibility of the applications in cases where the logic of the applications could change quite frequently, the application logic could be defined on top of a rule engine that interacts with the virtual objects. Rule engines are designed to decouple business logic from the rest of the application enabling the business rules to change without even restarting the application. In the context of IoT, this scenario is quite useful, particularly for the research environment where various application logic must be evaluated. E.g., to optimize energy consumptions in a building, several control strategies must be evaluated.

Second, when the application logic is done in external applications, they must be able to communicate with the virtual objects through a communication channel. Therefore, the virtual object must be exposed through communication interfaces that are commonly used within the IoT domain. Furthermore, monitoring applications usually need to store the data in a persistent database to be further analyzed by the users or automatically by other applications. Therefore, the historical state of the virtual objects must be able to be stored in a persistent database.

Considering these two possibilities, IoTLink must first transform the virtual object which is defined in a platform-independent model into a platform-specific model. Then, the application logic can be defined within the same application using the platform-specific model for the first case. In the second case, the virtual objects that have been transformed into a platform-specific model must be serialized to the data format that can be understood by the communication technology, which is used to publish the virtual objects.

5.1.2.5 Polyglot Communication Design

IoTLink design supports different communication means on the lowest layer which deals with connections to heterogeneous data sources including physical devices. On the fusion layer, the data are processed, aggregated, and fused to extract the required information about the domain objects. This information is then mapped to the domain objects which can be accessed through different communication means provided by the output layer. Consequently, IoTLink is able to act as a translator by transforming the data it receives through different protocols into the desired communication protocols. This capability is quite essential to bridge the syntactic incompatibility between IoT distributed components which present one of the major obstacle to accomplishing the IoT vision.

5.1.3 IoTLink Workflow Design

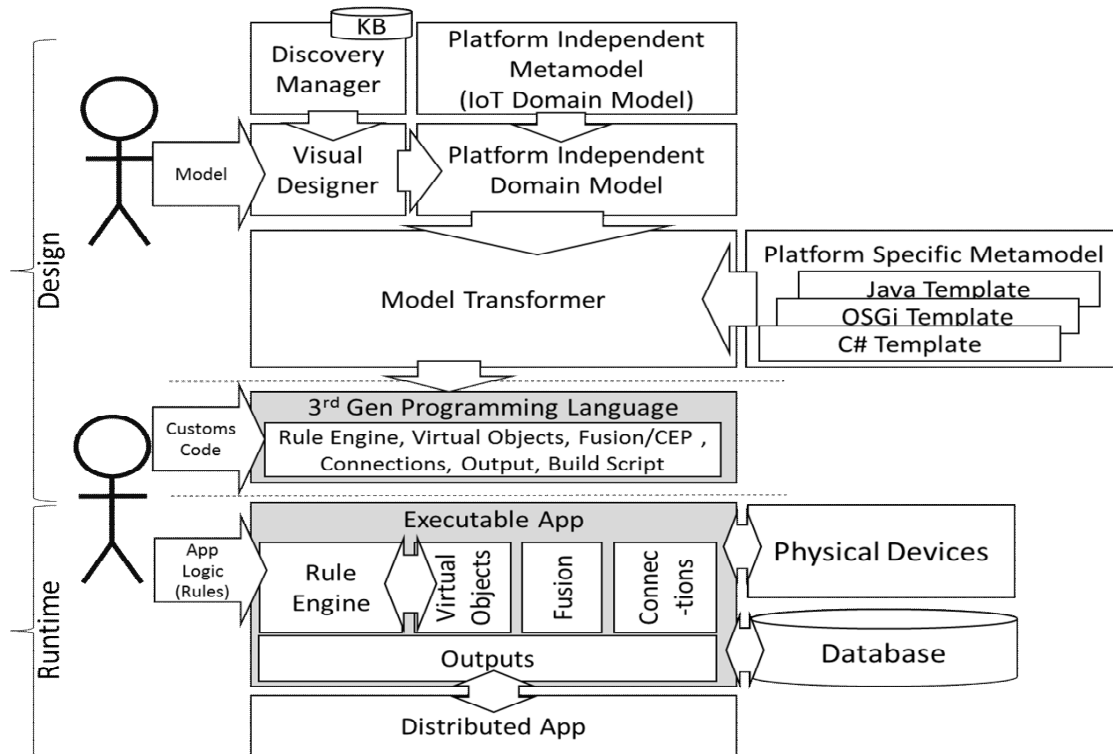


Figure 47. IoTLink Development Concept

The envisioned MDD approach with IoTLink is shown in Figure 47. It comprises two phases, including a design process and runtime. At the design process, the developers define a visual

model with IoTLink's visual designer. When the devices to be used are not known, the developers could find shared devices through the discovery manager. The discovery manager uses the information in its knowledge base to provide information how the shared devices can be accessed. When the devices to be used are known, the developers can directly use the connection components provided by the visual designer. Then, the developers select the required components and them to the domain objects that model how the data are processed and flow from physical devices to the virtual objects then to the output components.

The visual designer takes the visual model and extracts the platform-independent model based on the IoT metamodel. When the developers command IoTLink to generate the implementation of the model, the model transformer transforms the platform-independent model into a platform-specific model based on the platform-specific metamodel, which comprises templates of third generation programming language. When the third generation programming language is generated, it comprises the implementations of each layer of the proposed architecture. These components are wired according to the model. The developers may modify it according to their needs, e.g., implementing application logic to react to the state of the objects. However, they need to separate their implementations with the generated programming language to avoid that their modifications are overwritten when the code needs to be regenerated.

The programming language can then be compiled as an executable application which can run as proxies to the physical domain objects. When it is run, the application initializes a communication with the physical devices to obtain the required data, as well as to send actuation commands. The data is then passed to the chosen components in each layer and then assigned to the virtual objects. When the state of the virtual objects is to be stored in the persistence database, the application establishes communication with a database management system and stores the states of the physical domain objects. When a rule engine is added to the model, the developers are able to modify the application logic using a rule language that can be embedded in the code before it is compiled. Alternatively, rule engines usually allow them to exchange the rules at runtime to modify the application behavior without stopping the applications.

As explained in Section 5.1.2.4, the virtual objects could also be exposed through various communication technologies that can be accessed by distributed applications on the network. This requires the virtual objects to be mapped onto data formats that are required by the chosen communication technology. For technology such as web services, this mapping is done by serializing the data structure of the objects defined in the 3rd generation programming language such as Java onto data formats such as XML, JSON, or SOAP.

To simplify the configuration process, IoTLink generates most of the required configuration consistently according to the domain model defined by the developers. For instance, the database schema, and the data format that is exposed by the output components follow the domain model by default.

5.1.4 IoTLink User Interface Design

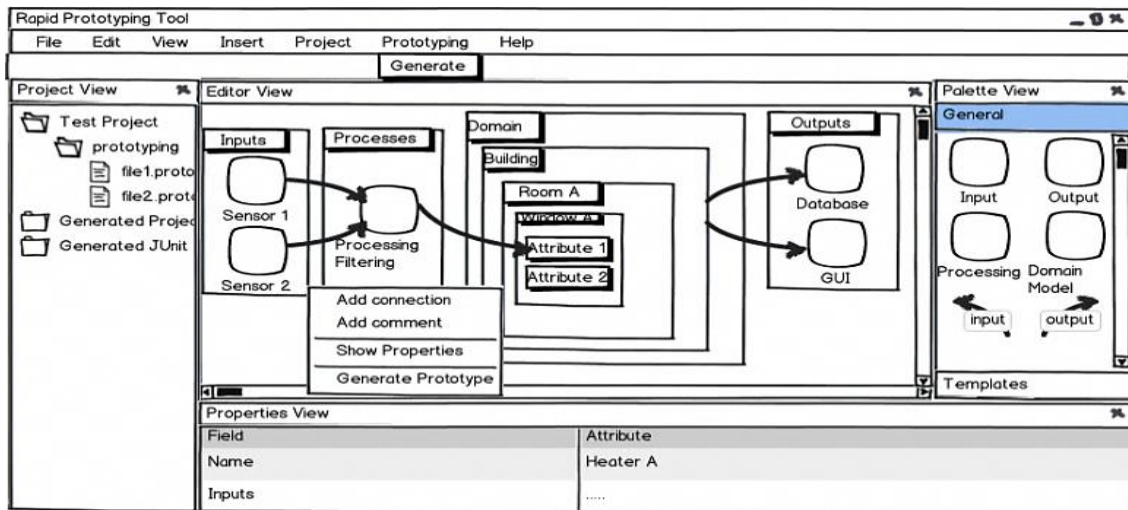


Figure 48. The mock-up user interface for the development tool (Rusmita, 2012)

A low fidelity wireframe was developed using Balsamiq⁵⁵, which is illustrated in Figure 45. The user interface design projects the components in the architecture of a model-driven tool. Initially, the components are arranged in a palette view that can be selected by the users and used in the main workspace. In the main workspace, the components are grouped based on the architecture layer to provide a clear separation of concern. As depicted in Figure 48, the components can be linked together to configure the direction of the data flow. Each component has specific constraints that regulate their input and output, e.g., Input components can only be connected to sensor fusion components or to an attribute of a domain object, and the sensor fusion component could be connected to another sensor fusion component or to a property of the domain objects. These constraints make sure that the developers make the correct compositions as envisioned by the proposed architecture.

The middle container is where the users have to define the domain objects. Initially, IoTLink used UML object diagrams as the object model. However, the users who have an electrical engineering background did not understand the diagram and mentioned that they need a simpler diagram that resembles how the physical objects are placed. Therefore, the author proposes to present simple rectangles that could contain other rectangles since they resemble how physical objects in buildings or manufacturing stations are organized.

At the bottom side, the property window of each component is shown which allows users to configure the behavior of the components. On the left side, the users could see the model that they are currently working on as well as the Java code that is supposed to be generated from the model.

⁵⁵ <http://balsamiq.com/products/mockups/>(Retrieved on July 3, 2014)

The design was evaluated by the users and went through several times of changes according to users' feedbacks. For instance, initially the components were not grouped, and it confused the users when the number of components is high.

5.1.5 IoTLink Implementation

After the user interface design had become mature, several technologies to implement IoTLink were investigated. Several options were considered by this work. The first option was to develop IoTLink as a native android application so that the domain experts could use their tablet to create the model rapidly while they are surveying the client site. However, after gathering the feedback from the subject-matter experts, they prefer to work with pen and paper when analyzing their clients' system since this gives them the best freedom to record any information obtained from their users and they concerned that using tablet might distract them from their work. Moreover, several developers mentioned that having a model on a tablet limits the possibility of having a complex model since tablet has a limited screen size and support limited input device.

The second option was to use a web-based technology such as HTML5, which allows development being done from any device having a browser that supports HTML5. After investigating the current HTML5 capability, this work concluded that HTML5 cannot provide the best user experience for the domain experts nor developers in terms of performance and flexibility. Moreover, when IoTLink should produce Java code that can be extended by professional developers, all features needed by developers to be productive in Java development such as Java parser and code completions must be implemented from scratch which is not the focus of this research.

The third option was to develop IoTLink as Eclipse⁵⁶ plug-ins. The developers preferred the last option since Eclipse offers many features for managing Java development such as a Java code parser, code completions, automatic build, and support for the code repository. Therefore, this option offers the most promising solution to support the further development of the generated Java code.

⁵⁶ <http://www.eclipse.org/> (Retrieved on June 2014)

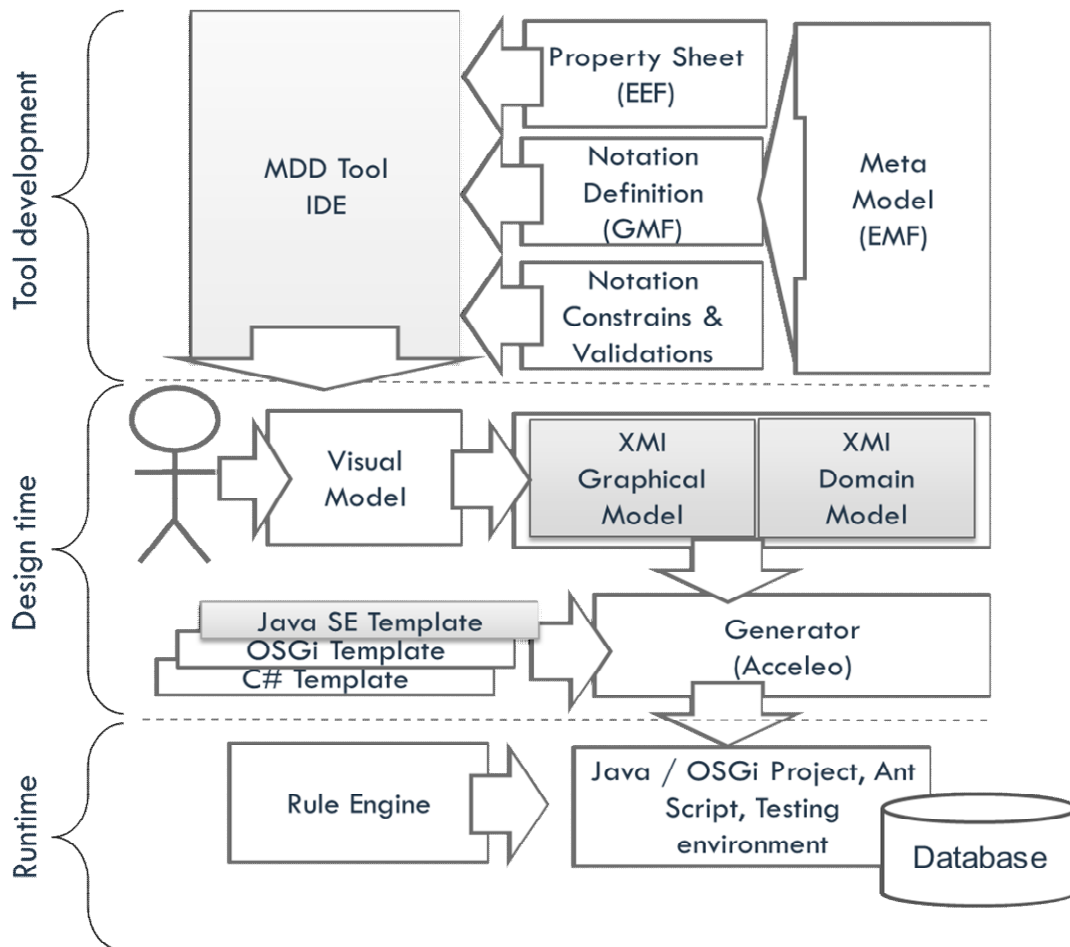


Figure 49. Illustration of Eclipse plug-ins used to build IoTLink

Eclipse IDE is designed as a Rich Client Platform (RCP) for developing general purpose desktop applications. Eclipse allows different sets of plug-ins to be added for extending its functionality. Eclipse's plugins management system is based on the OSGi framework specification (O. Alliance, 2007). Its user interface is built upon the Standard Widget Toolkit (SWT) framework (Northover & Wilson, 2004). Developing IoTLink on top of Eclipse was done by exploring the Eclipse Modeling Project (Gronback, 2009) which has already provided the necessary building blocks for developing a custom modeling language and model transformation which can be used to generate programming language such as Java. After a careful investigation, the following plug-ins are selected for developing IoTLink:

- Eclipse Modeling Framework (EMF) to define the Metamodel of the modeling language (Steinberg et al., 2008)
- Eclipse Graphical Modeling Framework (GMF) to create a graphical editor (Foundation, 2007)
- Extended Editing Framework (EEF)⁵⁷ to create a property editor for the EMF elements

⁵⁷ <http://www.eclipse.org/modeling/emft/?project=eef> (Retrieved on June 13, 2014)

- Aceleo to create a model transformation from the EMF elements into Java code (Musset et al., 2006).

Figure 49 illustrates a high-level architecture, which consists of the building blocks that were implemented to develop IoTLink and the components produced by IoTLink at the design time as well as the component produced at the runtime.

To develop IoTLink, a Metamodel of visual language was defined using an EMF model. The Metamodel is designed based on a layered architecture framework depicted in the Figure 46. The layers in the architecture are implemented as core classes which are then extended with the concrete implementations to represent the components that IoTLink supports. For instance, the Connection class is subclassed by the ArduinoSerial, Plugwise, SOAPInput, RESTInput, and LinkSmartInput classes. The Fusion class is inherited by the VotingFuse, MinMax, Average, and KalmanFilter. The Output class is inherited by the RelationalDatabase, SOAPOutput, and RESTOutput. This design allows IoTLink to be extended easily in the future by inheriting the core classes shown in the green color in Figure 50.

The Metamodel is then derived by GMF to define the graphical definition model, known as *gmfgraph*, which determines how the graphical notations look like as well as the relations and constraints between the notations. Furthermore, GMF derives a tooling definition model, known as *gmftool*, which defines the content of the palette menu, the menu containing the components supported by IoTLink such as the connections to the sensors. The notations to be displayed on the main canvas and the items listed in the palette are then mapped in a mapping configuration, known as *gmfmap*, which is used by GMF to decide which notation should be shown on the main canvas when an item from the palette is dragged and dropped to the main canvas. To create a property sheet for each component on the palette, IoTLink uses EEF. It derives the Metamodel to generate an EEF model that defines the content of the property sheet for each component. This model can be used to define a custom behavior of the property sheet such as adding a validation on the text fields, or changing the fields with specific widgets which could simplify entering the required information such as checkbox for a Boolean entry.

GMF by default is configured to serialize the model defined by the users in XMI format, a commonly used data format for exchanging UML models, introduced by the Object Management Group (OMG). GMF can be configured to separate the object model containing the domain model defined in the diagram and the graphical definition containing information related to the visualization of the domain objects (e.g., position, layout, and fonts).

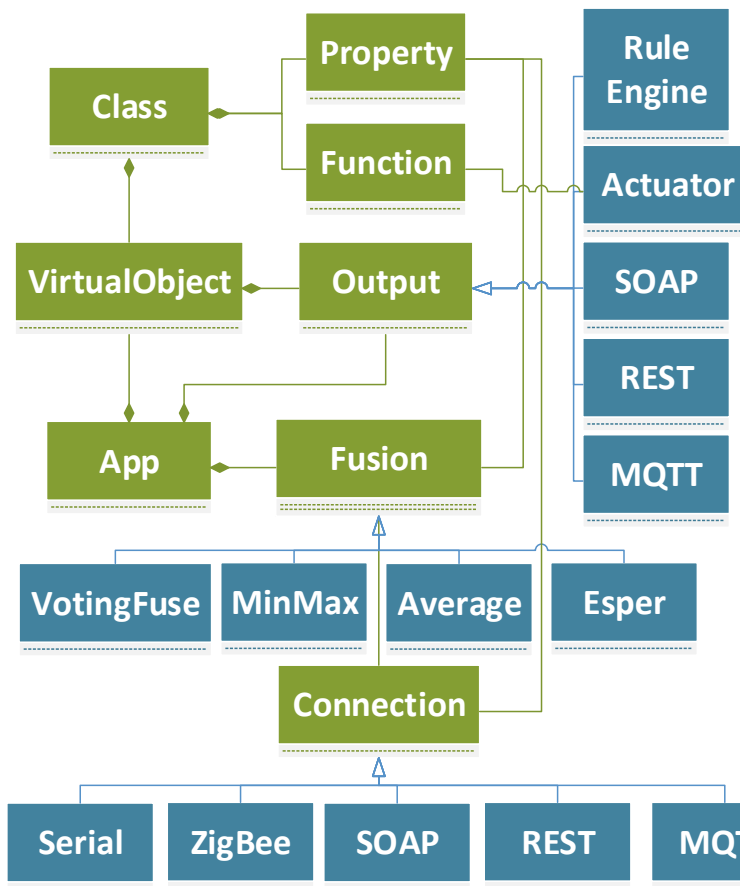


Figure 50. Logical view of the Platform-Independent Metamodel (PIM) for IoTLink (the properties of the classes are omitted to simplify the picture).

The domain model data are used by the code generator component to generate the necessary Java code based on the model transformation rules. The code generator was implemented initially with XPand⁵⁸. However, after the first prototype was developed, the performance of the code generator was unacceptable for the users and therefore the code generator was re-implemented using Acceleo. Currently, the transformation rules are able to transform the model into software artifacts, including Java code, the build script to simplify the deployment, and the required configuration files. In addition, it is able to generate a web page that is useful for testing the prototype. Other programming languages could be generated as well by providing the code generator with the corresponding transformation rules.

The initial implementation of IoTLink (depicted in Figure 51) that provides a basic prototype of IoTLink that contains only connections to the Arduino, Plugwise smart plugs, and the XPand code generator. The goal of the initial version was to be evaluated by the users and gather users' feedback at the very early phase of the development. The evaluation will be further elaborated in Chapter 8.

⁵⁸ <http://www.eclipse.org/modeling/m2t/?project=xpand> (Retrieved on August 20, 2014)

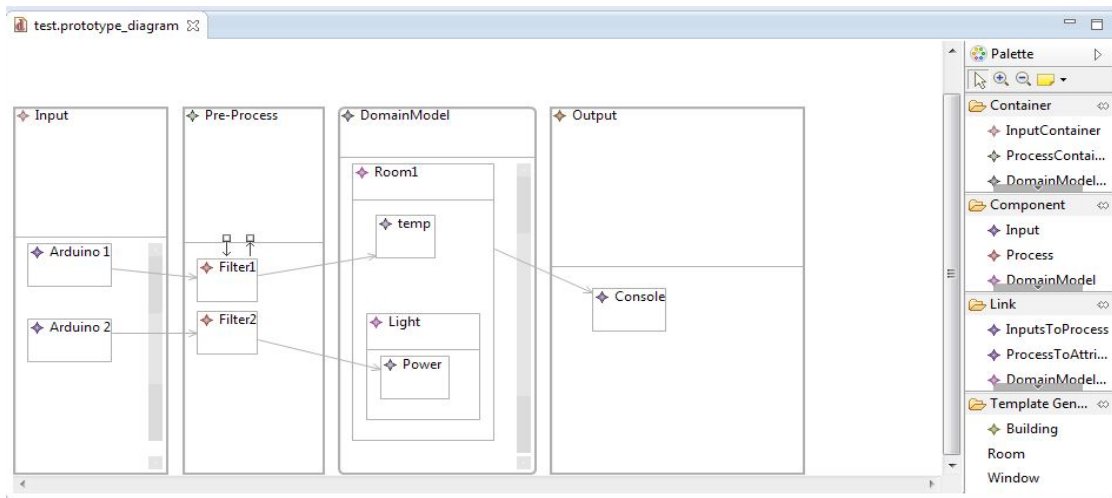


Figure 51. The first iteration of the model development tool

After several iterations, the features related to the usability were implemented, and the result is depicted in Figure 52. Now, the validation of the diagram shows error messages and mark the notations with red icons when the required information is not yet completed. The components use different colors and icons to help developers differentiate them. In addition, several wizards are implemented to guide developers filling in the required information. Some of the wizards are able to present the possible information required by the component. For instance, the components that require the users to define a serial port shows a list of active serial ports and the wizard of SOAPInput provides the method and parameters of the Web Service that the user could easily choose from.

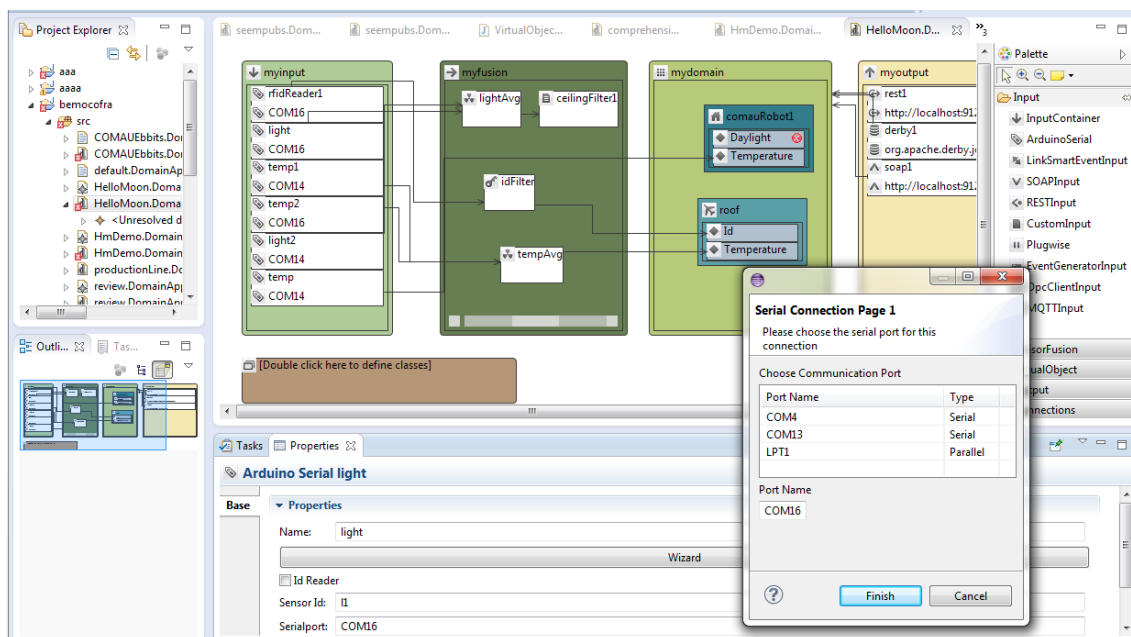


Figure 52. The final iteration of the model development tool

The main interface of IoTLink is the main canvas (in the middle) which contains five rectangles as depicted in Figure 52. The first four rectangles with green and beige colors are containers to group different components to be used for building a prototype. This separation provides a clear view of the components as well as the flow of the connections between components. The first four rectangles include the following containers (from left to right):

- Connection container
- Sensor Fusion container
- Virtual Object container
- Output container

The last rectangle with a brown color is a place holder for the template classes. The template classes work similarly to classes in an object-oriented programming. The classes define the structure of the objects. This helps the developers to maintain changes, especially when the number of the object is quite large. By changing the structure of the class, the structure of the objects having the same class are adjusted automatically.

On the lower side of the user-interface, the property sheet of the active object is displayed. The property sheet allows developers to configure the component by entering the necessary information. For instance, the main canvas requires configuration of the Java project to be generated by the code generator. As illustrated in Figure 53, the developers have to enter the base package for the Java codes, the name of the project, the name of the application which is used as the main package name, and the path where the code should be generated.

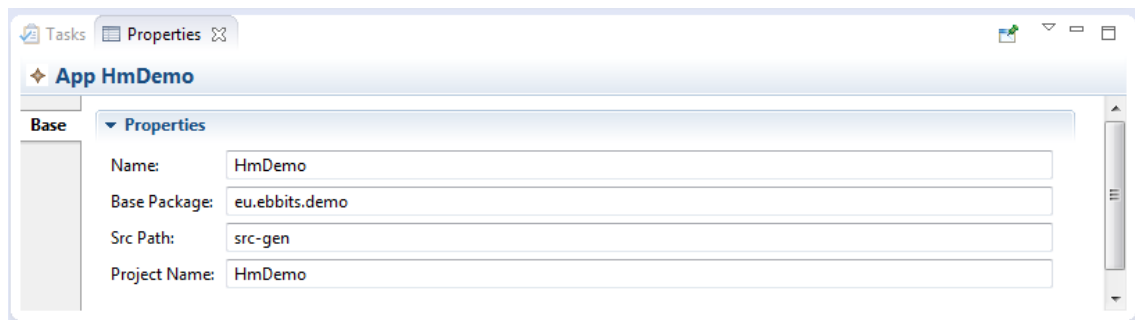


Figure 53. The setting of the project to be generated on the property sheet

The right side of the user interface shows the palette menu containing components that IoTLink supports. The menu is also grouped into Connections, Sensor Fusion, Virtual Object, Output, and Links.

5.1.5.1 Implemented Connection Components

The connection components in the palette are responsible for building connection to a data source which could be physical devices, sensor networks, or software interfaces such as Web Services that provide an access to the actual sensors.

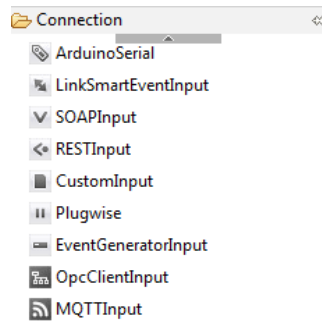


Figure 54. The connection components

Every connection has a flag to let the system know whether it is a connection to a sensor that deliver ID data such as an RFID reader or it is a connection to other kind of sensors that deliver the contextual information about an object. When the flag is set true, the system treats the sensor values as an ID of an object within the vicinity of the sensor. This implies that the context of the objects could be determined by the sensors near to the ID reader. Therefore, the system should correlate the sensor values to the corresponding object. IoTLink provides a way to model this use case by defining the physical objects that are mobile as “Moving Objects” which is elaborated more detail in section 5.1.5.3.

Currently, IoTLink supports connections to devices that are widely used for IoT prototyping as well as well-known network protocols that are used to enable communications in IoT e.g.:

ArduinoSerial component enables communication with Arduino⁵⁹ boards. Arduino has been widely used for hardware prototyping because of its low cost, offers a simple programming model, and the flexibility to be extended with analog or digital sensors and actuators. The boards can be reprogrammed to read and write to its IO pins as well as to communicate with a PC through USB port that emulates a serial communication. IoTLink is able to parse sensor data that are sent by an Arduino board with the following format:

$$*[\text{SensorID}]=[\text{Value}]\$[\text{Checksum}]\#$$

Whereas the SensorID is used to indicate different sensors connected to the board and the checksum is used to make sure that the package sent by the board is correctly received by the PC.

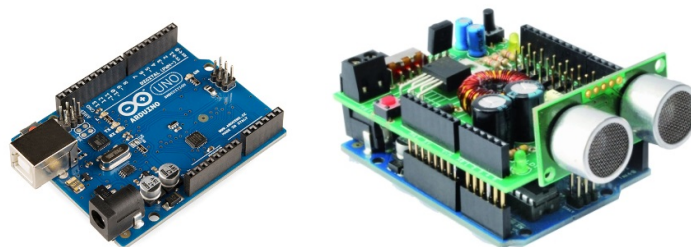


Figure 55. Arduino boards which can be extended to create a more complex sensor and actuator.

⁵⁹ <http://www.arduino.cc/> (Retrieved on August 1, 2014)

LinkSmartEventInput component which enables the generated code to communicate with LinkSmart proxies that publish sensor events through a LinkSmart event manager. LinkSmart middleware has been used in more than 11 European research projects relevant to IoT. During these projects several LinkSmart proxies have been developed to provide an abstraction layer between the physical devices, manufacturer specific implementations and reusable components for building applications. IoTLink is able to subscribe LinkSmart events which contain the following key-value pairs as the event payload:

1. sensor value, e.g., key="value", value="1.2"
2. sensor id, e.g., key="sId", value="X001223132"
3. when the measurement was taken, e.g., key="timestamp", value="2013-08-12 09:30:12.543"

SOAPInput, SOAP based Web Services are widely used to enable communications between various enterprise applications over HTTP protocol that reduces the hurdle of configuring network firewall for different applications. In the last decade, SOAP based Web Service has been proposed for abstracting the communication to the devices in order to achieve horizontal integration in businesses. There exists a lightweight implementation of soap based Web Service for devices (Jammes et al., 2005). Thus, it makes sense to include a component that is able to communicate with this protocol. The *SOAPInput* component can be used by the user to pull the data exposed through a SOAP Web Service. Using this component, the user must define the method of the service to be invoked using a wizard as depicted in the Figure 56.

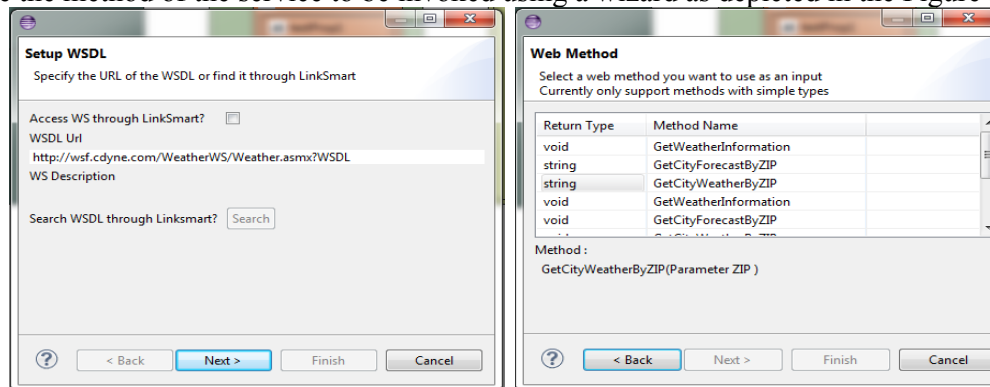


Figure 56. Wizard for selecting the Web Service method

Since the data sent through a Web Service is marshalled in a SOAP envelope, the *SOAPInput* component needs to be configured to extract the relevant sensor values from SOAP messages. For this purpose, an XPath⁶⁰ expression must be used by the users to parse the incoming soap objects. XPath allows developers to query a piece of information from an XML document. For instance, Table 3 shows a truncated soap message from a weather service ws.cdyne.com and the XPath expression to extract the required sensor data.

⁶⁰ <http://www.w3.org/TR/xpath/> (Retrieved on August 1, 2014)

Table 3. SOAPInput uses XPath to extract single data from a SOAP message

<pre><soap:Body> <GetCityWeatherByZIPResponse XMLns="HTTP://ws.cdyne.com/WeatherWS/"> <GetCityWeatherByZIPResult> </pre>	<pre>Xpath expression : //temperature</pre>	<pre>Result : 64</pre>
<pre> <Temperature>64</Temperature> <RelativeHumidity>96</RelativeHumidity> <Pressure>30.07F</Pressure></pre>	<pre>Xpath expression : //humidity</pre>	<pre>Result : 96</pre>
<pre> </pre> <pre> </GetCityWeatherByZIPResult> </GetCityWeatherByZIPResponse> </soap:Body></pre>	<pre>Xpath expression : //pressure</pre>	<pre>Result : 30.07F</pre>

RESTInput, REST provides a lightweight alternative to SOAP-based Web Service and has gained significant popularity for enabling communication between applications on the internet. Big vendors such as Google and Amazon have offered REST interface for their online services. REST does not enforce any data format to be used, but instead encourages the use of self-describing messages by adding the format metadata in the HTTP header of the message (Fielding, 2000b). The *RESTInput* component allows the users to retrieve data by polling a REST service, hosted on a specific URL. Until now, REST services generally use XML and JSON format. To anticipate the two formats when extracting a single sensor value, this component uses an XPath and JSONPath (Goessner, 2007) (XPath like expression for JSON) expressions to parse the incoming XML and JSON respectively.



Figure 57. Plugwise devices use ZigBee for enabling wireless monitoring and automation in homes and commercial buildings⁶¹

PlugwiseInput, this connection provides an example of supporting commercially off the shelf devices. Plugwise uses ZigBee technology to enable wireless communication between sensors and actuators which can be used in the home automation domain. The Plugwise circle (the first device depicted in Figure 57) is able to measure the power consumption and switch the electricity. These devices are able to communicate with a PC through a USB receiver. *PlugwiseInput* communicates with the receiver through a serial communication and able to pull the power consumptions from the Plugwise devices that are connected to the USB

⁶¹ <http://www.plugwise.com/products/#filter=devices> (Retrieved on August 1, 2014)

receiver. Additionally, the *PlugwiseInput* is able to send messages to each device to perform the actuation such as switching on and off the power.

OPCClient, this component enables an integration of industrial devices into the generated code through an OPC middleware which has been widely accepted as a de facto standard for application integration in the industrial environment. As described in section 3.1.1, OPC consists of an OPC server, which uses the native device drivers to communicate with industrial controllers and provide a unified data access through Microsoft DCOM (Horstmann & Kirtland, 1997) or Web Service connection for the newer specifications (OPC XML-DA and OPC-UA). The OPC server mirrors the memory address of industrial controllers (PLC) that store the actual sensor values and the state of the devices that are connected to them. To access the data on the OPC server from third party software, an OPC client must be used. The *OPCClient* component uses *UTGard* which is a Java implementation of *OPCClient* API, implemented in the context of Eclipse OpenScada project (Rose & Reimann, 2014). *UtGard* is able to access OPC servers that support OPC-DA v2.0. The *OPCClient* component can be configured to pull an OPC variable.

MQTTInput, this connection enables the generated application to receive data from an MQTT broker (Locke, 2010). MQTT is an emerging communication standard for IoT that adopts publish-subscribe paradigm. MQTT has been used for enabling sensor communication via satellite link, dial up connections, and used in healthcare, home automation, and utility companies (Lampkin et al., 2012). MQTT features a small footprint and three levels of QoS, which makes it ideal to run on devices with limited resources and unreliable network with low bandwidths. *MQTTInput* can be configured for subscribing a topic which is used by devices to publish the sensor values. This component requires that the payload of the event contains the following data delimited by a semicolon “;”:

- sensor value, e.g., "value=1.2"
- sensor id, e.g., "sId=X001223132"
- time when the measurement was taken, e.g., "timestamp=2013-08-12 09:30:12.543"

EventGeneratorInput, for testing purposes *IoTLink* provides a connection to a data generator. It could be configured to generate a random sensor value every second as well as to generate ID values every second.

CustomInput, since this work cannot foresee all possible components during the implementation, the *IoTLink* must provide a way for the user to include input components which are not yet implemented as part of *IoTLink*. Thus, *CustomInput* allows users to define directly a Java class within the modeling language. The users have options to only define a Java interface that must be implemented after the code is generated, or to define a Java class with the implementation directly from *IoTLink*. To support that latter, *IoTLink* provides a Java code editor.

5.1.5.2 Implemented Sensor Fusion Components

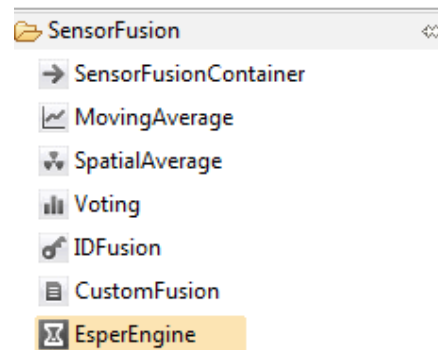


Figure 58. Supported Sensor Fusion Modules

IoTLink provides several predefined sensor fusion algorithms that could be used to process noisy sensor data. When necessary, the sensor fusion modules can be combined as a network of processes as depicted in Figure 59. This allows data to be processed through a network of algorithms until the desired information is extracted. Each of the processes is performed in separate a thread to enable parallel processing of sensor data which increases the performance of the application particularly on a multicore hardware. Moreover, keeping the processes in separate threads prevents any complex data processing that requires a long time to process, blocking the whole system.

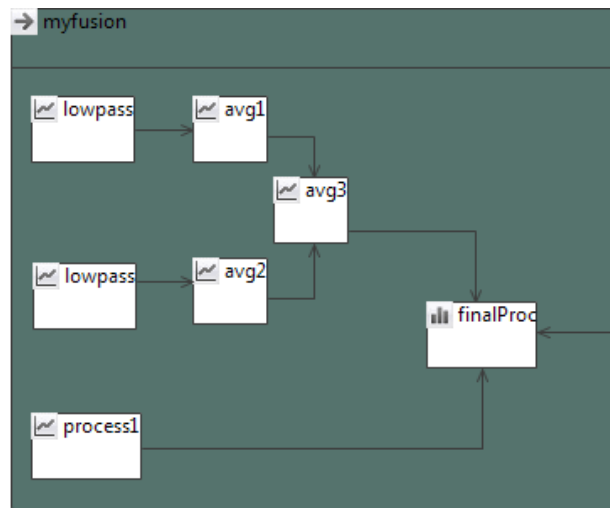


Figure 59. A mesh of sensor fusion modules

The available algorithms that are currently supported by IoTLink are as follows:

4. *Moving Average Filter* averages the last N data or the last N data within a time frame. Moving Filter is commonly used for a time series data to observe a long term trend instead of depicting a short term fluctuation which might be caused by sensor noise (Wei, 1994).
5. *Spatial Average* averages the data from a cluster of sensor that have the same type and observe the same events. A cluster of sensor is normally used in critical applications to

obtain data with high confidence since providing redundancy could compensate the failure of a sensor by other sensors in the cluster.

6. Voting Fusion determines the property of an object based the majority of Boolean value provided by a cluster of sensors observing the same event. Similar to the spatial average, the cluster is also used to compensate a sensor failure.
7. ID Fusion fuses duplicate ID data that are sent by an ID reader. The algorithm ignores the same id being sent multiple times in a short time frame.

Complex Event Processing

In addition to the implemented algorithms, IoTLink must anticipate that more advanced users need a sensor fusion component that can be configured flexibly to extract useful information from the sensor streams. This requirement can be fulfilled by the existing CEP implementations such as Esper, Oracle CEP, and Storm. IoTLink has integrated Esper, which is a leading open source CEP engine whose performance and reliability has been proven by big companies such as PayPal, Accenture, Raytheon, and Oracle. Esper takes an advantage of the query language similar to SQL, called Event Processing Language (EPL). EPL enables users to query information from the event objects that are inserted into the engine. Since the generated code abstracts sensor streams with the SensorData class, the users are able to query the information from the SensorData objects that have been inserted into the engine. The event streams originating from all data sources are identified by IDs which can be used to perform processing of that particular stream. For instance, Figure 60a shows an example of performing an average on the last five sensor data from the sensor with an Id of “Id1”.

```
a) SELECT avg(value) as avgValue FROM SensorData(Id="Id1").win:length(5)
b) SELECT sum(value) FROM SensorData group by timestamp
```

Figure 60. An example of EPL query of a moving window average (a) and sum of the sensor values based on the timestamp (b).

EPL statements determine how the sensor streams should be processed by the engine. EPL Statements may consist of one or more views. Views may represent the windows of the stream as well as statistics of the streams. The example shown in Figure 60a contains a view representing a window of five sensor data which is expressed in “win:length(5)”. ESPER also allows aggregation and grouping using “group by” and “having” clause which is useful to, e.g., calculate the sum of values based on a particular group as depicted by Figure 60b.

Using Esper engine, the users only need to define EPL statements in the property sheet of EsperEngine as depicted in Figure 61. Similar to other sensor fusion components, each EsperEngine block will be generated as a separate thread, and the result of the thread is pushed to the component that is connected to it e.g., in Figure 61, the last ten sensor values belong to the “a1” is averaged and pushed to the “Prop1” of the “chlidarray_0” by the “esperAvg” block.

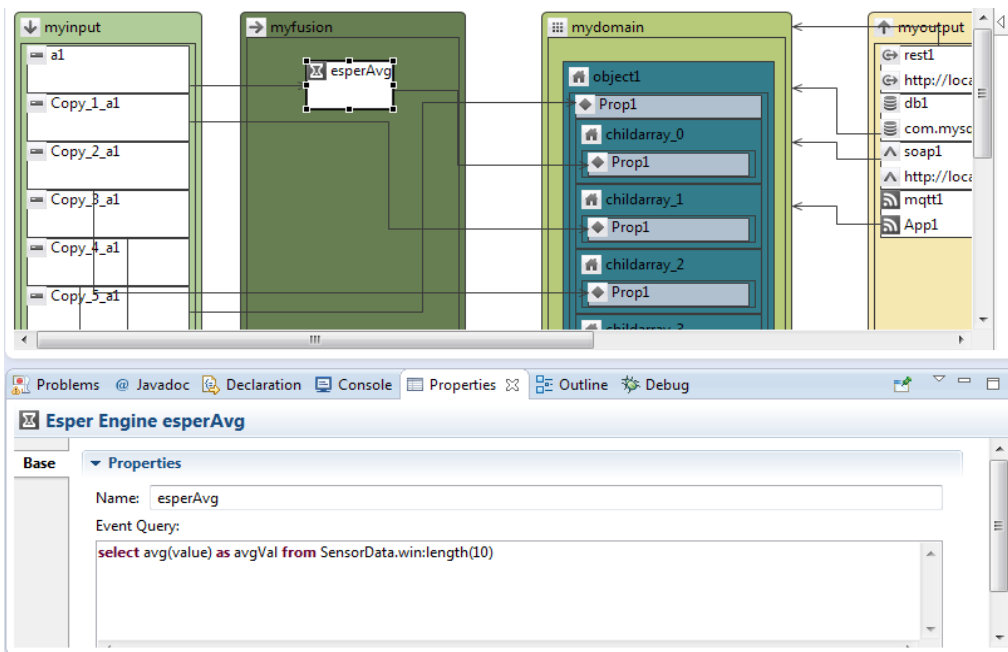


Figure 61. Configuring Esper engine through EPL in the property sheet of the Esper Engine component.

Customized sensor fusion algorithm

Despite the Esper’s flexibility to process sensor stream, it would not be possible to cover all sensor fusion use cases with only a solution. For instance, when dealing with picture or sound data, other sensor fusion algorithms must be used. Therefore, IoTLink allows the users to define their own customized algorithms in the Java language that implements the “SensorFusion” interface. In this case, the sensor data are represented in a byte array, which is quite generic to represent digital signals. The users must implement two methods. First, the syncFuse that runs the fusion process in the main thread and blocks the system during the fusion process. Second, which is also the recommended way to use, is to implement the asyncFuse method that must process the sensor data in a separate thread.

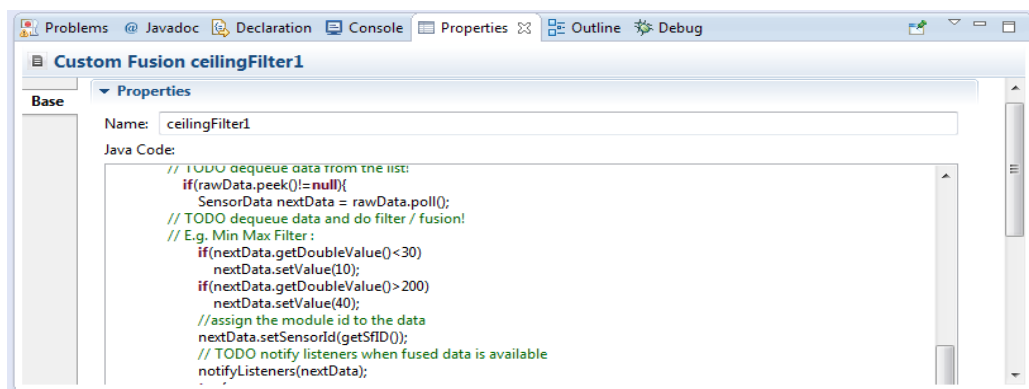


Figure 62. Defining the sensor fusion algorithms in Java.

5.1.5.3 Virtual Object Components

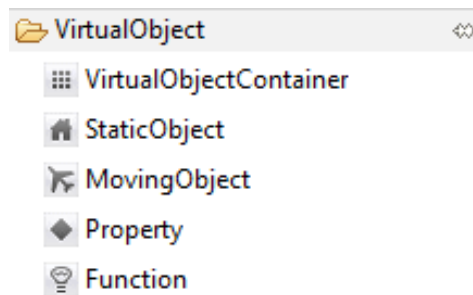


Figure 63. Components to build the objects

The virtual object menu provides detailed components that can be used to define the concrete virtual objects to represent the physical objects.

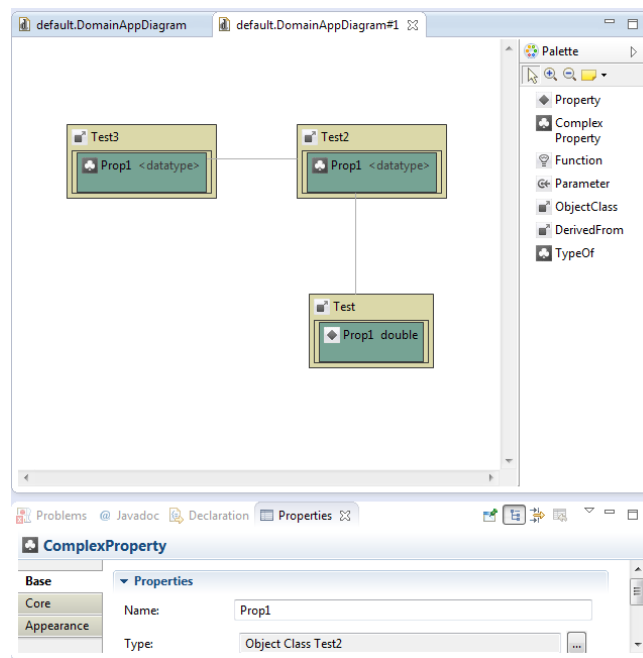


Figure 64. Defining class template

As Figure 64 shows, IoTLink allows developers to create classes and assign them to concrete virtual objects (Figure 65). When a class is assigned to an object, the structure of the class is replicated into the object. Having the complete structure of each object is necessary because the users might want to link the concrete virtual objects to the sensors that observe them. When the structure of a class is changed, the changes are propagated to the virtual objects with that type. This approach is very useful when maintaining a large number of concrete objects. However, during the design iteration, the author found out that for a very simple prototype that only requires one or two objects per class, creating a class for them seems to be unnecessary overhead, therefore the author added a feature that allow the developers defining the concrete object first, then convert the concrete objects to classes.

IoTLink separates the class definition on a child diagram in order to simplify the user interface. To define a class first the developers have to double click on the TemplateContainer (the lowest rectangle with a brown color in Figure 52) which opens a child canvas. After the classes have been defined, the developers can create either a static or moving object and assign the class through a wizard as depicted in Figure 65. When the class of the object has been set, the class structure is copied to the object, and a link between them is defined. This link is used by IoTLink to update the objects when the structure of a class is updated. As shown in the Figure 65, IoTLink provides three options how it should copy the structure of the class or object. The default behavior duplicates the structure of a class into all objects that have a reference to it similar to how the object-oriented programming works. The first option merges all properties, functions, and children of a class and its concrete objects. The third option duplicates the structure of an object to a class. The third option is useful when the developers want to create only an object for a small prototype, whereas the second option is useful when the developers have to maintain changes to many objects of the same class.

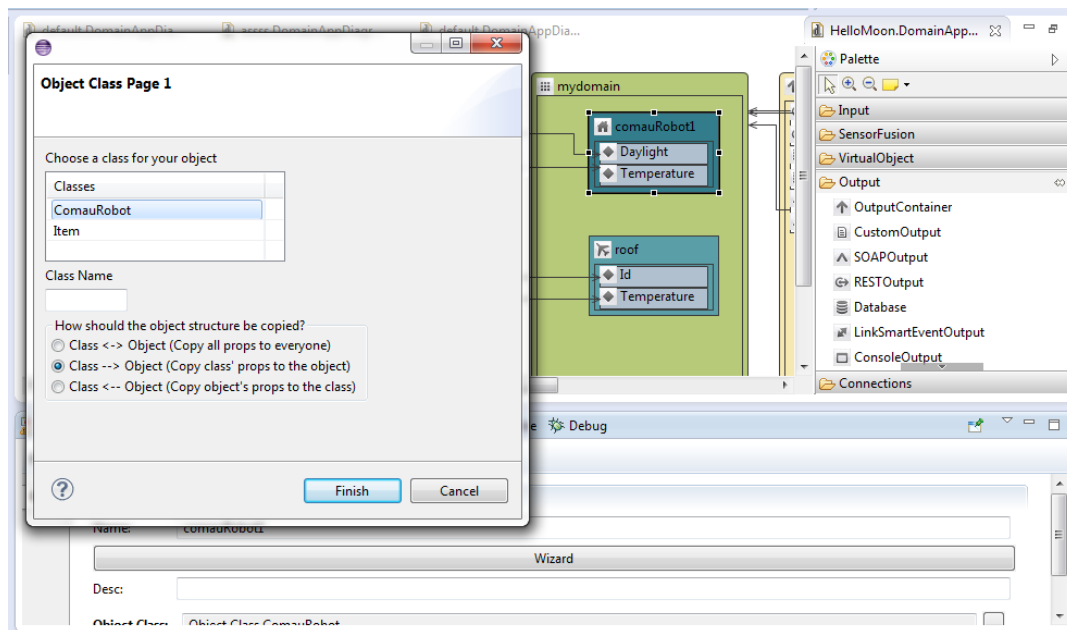


Figure 65. Setting the class of an object

Linking Properties and Functions

After the connections, sensor fusion, and virtual objects are defined, developers can define the relation between the sensor connections and the virtual objects as well as with the sensor fusion modules when required. Linking a sensor connection to a property of a virtual object describe that the sensor is used to observe the physical world events that determine that particular property of the virtual object. When two or more sensors are required to determine the property of a virtual object, the sensor connection could be linked to a sensor fusion module that fuses the sensor values from both sensors by applying a certain algorithm. In addition, the developer could link each function of the virtual object to an actuator. When the function is called, it forwards the parameters to the actuator connection object.

5.1.5.4 Implemented Output Components

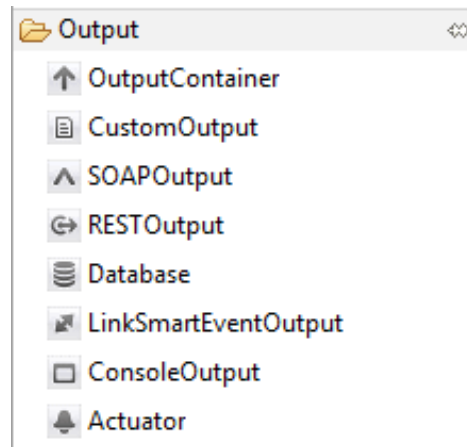


Figure 13. Output components

The output components define how the virtual objects can be serialized and retrieved by external applications. At the moment, there are several implementations that allow virtual objects to be exposed through Web Services, as tables in a database, or through event brokers such as MQTT. In addition, the virtual objects could also be pushed to a rule engine that can be configured to react based on the actual state of the virtual objects.

Mapping virtual object to SOAP based Web Service

SOAP based Web Service is the standard backbone for service oriented architecture which has been used extensively for enterprise application integration (EAI) and enabled business to business (B2B) communication. In the enterprise environment Web Service technologies have become a de facto standard for enabling interoperability (Vernadat, 2003). To enable communication, Web Services rely on a protocol stack consisting of:

- Transport protocol, responsible for transporting the messages between applications. Several transport protocols could be used, e.g., HTTP, SMTP, and TCP. However, in practice, HTTP dominates the Web Service implementations since most networks have been configured to allow HTTP traffics.
- Messaging protocol, responsible for serialization and deserialization of messages exchanged between applications. An XML-based format such as XML-RPC and SOAP usually are used since they offer a good readability which simplifies tracing the messages when problems arise. SOAP is widely supported by various programming languages, the implementations still vary which hamper the interoperability between the applications written using different libraries.
- Description protocol, responsible for describing the structure of the service, including the methods, functions, parameters, and data structure which can be used by applications to generate the stub code. The Web Service description language (WSDL) is a description language that has been used widely.
- Discovery protocol, responsible as a registry where services could be published, and applications could find the services that they need. UDDI provides a specification how this could be implemented. However, the adoption of the UDDI is quite slow so far.

SOAPOutput is pre-configured to generate the necessary code that configures Apache CXF (Apache, 2009) with the most frequently used configurations so that the users do not need to be bothered with the required configurations. The default configurations use the HTTP protocol as a transport, SOAP 1.2 as the message protocol, and it generates a WSDL automatically.

IoTLink maps the virtual objects to SOAP messages by generating JAX-WS annotations on the Java beans classes. To retrieve the state of the physical objects, the *SOAPOutput* component generates a Web Service with methods that return the virtual objects depending on the classes defined in the domain model. E.g., if there is a static object with an Id of “UB01” has a type of “WeldingCell”, the output component will generate a Web Service method named `getWeldingCel` (String Id). Since all static objects use their names as IDs, the users could invoke `geWeldingCel` (“UB01”) to retrieve the state of object UB1. The result is serialized into the following SOAP message as depicted in Table 4.

Table 4. An example of representing a virtual object with a SOAP-message.

Request	
<pre><Envelope XMLNs="HTTP://schemas.XMLsoap.org/soap/envelope/"> <Body> <getWeldingCell XMLNs="HTTP://mydomain.virtualobject.bemocofra.eu/"> <arg0 XMLNs="">UB01</arg0> </getWeldingCell> </Body> </Envelope></pre>	
Response	
<pre>Content-Length: 524 Server: Jett (9.0.6.v20130930) Content-Type: text/XML; charset=UTF-8 <soap:Envelope XMLNs:soap="HTTP://schemas.XMLsoap.org/soap/envelope/"> <soap:Body> <ns1:getWeldingCellResponse XMLNs:ns1="HTTP://mydomain.virtualobject.bemocofra.eu/"> <return XMLNs:ns2="HTTP://mydomain.virtualobject.bemocofra.eu/"> <class>WeldingCell</class> <description/> <LastUpdate>2014-05-07T11:47:46.510+02:00</LastUpdate> <Humidity>34.0</Humidity> <Lighting>867.4</Lighting> <PowerConsumption>0.0</PowerConsumption> <Temperature>24.0</Temperature> </return> </ns1:getWeldingCellResponse> </soap:Body> </soap:Envelope></pre>	
	<p>Properties of the virtual object UB01</p>

Actuation service

To represent the actuation services of the physical objects, the *SOAPOutput* generates a method for each function that is modeled in the virtual object. For instance, if the developer

created a welding cell class offering a “SetOn (boolean status)” service which is connected to the physical actuator, a method in the Web Service named ”weldingCellSetOn(String id, boolean status)” is generated by the *SOAPOutput*. The method requires the id of the object that should be invoked and uses the id to forward the request to the corresponding physical actuator.

The routing of actuation request is possible because the user has defined the relation between the connection object, a virtual object, and the actuator object in the model. From this model, the code generator generates a hash map containing the relation between the virtual object IDs, the names of the functions, and the connections to the physical devices which are used to forward messages from the output components to the virtual objects and finally to connection objects.

Mapping virtual object to REST based services

REST is an architectural style designed for hypermedia (Fielding, 2000a). REST exploits HTTP protocol to provide an abstraction for software resources. The architectural style consists of four interface constraints:

- Resource identification through Uniform resource identifier (URI)
- Uniform interface to create, read, update and delete (CRUD) resources through HTTP GET, POST, DELETE and PUT messages.
- Self-descriptive messages which mean that the media type of the messages must be indicated in the transfer protocol
- Stateful interactions through hyperlinks which mean that direct interaction with a resource is stateless and stateful interactions can be achieved by state transfer.

RESTOutput generates code which maps the virtual objects and their properties as resources in the REST architectural style. To implement this architecture, each virtual object must be identified by a unique URI that contains its unique Id. Additionally, to provide a clear structure of the virtual objects, the *RESTOutput* groups them based on their classes and makes these resources available through URLs that fulfil certain patterns as depicted in Figure 66. Figure 66 illustrates a URL to access a virtual object with Id of “UB01” and a type of “WeldingCell” which is derived from the base class “Virtual Object”. Since the UB01 object is also a type of “Virtual Object”, it could also be retrieved with a shorter URL: HTTP://localhost:9123/rest/ virtualobject/UB01.

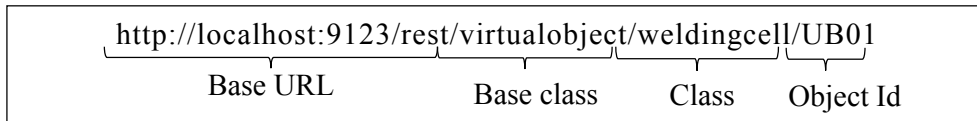


Figure 66. An example of the URL generated by the RESTOutput component to access a virtual object

The states of the virtual objects can be encoded in XML and JSON formats depending on the metadata sent by the client. For instance, Table 5 shows how the *RESTOutput* encodes the states of the UB01 object with XML and JSON formats.

Table 5. Response and request of the virtual object through the RESTOutput

Request	Request
Accept: Application/json	Accept: Application/XML
Response	Response
Content-Type: application/json Date: Wed, 07 May 2014 14:17:11 GMT Server: Jett (9.0.6.v20130930) <pre>{ WeldingCell: { class: "WeldingCell" description: "" id: "UB01" LastUpdate: "2014-05-07T16:17:11.099+02:00" Humidity: 33 Lighting: 290.8 PowerConsumption: 0 Temperature: 25 } }</pre>	Content-Type: application/XML Date: Wed, 07 May 2014 14:21:53 GMT Content-Length: 324 Server: Jett (9.0.6.v20130930) <pre><?XML version="1.0" encoding="UTF-8" standalone="yes" ?> <WeldingCell> <class>WeldingCell</class> <description /> <id>UB01</id> <LastUpdate>2014-05- 07T16:21:52.973+02:00</LastUpdate> <Humidity>33.0</Humidity> <Lighting>319.8</Lighting> <PowerConsumption>0.0</PowerConsumption> <Temperature>25.0</Temperature> </WeldingCell></pre>

Moreover, the *RESTOutput* component also generates URLs to retrieve a list of available objects based on their classes. These URLs “all” postfix instead of any specific IDs in the last part of the URL which look like the following pattern:

- [Base url]/virtualobject/all
- [Base url]/virtualobject/[Class]/all

An example of the URL to retrieve a list of virtual objects “CarFrame” which has a child of “Engine” is illustrated in Figure 67.

```

▼<CarFrames>
  ▼<CarFrame>
    <class>CarFrame</class>
    <LastUpdate>2013-10-17T18:47:18.110+02:00</LastUpdate>
    <BodyType>25.2</BodyType>
    <Id>X000001</Id>
  </CarFrame>
  ▼<CarFrame>
    <class>CarFrame</class>
    <LastUpdate>2013-10-17T18:47:21.132+02:00</LastUpdate>
    <BodyType>24.8</BodyType>
    ▼<Engine>
      <class>Engine</class>
      <LastUpdate>2013-10-17T18:47:21.629+02:00</LastUpdate>
      <EngineSize>0.0</EngineSize>
      <Id>X000003</Id>
    </Engine>
    <Id>X000002</Id>
  </CarFrame>
  ▼<CarFrame>
    <class>CarFrame</class>
    <LastUpdate>2013-10-17T18:47:24.632+02:00</LastUpdate>
    <BodyType>26.3</BodyType>
    ▼<Engine>
      <class>Engine</class>
      <LastUpdate>2013-10-17T18:47:25.129+02:00</LastUpdate>
      <EngineSize>0.0</EngineSize>
      <Id>X000004</Id>
    </Engine>
    <Id>X000003</Id>
  </CarFrame>
    
```

Figure 67. REST output for retrieving the values of the objects

Actuation service

Representing the actuation services of the physical objects, the *RESTOutput* component maps each actuation service as a resource under the corresponding virtual object. The function and its parameters can be accessed through URL query parameters which look like the following pattern:

[base url]/virtualobject/[classname]/[id]/[funcName]?[param=value]

For instance, let us assume that the developers create a function “SetOn(boolean status)” belongs to the virtual object “UB01”. When the Java code is generated, the *RESTOutput* maps the function to the following URL:

HTTP://localhost:9123/rest/virtualobject/UB01/setOn?status=false

When the actuation URL is called, the request is forwarded to the corresponding Java object which then forwards the request to the corresponding connection object based on the links that the users have defined in the visual model.

Mapping virtual objects to relational database

In most cases, particularly for exploring the behavior of the physical objects, the users need to store the state of the virtual objects over time in a persistent storage which can be analyzed for extracting useful information. E.g., energy consumption data over time could be analyzed to find saving potential.

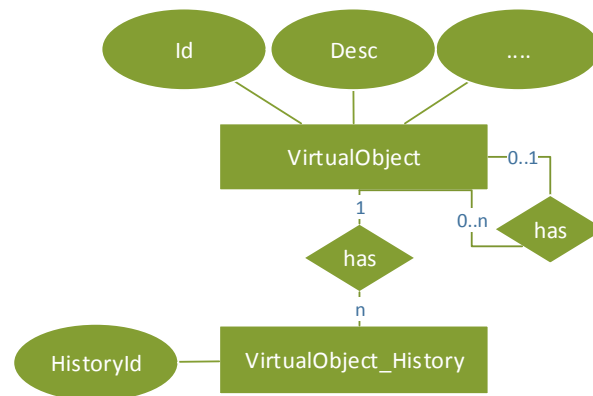


Figure 68. Virtual object schema generated by IoTLink

For this purpose, IoTLink provides a *DatabaseOutput* component which is able to establish a connection to a database management system, generates the database schema, and automatically stores the historical as well as the actual states of the virtual objects. The *DatabaseOutput* component uses EclipseLink⁶², which implements the Java Persistence API (JPA) (DeMichiel & Keith, 2006) to map Java objects into a relational or document based database (Plugge et al., 2010). Since IoTLink generates Java objects and classes based on the model defined by the users, *DatabaseOutput* only requires adding JPA annotations on the Java classes. The annotations are used by EclipseLink to generate the corresponding tables in the database. At runtime, the current state of the virtual objects is stored in tables corresponding to their classes. In addition, another table for each class is created to store the historical states of the physical objects every time their states have changed. Figure 68 shows an example of the database schema which contains a table for storing the current state and a table to store historical states of the virtual objects. A virtual object may have other virtual objects as its children. They are stored in separated tables which are referenced by the IDs as foreign keys in the parent's table. Please note that the generated the "VirtualObject" table is replicated and named according to the Java classes defined in the model.

Publish subscribed based events

When the users plan developing distributed applications, which need to get the status of the physical objects, IoTLink is able to push the states using publish-subscribe pattern. IoTLink supports two event brokers, which are the LinkSmart event broker and MQTT event broker. LinkSmart event broker relies on SOAP Web Service communication, which requires the subscribers to provide a Web Service, implementing an "EventSubscriber" interface. Then the application could subscribe to specific event topics corresponding the virtual object.

Both *MQTTOutput* and *LinkSmartEventOutput* could be configured to publish events with two types of topic format. First, a flat structure topic that only includes the class name of the object, the object id, and the property name as follows:

Topic = baseTopic/virtualobject/[Classname]/[Object Id]/[Property name]

⁶² <http://www.eclipse.org/eclipselink/> (Retrieved on August 14, 2014)

The topic structure allows the developers to subscribe all events based on the class of the virtual objects using a wildcard topic. For instance, in the building automation domain, when developers are interested in events related to all rooms, they could subscribe to the following topic.

Topic = baseTopic/virtualobject/room/*

When developers require a high-performance event broker and have a very small data, such as foreseen in telemetry applications, IoTLINK allows developers to use MQTT event broker which is designed for this purpose. MQTT recently has been used for IoT applications because its reliability for handling wireless transmission. A widely used MQTT broker is called mosquitto⁶³ uses a TCP connection and a very small protocol as explained briefly in MQTT specification (Locke, 2010). The *MQTTOutput* component also uses the same topic structure which also allows the users to subscribe to individual objects as well as classes of objects using the wildcard character (note that the wildcard character in mosquitto is a hash '#').

The second format follows a hierarchical structure containing the objects id as shown by the following example

Topic = baseTopic/virtualobject/[Object Id1]/[Object Id2]/.../[Property name],
where the subsequent object is a child object of the prior object.

The second topic pattern allows the application to subscribe to all events belong to an object and its children. E.g., when an application would like to observe room one and all devices in that room, it needs to subscribe to the following topic.

Topic = baseTopic/virtualobject/room1/*

Console and file logging

This component enables developers to log all messages to trace problems when they occur. This output component uses the Log4J framework (Gülcü, 2003) which can be configured to log messages into different output channels such as a console window, a log file, or a database. Log4J could also be configured to log messages which are tagged with eight levels of messages ranging from “OFF” which means no messages are logged, “FATAL” which means that it logs severe error messages, “ERROR” which means it logs runtime error messages, “INFO” which means that it logs informative messages, “DEBUG” and “TRACE” which mean it logs detailed messages used for tracing the flow of the system.

Drools rule engine

In an automated environment, IoT applications are required to perform an actuation based on the states of the physical objects. The behavior, responding to these changes could be decoupled by implementing them on a rule engine. There exist many rule engines that are used in business environments. These engines are categorized based on how they execute the rules. The Forward Chaining engines execute consequence based on conditions expressed in the rule which implies “IF then ELSE” type of logic. The Backward Chaining engines, also

⁶³ <http://mosquitto.org/> (Retrieved on August 14, 2014)

called goal oriented, try to resolve the facts that fit the goal. Drools is able to perform reasoning using both approaches and draw conclusion based on the facts and rules fed to the engine (Bali, 2009).

Drools allows the rules to be defined in different language dialects. The native Drools dialect is called *mvel*, which follows a simple structure as depicted in Figure 69.

```
rule "name"  
  attributes  
  when  
    LHS (conditions)  
  then  
    RHS (consequences)  
end
```

Figure 69. Drools rule language format

Drools evaluate the rules when new facts are inserted into the engine or when the facts have changed. During the rule evaluation when the conditions of the rules are met, the consequence part of the rule is executed. If conditions of several rules are met, Drools apply a conflict resolution strategy by changing the order of executions based on the salience value of the rules. This requires developers to provide the priority of the rules when they define the rules. The rules could be stored in a database and maintained using Drools Guvnor, which offers a web-based interface for defining and editing rules as depicted in Figure 70. The web-based interface is able to provide the users with a domain specific language (DSL) which can be tailored close to a natural language. However, a mapping between the phrases used in the DSL and the rule language must be provided by the developers.

The *DroolsOutput* component provided by IoTLink can be configured to retrieve the rules from a Guvnor Database and instantly apply any rule changes when the generated applications are still running. This allows different control strategy to be investigated during the prototyping phase. Enabling this feature, the *DroolsOutput* component must be configured with the URL of the Guvnor. When the Java code is generated, the code generator also generates a jar file containing the domain model of the application prototype that could be imported to the Guvnor to enable type-safe feature when defining the rules.

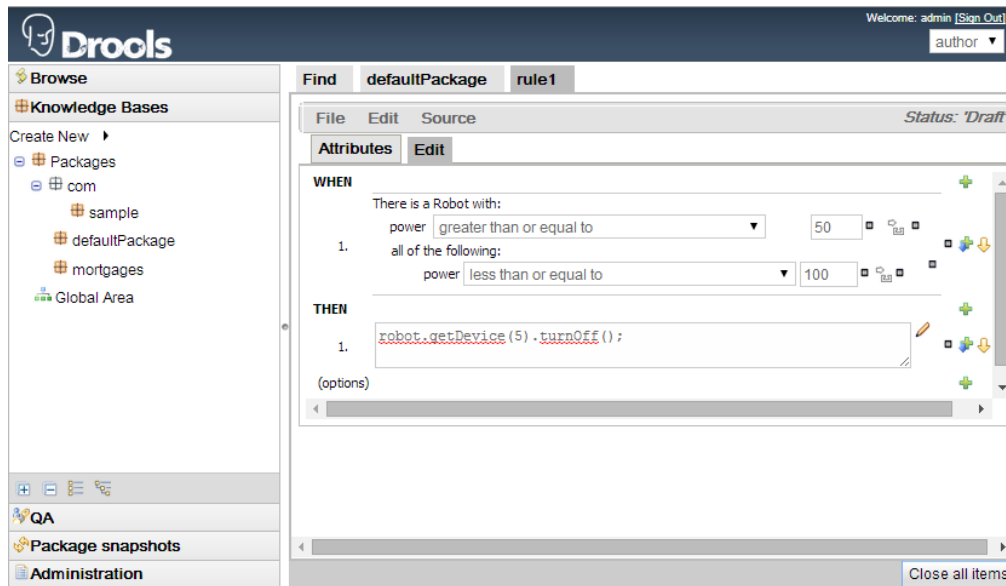


Figure 70. Defining rules, responding to the state of the physical objects.

Actuator

The actuator component provides an abstraction for devices or part of a device that could perform an action. For instance, each Plugwise smart-plug has a switch that could be turned on and off. The actuator component is used to represent this switch. The actuator component can be mapped to the functions of the virtual objects to perform an action when they are invoked. This means the function and the actuator must be linked. When defining an actuator component, the ID of the actuator must be provided so that the connection component knows which actuator should be used. This feature can only be supported when the objects are connected to SOAP- and RESTOutput.

5.1.5.5 Diagram Serialization

IoTLink serializes the visual diagram into two XMI files which have *.DomainApp extension and *.DomainAppDiagram extension. The first file (DomainApp) contains the semantic definition of the input, output, and sensor fusion components as well as the virtual objects while the second file (DomainAppDiagram) contains the definition of the visual elements that visualize the elements in the first file such as the location of the notations on the screen, the font used by the notations, and colors of the notations. The structure of the first file follows the domain model defined in the visual diagram which serialize components into XML elements under the relevant container element. The configuration of the components is serialized as XML attributes within the XML elements. Each XML element is identified by a unique Id which is referred by the visual elements in the second XML file.

Figure 71 illustrates the serialization a prototype model shown in Figure 52. The XMI data shows that under the virtualobjectContainer there is a StaticObject named comauRobot which has a property “Daylight” with a data type of “double”. The Daylight property is connected to a sensor fusion module with an id of “_7GF94LHGEeOzOfz2PfgUjQ” that refers to “ceilingFilter1” element under the sensorFusionContainer element. The ceilingFilter1

receives an input from “lightAvg” which in turn receives inputs from two ArduinoSerial components, “light” and “light2”. All connections from a component to another component is done through referencing the ID of the elements of a specific attribute as defined by the ECore metamodel depicted partially in Figure 50 e.g., a Property has an attribute of “fusedInput” that holds a link to a sensor fusion module. A sensor fusion module has an attribute “inputs” which holds links to other sensor fusion modules or Input components. The Output components have “virtualobjects” attribute that link each output to a virtual object container. These links define the virtual objects to be included with the output components when they are running.

As illustrated in Figure 71, the definition of the “ComauRobot” and “Item” classes which act as templates for the virtual objects are serialized under “templateContainer”.

```

<lsapp:App xmi:version="2.0" XMLNs:xmi="HTTP://www.omg.org/XMI" XMLNs:xsi="HTTP://www.w3.org/2001/XMLSchema-instance" XMLNs:lsapp="HTTP://linksmart.eu/appmodel/0.1" xmi:id="_oSoYUEieEeOxz6VInrA8qg" name="HelloBemo" basePackage="eu.ebbits.demo" srcPath="src-gen" projectName="HmDemoProject">

  <virtualObjectContainer xmi:id="_oSo_YEieEeOxz6VInrA8qg" name="mydomain" output="__OutID1 _OutID2 _OutID3" app="_oSoYUEieEeOxz6VInrA8qg">
    <virtualObjects xsi:type="lsapp:StaticObject" xmi:id="_R1" name="comauRobot1" objectClass="_RobotID">
      <properties xmi:id="_HVeLsEx4EeOV09h9w37a5w" name="Daylight" fusedInput="_7GF94LHGEeOzOfz2PfgUjQ" type="double"/>
      <properties xmi:id="_N9tUcEz7EeOtqaJA_uNPgQ" name="Temperature" fusedInput="_skDW8LaLEeOQmcpKbZz0VA" type="double"/>
    </virtualObjects>
    <virtualObjects xsi:type="lsapp:MovingObject" xmi:id="_XelM8Eq6EeO-MecvdB8x2g" name="roof" objectClass="_uQ8VAEq7EeO-MecvdB8x2g">
      <properties xmi:id="_8Sh00Eq6EeO-MecvdB8x2g" name="Id" fusedInput="_k2o9wEq7EeO-MecvdB8x2g" type="String"/>
      <properties xmi:id="_lvyusLHGEeOzOfz2PfgUjQ" name="Temperature" fusedInput="_HbgTILK-EeOgi-tuWQZSEg" type="double"/>
    </virtualObjects>
  </virtualObjectContainer>

  <sensorFusionContainer xmi:id="_oSo_YUieEeOxz6VInrA8qg" name="myfusion" app="_oSoYUEieEeOxz6VInrA8qg">
    <sensorfusions xsi:type="lsapp:IDReaderFusion" xmi:id="_k2o9wEq7EeO-MecvdB8x2g" name="idFilter" property="_8Sh00Eq6EeO-MecvdB8x2g" inputs="_BTE0Eq6EeO-MecvdB8x2g"/>
    <sensorfusions xsi:type="lsapp:CustomFusion" xmi:id="_7GF94LHGEeOzOfz2PfgUjQ" name="ceilingFilter1" property="_HVeLsEx4EeOV09h9w37a5w" JavaCode = "">
    <sensorfusions xsi:type="lsapp:SpatialAverage" xmi:id="_AzvU8LK-EeOgi-tuWQZSEg" name="lightAvg" inputs="_bTovAEyTEeO6_btxVWQNEQ_gVOikLK6EeOgi-tuWQZSEg" fusion="_7GF94LHGEeOzOfz2PfgUjQ" mode="AVG_WHEN_TIME_ELAPSED" timeInterval="1000"/>
    <sensorfusions xsi:type="lsapp:SpatialAverage" xmi:id="_HbgTILK-EeOgi-tuWQZSEg" name="tempAvg" property="_lvyusLHGEeOzOfz2PfgUjQ" inputs="_DQ8LwE0nEeOsQf2hYAK7hA_EJMxsLK6EeOgi-tuWQZSEg" mode="AVG_WHEN_TIME_ELAPSED" timeInterval="1000"/>
  </sensorFusionContainer>

  <inputContainer xmi:id="_oSo_YkieEeOxz6VInrA8qg" name="myinput" app="_oSoYUEieEeOxz6VInrA8qg">
    <inputs xsi:type="lsapp:ArduinoSerial" xmi:id="_BTE0Eq6EeO-MecvdB8x2g" name="rfidReader1" fusions="_k2o9wEq7EeO-MecvdB8x2g" IdReader="true" serialport="COM16" baudrate="115200" sensorId="id"/>
    <inputs xsi:type="lsapp:ArduinoSerial" xmi:id="_bTovAEyTEeO6_btxVWQNEQ" name="light" fusions="_AzvU8LK-EeOgi-tuWQZSEg" serialport="COM16" baudrate="115200" sensorId="l1"/>
    <inputs xsi:type="lsapp:ArduinoSerial" xmi:id="_DQ8LwE0nEeOsQf2hYAK7hA" name="temp1" fusions="_HbgTILK-EeOgi-tuWQZSEg" serialport="COM14" baudrate="115200" sensorId="t1"/>
    <inputs xsi:type="lsapp:ArduinoSerial" xmi:id="_EJMxsLK6EeOgi-tuWQZSEg" name="temp2" fusions="_HbgTILK-EeOgi-tuWQZSEg" serialport="COM16" baudrate="115200" sensorId="t1"/>
    <inputs xsi:type="lsapp:ArduinoSerial" xmi:id="_gVOikLK6EeOgi-tuWQZSEg" name="light2" fusions="_AzvU8LK-EeOgi-tuWQZSEg" serialport="COM14" baudrate="115200" sensorId="l1"/>
    <inputs xsi:type="lsapp:ArduinoSerial" xmi:id="_skDW8LaLEeOQmcpKbZz0VA" name="temp" property="_N9tUcEz7EeOtqaJA_uNPgQ" serialport="COM14" baudrate="115200" sensorId="t1"/>
  </inputContainer>

  <outputContainer xmi:id="_oSo_Y0ieEeOxz6VInrA8qg" name="myoutput" app="_oSoYUEieEeOxz6VInrA8qg">
    <outputs xsi:type="lsapp:RESTOutput" xmi:id=" _OutID1" name="rest1" virtualobjects=" _oSo_YEieEeOxz6VInrA8qg"
  </outputContainer>

```

```

baseUrl="HTTP://localhost:9123/rest"/>
  <outputs xsi:type="lsapp:Database" xmi:id="_OutID2" name="derby1" virtualobjects="_oSo_YEieEeOxz6VInrA8qg"
  jdbcDriver="org.apache.derby.jdbc.EmbeddedDriver" jdbcUrl="jdbc:derby:reviewDB;create=true" user="review" pass="review"/>
  <outputs xsi:type="lsapp:SOAPOutput" xmi:id="_OutID3" name="soap1" virtualobjects="_oSo_YEieEeOxz6VInrA8qg"
  baseUrl="HTTP://localhost:9123/soap"/>
</outputContainer>

<templateContainer xmi:id="_oSo_ZEieEeOxz6VInrA8qg">
  <classes xmi:id="_RobotID" name="ComauRobot" objects="_R1" package="mydomain">
    <properties xmi:id="_i-mIEloEeOawftYiWwOlq" name="Daylight" type="double"/>
    <properties xmi:id="_MQ2xEEz7EeOtqaJA_uNPgQ" name="Temperature" type="double"/>
  </classes>
  <classes xmi:id="_uQ8VAEq7EeO-MecvdB8x2g" name="Item" objects="_XeIM8Eq6EeO-MecvdB8x2g" package="mydomain">
    <properties xmi:id="_uQ9jIEq7EeO-MecvdB8x2g" name="Id" type="String"/>
    <properties xmi:id="_L5umgE0nEeOsQf2hYAK7hA" name="Temperature" type="double"/>
  </classes>
</templateContainer>

</lsapp:App>

```

Figure 71. An example of domain model serialization in XMI format

5.1.5.6 Generated Platform Specific Model

The development tool is able to generate Java code based on the XMI domain model based on the visual model. Currently, this work has only implemented a code generator for the Java standard edition. However, other programming languages can be generated by providing the required code templates. The generated Java code can be run providing the users with a software representation of the IoT, which can be accessed by external applications through the output components chosen while modeling.

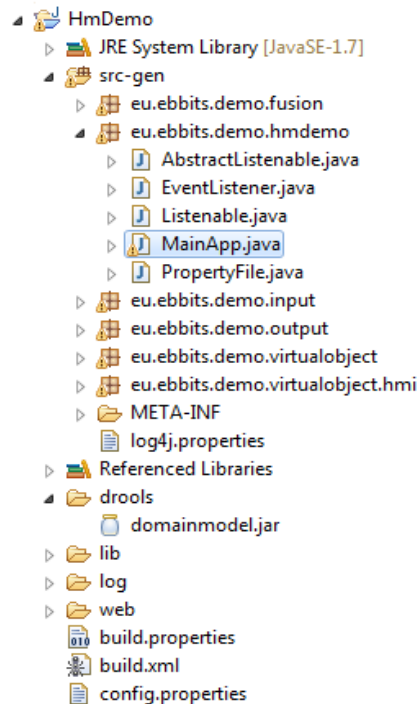


Figure 72. Example of the generated Java project.

The structure of the generated code can be seen in the Figure 72. The generated software artifacts consist of Java source code, a log file, the required runtime libraries, configuration files, a web interface for testing, and an Ant build script that can be used for compiling and produce a runnable JAR. In the example shown in Figure 72, the generated code is structured in a Java project called “HmDemo” and “eu.ebbits” as a base package. The input package (“eu.ebbits.input”) holds all implemented input components which are derived from Connections interface. Similarly, the fusion (“eu.ebbits.fusion”) and output (“eu.ebbits.output”) packages hold the implementation of SensorFusion and Output components respectively. Each class, representing the domain model of the virtual objects is generated in the “eu.ebbits.virtualobjects.hmi”. All naming is configured during modeling by filling in the names on the property sheet of the diagram. To allow communication between the software components in these packages, they adopt a publish-subscribe pattern that is able to propagate the data from the physical devices up to the output components instantly.

The central component that acts as a controller for the whole system is the MainApp class which is generated in the “eu.ebbits.demo.hmdemo”. The interaction between the components of the generated code is depicted in the Figure 73. Firstly, the MainApp initializes the concrete objects of the Connection, SensorFusion, VirtualObject, and Outputs. After the virtual objects are initialized they are stored in a singleton Map within the VirtualObjectFactory classes so that the Input, SensorFusion, and the Output components could retrieve them easily.

The input package consists of connection classes that are responsible for establishing connections to the physical devices, as well as network protocols such as Web Services. Each device type and network protocols may require a specific implementation which must be implemented as a code template that is used by the code generator to produce the Java code.

When the connection objects are linked to the sensor fusion modules, the incoming data from the connections is pushed to the respective sensor fusion modules. The data could go through several levels of fusion depending on how the sensor fusion components are modeled. After the data is processed by the last node of sensor fusion, it is assigned to a property of a virtual object. If the data does not need to be processed through the sensor fusion modules, the connections can be linked to the properties of the virtual object directly. In this case, as soon as the data from the connection objects are available it is directly pushed to the corresponding domain object and assigned to the property to which connection object is connected.

The classes defined in the diagram are generated as Java classes, and the objects are initialized in the MainApp class. When output components that use the annotation framework, e.g., REST-, SOAP- and DatabaseOutput are used, the generated Java beans are annotated correspondingly.

The initialization of the static objects is hard coded in the MainApp class since these objects are already known when modeling the diagram, . On the contrary, the concrete moving objects are not known while modeling the diagram. Therefore, they must be initialized when a new Id is delivered by an Id reader.

The VirtualObjectFactory is responsible for storing the virtual objects in the memory which could be used by any output components such as Database, REST, and SOAP to retrieve their

actual states. For this purpose, the code generator generates methods which can be used to retrieve all available virtual objects based on its class name, Id, or descriptions e.g.:

- getObject(), return all objects that have different types
- getObjectById(), return an object that with the id / null
- getObjectByDesc(), return all objects that contains the description / null
- getObject(Classname)(e) (), returns all objects of type [Classname]
- getObject(Classname)ById (String id), return an object with the id /return null if there is no object with that id
- getObject(Classname)ByDesc(String desc), return all object that contain the desc

In addition, the VirtualObjectFactory is used by some of the output components to provide Web Service methods that forward the function calls of the virtual objects through the connection components. Thus, the code generator generates functions of each class and annotate them to be published as web methods. The generated methods adopt the following format:

[ClassName][FunctionName](String id, String...Parameters),

where Id denotes the Id of the virtual object to be invoked and the array of parameters which are required by the function of the object. For instance, when a class with a name of “Class1”, offer a function with the name of “Func1”. The wrapper function generated in the VirtualObjectFactory is named “*public void class1Func1(String id, String param1)*”.

Moreover, the code generator generates and initializes the objects of the output components in the MainApp class. The output objects are registered as listeners for the property change events of the virtual objects. Acting as property listeners enable them to instantly perform output routines such as writing to the database or transmitting the data over a network, when the state of the virtual objects change.

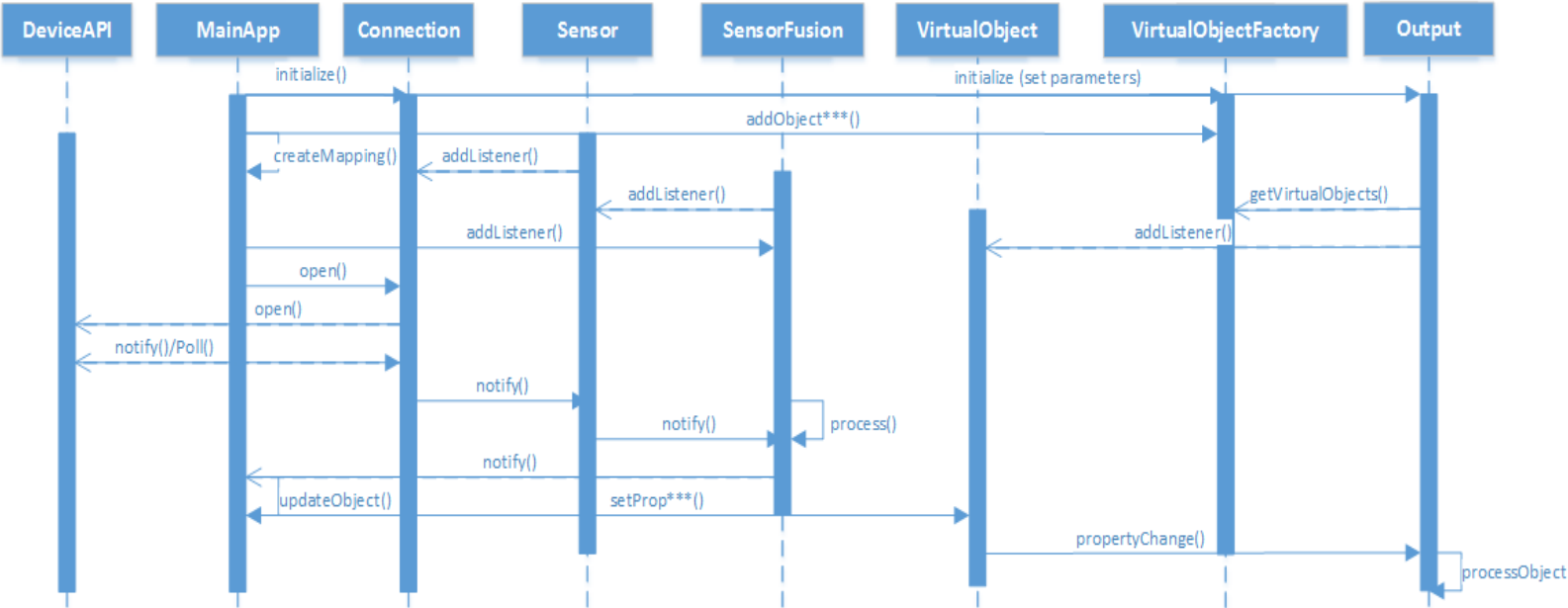


Figure 73. Interaction diagram between the generated classes for updating properties.

Intentionally left blank

Next, the generated objects must be chained together according to the links defined in the graphical model. The links are implemented in the MainApp by making the objects as the listeners of the other objects to which they are linked. Chaining these objects through publish-subscribe method provides a simple solution and yet quite powerful for ensuring that the data is propagated from one component to another component instantly.

Extending the generated prototype, the developers may choose to extend the generated Java code or use it as a standalone application.

Table 6. Example of customizing the generated code in Java and how to obtain the virtual objects.

```
Public class MainApp implements EventListener, PropertyChangeListener {
    .....
    public static void mai (String[] args) {
        MainApp app = new MainAp ();
        app.executeUserCod ();
    }

    @generated NOT
    //example of processing data by polling the property of the object
    public void executeUserCod (){
        TestClass objTestClass = getMydomainObjectService().getTestClass("object1");
        if(objTestClass.getProp1() > 1.0){
            // raise an alert
        }
    }

    //example of processing data based on property change events
    @generated NOT
    @Override
    public void propertyChang (PropertyChangeEvent evt) {
        // Get the object which is just updated
        VirtualObject obj = (VirtualObject) evt.getSource();
        String updatedPropertyName = evt.getPropertyName();
        //do something
        if (updatedProperty.equals("Prop1") &&
            ((Number)evt.getNewValue()).doubleValue() > 1.0){
            // raise an alert
        }
    }
    // End of user code
    .....
}
```

When extending the Java code directly, developers are encouraged to use a Generation Gap pattern (Greenfield & Short, 2003). The pattern suggests that any generated code must be separated from manually written code to avoid the customized code being overwritten when the developers must re-generate the code. The generation gap pattern solves this problem by inheriting the generated code and perform customization in the child classes. When mixing the manually written code and the generated code in the same classes cannot be avoided, the

users can use annotation “@generated NOT” to avoid the custom code being overwritten as depicted in Table 6.

When the developers plan to extend the prototype using other programming languages or when distributed components are required, the generated code can be run as an independent Java application that is accessible through the output components e.g., REST, SOAP, LinkSmart or MQTT.

Prototype deployment and distribution



Figure 74. Deployment artifacts for redistributing the prototype on another hardware platform.

Enabling developers to build and deploy the prototype easily, the generated software artifacts come with a build script which can be used for packaging and redistributing the application. In addition, it includes a configuration file which stores the configuration of the components such as network configurations. It also generates a web-based user interface which can be used to test the application (depicted in Figure 75) and a log file that keeps track the flow of the software modules this helps developers to identify any problem at runtime.

For testing the prototype, the web page shows the properties of the static objects and visualizes the numeric properties in a time series graph which could help identify the property values which are out of ordinary rapidly. In the case of a moving object, every time a new ID is detected, the webpage shows them as a new entry with a barcode representing the ID. Figure 75 shows two QR codes of two objects that had been identified by an RFID reader when they entered a manufacturing cell. When the first object is identified by the RFID reader, it becomes the active Moving Object in the station, and the relevant sensor values are correlated to that object. When the second object is detected by the RFID reader, the first object is not anymore the active object and therefore the relevant sensor readings are correlated to the second object. The dynamic correlation of sensor readings and objects in the environment could also be seen in different cases such as the occupancy of a room by people who enter and leave from time to time as discussed in section 5.1.5.3.

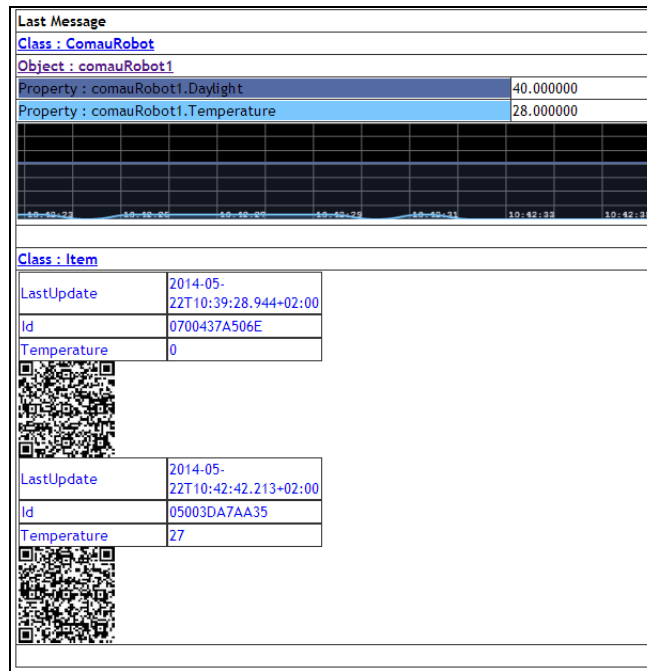


Figure 75. An example of the Web output which could be used for debugging the application.

After the code is generated, and the necessary modification to the code has been done, the developers could generate a deployable package by running the provided ant script. The ANT script will generate an executable JAR with the required runtime libraries. To deploy the application on another computing platform, assuming that Java standard edition could run on it, the developers only need to change the configurations relevant to the targeted hardware such as the serial ports and network addresses. Then they need to copy these artifacts to the targeted hardware. During the development, this work has evaluated this approach by deploying the artifacts on a Raspberry Pi board with a Java installed on it.

5.2 Conclusion

This chapter presents the conceptual design and implementation of IoTLink. IoTLink aims at enabling inexperienced developers building IoT application prototypes rapidly. To support inexperienced developers, IoTLink combines the strength of MDD and FBP allowing inexperienced developers to create virtual objects for representing the physical objects that are the concern of the problem domain.

In summary, the major contribution of IoTLink includes:

- Architecture for abstracting sensors and actuators within the domain objects
- Conceptual design for model driven tools in IoT development
- Polyglot communication architecture including support for well-known IoT communication protocols.

- IoT modeling language to map sensors and actuators into virtual objects
- Model-based sensor fusion framework to process IoT data

The architecture proposed by this work decouples IoT specific technologies with the domain concepts. The architecture consists of five layers. The first layer deals with abstracting heterogeneous data sources that are able to provide the states of the virtual objects. The connections are designed to communicate with well-known IoT technology. For instance, SOAP-based Web Services, RESTful services, MQTT, as well as domain specific technologies that such as OPC that is used frequently as a middleware for building and industrial automation, and Arduino serial communication that is often used for IoT hardware prototyping. This layer pulls and receives data from the data sources and pushes them to the other components at different layers in the architecture.

The second layer deals with extracting the actual state of the virtual objects from the incoming data. This involves data aggregation and fusing sensor readings from several types of sensors. IoTLink employs configurable arithmetic modules and Complex Event Processing engine which can be configured through Event Query Language to recognize event patterns and perform aggregation functions. IoTLink provides a model-based framework to separate the concerns between each processing module. Each specific module can then be combined as a network of processes which are able to work in parallel.

The third layer is dealing with the domain model definition which represents the physical objects relevant to the problem domain. This layer defines how sensors and actuators should be abstracted. IoTLink provides a domain specific modeling language which is platform-independent to define virtual objects and link them to the sensors and actuators. The model can be transformed into a Platform Specific Model, which in this case is Java. The Java code can be extended by more experienced developers in a further phase of the development. The generated codes may also run as a standalone application that can be run and act as proxies for the physical objects that can be accessed by external applications from the network.

The fourth layer provides methods to deliver the states of the virtual objects to the application logic. For the first case where application logic is implemented within the same application, IoTLink allows developers to define the logic such as rules that are fed to Drools rule engine. This extends the flexibility of maintaining the application logic and allows rules to be updated even at runtime. For the second case where the application logic is implemented in external applications, IoTLink must map the state of the physical objects onto the different data format and communication protocols (e.g., database entries, XML and JSON that can be accessed from a RESTful service). The polyglot architecture employed by IoTLink, which supports different communication protocols and data format in the first and the fourth layer enables it to act as translator between different IoT components and achieve higher interoperability. The fifth layer illustrates the applications accessing the state of the virtual objects which can be done through a network or within the same application.

In practice, IoT development could be using services that are available and shared locally as well as on the internet. To use these services, the developers must first be able to discover them and select the devices that fulfil their requirements. This dissertation anticipates this need by discussing the discovery of devices in Chapter 6.

Chapter 6.

Sharing and Discovering IoT Semantically

Discovering IoT resources such as devices and smart objects is quite essential for the IoT scenario to enable sharing resources between several applications. Sharing IoT resources sometimes is required firstly to decrease the overall cost of the systems. Secondly, it is necessary to use a single device that affects the same environment in order to prevent conflicting decisions (e.g., a switch for the lighting in the room). In these scenarios, the device developers must be able to share the devices while the application developers must be able to find the adequate devices. In the IoT context, where millions of connected devices are predicted, finding an adequate device cannot be done manually. Therefore, semantic discovery that allows users to search devices based on their semantic properties have gained a lot of interest from the research community.

This dissertation discusses the current semantic discovery using semantic sensor web approaches such as Semantic Sensor Network (SSN) (Compton et al., 2012) and proposes an improvement by using relations between words in linguistic to semi-automatically organize devices in an ontology. The development and part of the evaluation has been done in the context of a master thesis under the writer's advises (Avila, 2013) therefore part of this section has been published in (Avila, 2013; Pramudianto et al., 2014). In the context of this dissertation, the result of the master thesis is extended with the integration of SSN ontology which is a widely accepted ontology for describing sensor observation and the use of WordNet, which contains more complex linguistic relations such as hyponym, hypernym, troponym (explained in detail in section 6.3.1.1).

6.1 Device Discovery

There exist several device discovery approaches in the local area network and mobile network that could enable the IoT discovery for instance, the implementations of the IETF Zeroconf (Williams, 2003) such as UPnP (Presser et al., 2008) and Apple's Bonjour⁶⁴ have

⁶⁴ <https://developer.apple.com/bonjour/>

revolutionized the way consumer electronic devices and applications discover each other and interact. However, these approaches are limited for the local area network scenario. In a broader context, a recent survey has revealed that discovering services over the internet has been investigated (Outay et al., 2007). These approaches have even been furthered with semantic technology in a smaller application domain. For instance, OpenGC, an organization founded for promoting geospatial interoperability, has defined a common sensor ontology schema which can be used to describe sensor deployments (Botts & Robin, 2007). This enables the applications to discover suitable sensors based on their semantic properties.

Although, device discovery has been investigated extensively in local area network, the IoT presents more challenges such as a larger scale of networks and a broader diversity in terms of technology and application domains. In this regards, it is irrational to assume that standardized and common terminologies will be used to describe and search for devices, particularly when considering that the devices could be shared by different service providers in all over the world. The terminology diversity complicates the discovery of suitable devices by the applications. Therefore, this work proposes to solve the terms diversity by taking advantage of online dictionaries such as Big Huge Thesaurus⁶⁵.

As part of the IoT prototyping framework proposed by this dissertation, a semantic discovery component is designed and developed. The design process follows the UCD approach that collects requirements from several user workshops, as described in Chapter 5. Having analyzed the users' requirements, it becomes clear that the device discovery must consider the developers that are not familiar with ontology. However, it must be able to facilitate discovery at the semantic level.

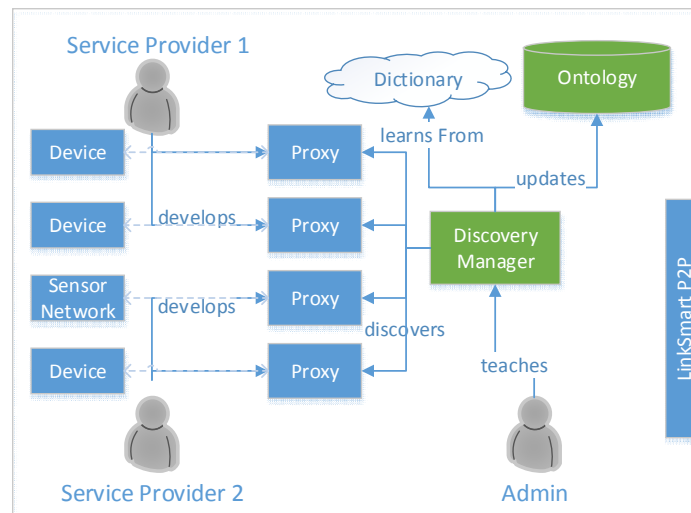


Figure 76. Architecture of the semantic Discovery Manager.

Addressing these requirements, a discovery broker named Discovery Manager was implemented as illustrated in Figure 76. The Discovery Manager should be deployed in each local network. Each Discovery Manager is responsible for discovering local devices at the

⁶⁵ <http://words.bighugelabs.com/> (Retrieved on August 5, 2014)

network level and maintain the information in the knowledge base. When the devices have left the network, it should be reflected in the knowledge base. The Discovery Manager should also expose the devices' information and capabilities to the other the Discovery Managers that are connected to it. This approach allows all Discovery Managers that are connected to have the same knowledge about the whole devices in the network. However, synchronizing devices between Discovery Managers is beyond the scope of this work and should be investigated as a future work of this dissertation.

On the local network, various ways can be used for the interaction between the Device Proxy and the Discovery Manager. In the first implementation, the author investigates WS-Discovery (Modi & Kemp, 2009) which is predicted by many works as the future discovery protocol for devices in Service Oriented Architecture environments (Zeeb et al., 2007). The semantic descriptions of the devices were embedded in the device proxies which then must be transferred to the Discovery Manager. Enabling the transfer, the device proxies expose a Web Service to get the device descriptions.

When the device proxies they became active, they announce themselves through a UDP announcement of WS-Discovery as well as responding to a WS-Discovery probe messages that are sent by the Discovery Manager. As illustrated in Figure 77, WS-Discovery supports ad-hoc and managed modes. Within an ad-hoc mode, there is no central server and therefore sending a "HELLO" and "PROBE" messages must be done through multicast. In a managed mode, a centralized server called discovery proxy is responsible to respond to the "PROBE" messages on behalf of the targeted services.

Theoretically, the Discovery Manager could listen to "HELLO" and "BYE" messages sent by the device proxies. However, these messages are not guaranteed since they are sent through UDP. And the initial test proved that sometimes these messages get lost, particularly when the network is congested with a significant amount of traffic. This of course varies depending on the network capacity, such as bandwidth and signal to noise ratio of the physical channel. Since the WS-Discovery does not offer a reliable messaging, as a work around the Discovery Manager must send a PROBE message every interval of time to trigger the announcements of the available Device Proxy. This approach, unfortunately, sends a lot of unnecessary traffic to the network.

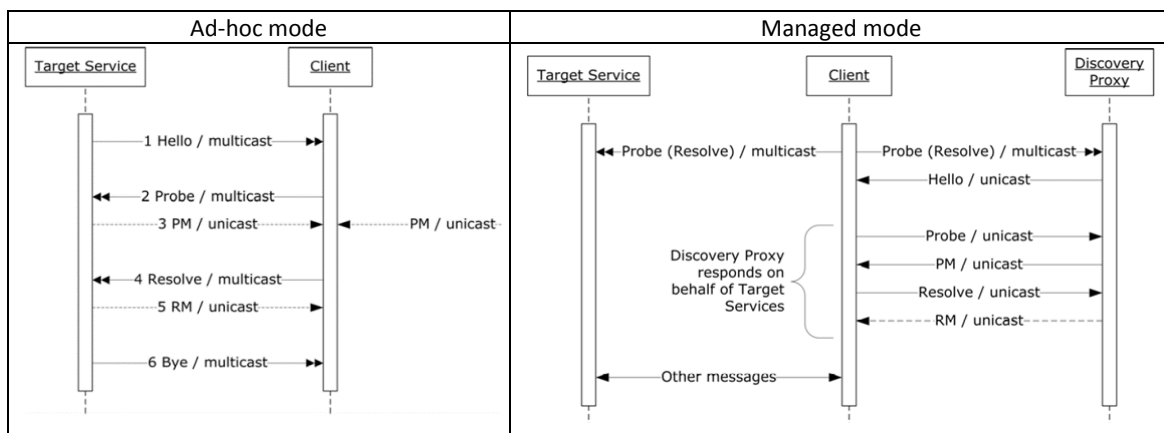


Figure 77. Interaction between target service and client in two different modes of WS-Discovery(Modi & Kemp, 2009)

The HALLO messages sent by the Device Proxies contain the address of the service that provides the device description. This enables the Discovery Manager to consume them without having to know their addresses in advance. Once the service is consumed, the Discovery Manager receives the semantic description in a JSON formatted string that contains the semantic information of the device such as device Id, types, capabilities, and information how to consume the device's services.

The second approach reverses the interaction of the first approach. It requires the Device Proxy registering themselves to the Discovery Manager upon activation. This approach has the advantage that Device Proxies do not need to offer Web Services which could be too resource demanding for resource-constrained devices. In the opposite, this approach requires that the Discovery Manager offers a Web Service allowing the Device Proxy to send the device description (e.g., `registerDevice (String description)`) and deregister itself (e.g., `deregisterDevice (String id, int expiredInSecond)`). In addition, to enable devices finding the Discovery Manager without knowing its address in advance, the Discovery Manager must respond to WS-Discovery's PROBE messages sent by the Device Proxies. The advantage of this approach is that it does not flood the multicast group by sending PROBE messages periodically. However, it is hard for the Discovery Manager to guarantee that the devices in the knowledge base is always available since there might be devices that left the network without deregistering themselves. To overcome this problem, the device manager applies a periodic membership renewal system that requires the device proxies to send a keep-alive message with the expected expiry time in order to renew their membership. The Discovery Manager deletes the devices in the knowledge base that have passed their deadline without renewing the membership.

6.2 Semantic Device Description

Once the semantic description has been extracted, they are registered in the Discovery Manager's knowledge base which is stored in an ontology. Before storing the knowledge about a newly discovered device, the Discovery Manager uses an online dictionary to provide a best effort approach in organizing the new device under the same or synonymous device categories that already exist in the ontology. First, it tries to find the device category as published by the device in its ontology. When a category has not yet existed, it accesses an online dictionary to find out whether a category with a synonymous term exists in the ontology. When a synonymous category exists in the ontology, it adds the new devices under this category and adds a new term in the synonym set (synset) of the term. This approach is suggested by Tsai et al. (Tsai et al., 2011) for handling terminology diversity of picture tags. When the application developers or the applications themselves search for a device with particular semantic attributes, the Discovery Manager looks up the given terms in the main device list as well as in the synsets.

Relying on online dictionaries for solving the diversity of terminology, however, does not always result in the term mapping that is desired since the developers also use terminologies that are not officially synonymous, but used interchangeably in specific application domains. Thus, the Discovery Manager must anticipate that a human intervention is needed from time to time for handling these exceptions. An admin role is defined to modify the ontology

mappings when the dictionaries do not provide the desired results i.e. not able to identify the terms that should be used interchangeably or it assumes that the terms are synonymous although in a specific context they are not used interchangeably.

On the device integration, the device developers must develop proxies representing their physical objects such as a single physical device or a network of sensors and actuators. The proxies must carry a metadata in a JSON formatted string describing the physical objects that it represents (depicted in Figure 78). The developers must define several attributes such as the device's unique id, type of the device, and its capability. For sensor devices, the metadata must also contain the unit of measurement. These fields can be extended by adding additional fields in the JSON file. The description must be made available by the proxy through a Web Service that fulfills a particular contract. Once the Discovery Manager retrieves the device description successfully, it parses the information, cross checks the terminology and adds them to the ontology.

```
{
  "id": "177b60c0-f914-11e2-b778-0800200c9a66",
  "type": "Smart meter",
  "capability": "Power consumption",
  "unit": "Watt",
  "location": "Room 2.45",
  "company": "SmartMeter Co.",
  "model": "378492"
}
```

Figure 78. Device description that must be carried by the proxy of the device (Pramudianto et al., 2014).

6.2.1 Identification

There exist identification standards that have been proposed for the IoT e.g., Electronic Product Code (EPC) which is used for identification based on the RFID (Brock, 2001), UPC which is used for product identification based on the barcode (Consortium, 2005), universally unique identifier (UUID) which is used for identification of the software resources (Leach et al., 2005), OpenID which is used for providing a universal ID for users on the internet (Recordon & Reed, 2006), and IPv6 (Jara et al., 2012). Since there is not a standard that can offer an optimal solution for IoT diverse ids schemes will always be used within different applications. Therefore, this work does not limit the id to any Id standard and uses a string attribute to enable different ID schemes, being used by the developers during the implementation.

6.2.2 Sensors and Actuators Categorization

In the context of IoT, devices can be categorized into two major types, including sensors, which could provide contextual information about things, and actuators which could influence the state of things and the environment. Sensors and actuators could be further decomposed based on different criteria e.g., Fink (Fink, 2012) divides sensor types based several categories e.g., how the signal is monitored (e.g., optical, electronic), the polymers types (e.g., conjugated, conducting, electrostrictive, electrochromic), and the measurement types (e.g., humidity, biosensors, mechanical, electrochemical, piezoelectric). Another classification focuses on sensors that rely on magnetic including proximity, microphones, velocity hall effect, linear variable-differential to determine the object position (Brauer, 2006). Bishop

categorizes sensors based on their measurement capability, including linear and rotational sensors, acceleration sensors, force measurement sensors, torque and power measurement, flow measurement, temperature measurement, distance measuring and proximity sensors, light detection, image and vision systems, integrated microsensors and vision sensors (Bishop, 2007).

Actuators could also be categorized according different criteria. Actuators produce energy that could influence its environment or any specific entities connected to the actuators. Casier et al. categorize actuators based on the form of energy that they produce, which, between others, include force and pressure, speed and acceleration, temperature, gas composition, electromagnetic fields, light (Casier et al., 2008). Bishop discusses types of actuators based on the input and output energy which can be categorized as follows (Bishop, 2007):

- Electrical actuators, such as diodes, thyristor, bipolar transistor, triacs, diacs, power MOSFET, solid state relay, etc.
- Electromechanical actuators, which subdivide in direct current motor, alternate current motor and stepper motor.
- Electromagnetic actuators, such as solenoid-type devices, electromagnets, relay, hydraulic and pneumatic, cylinder, hydraulic motor, air motor, valves, etc.
- Smart material actuators, such as piezoelectric, electrostrictive, magnetostrictive, shape memory alloy, electrorheological fluids, ultrasonic piezo motor, etc.
- Micro- and nanoactuators, such as micromotors, MEMS thin film optical switches, MEMS mirror deflectors, MEMS fluidic pumps and valves, NEMS drug dispenses, etc.

Brauer categorizes magnetic based actuators into (Brauer, 2006) :

- Electrohydraulic valves in airplanes, tractors, automobiles, and other mobile or stationary equipment.
- Fuel injectors in engines of automobiles, trucks, and locomotives.
- Biomedical prosthesis devices for artificial hearts, limbs, ears, and other organs.
- Head positioners for computer disk drives.
- Loudspeakers.
- Contactors, circuit breakers, and relays to control electric motors and other equipment.
- Switchgear and relays for electric power transmission and distribution

Another work categorizes actuators to two types of actuators including linear and rotary. "Rotary actuators, also called torque motors or torque actuators are electromechanical devices that develop torque with limited-angular travel. Linear actuators are force motors that develop force with limited linear travel" (Pawlak, 2006).

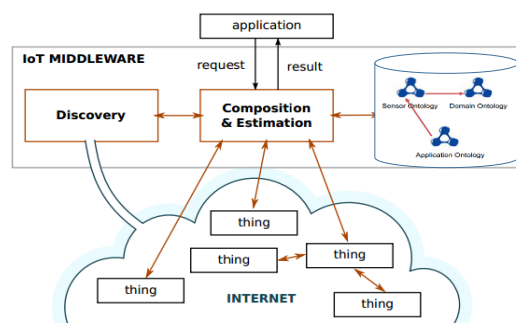


Figure 79. The role of ontologies within an IoT middleware adapted from (Hachem et al., 2011)

Avila decomposes the device classification further to include display devices, and communication devices (Avila, 2013). However, this dissertation argues that they could be categorized under the sensor and actuator categories depending on the context e.g., when communication the device receives signals, they act as sensors and as they send the signals they act as actuators to the communication medium. Display device also acts as sensors as they receive the signal to display the content, and act as an actuator as they influence the display to show the content.

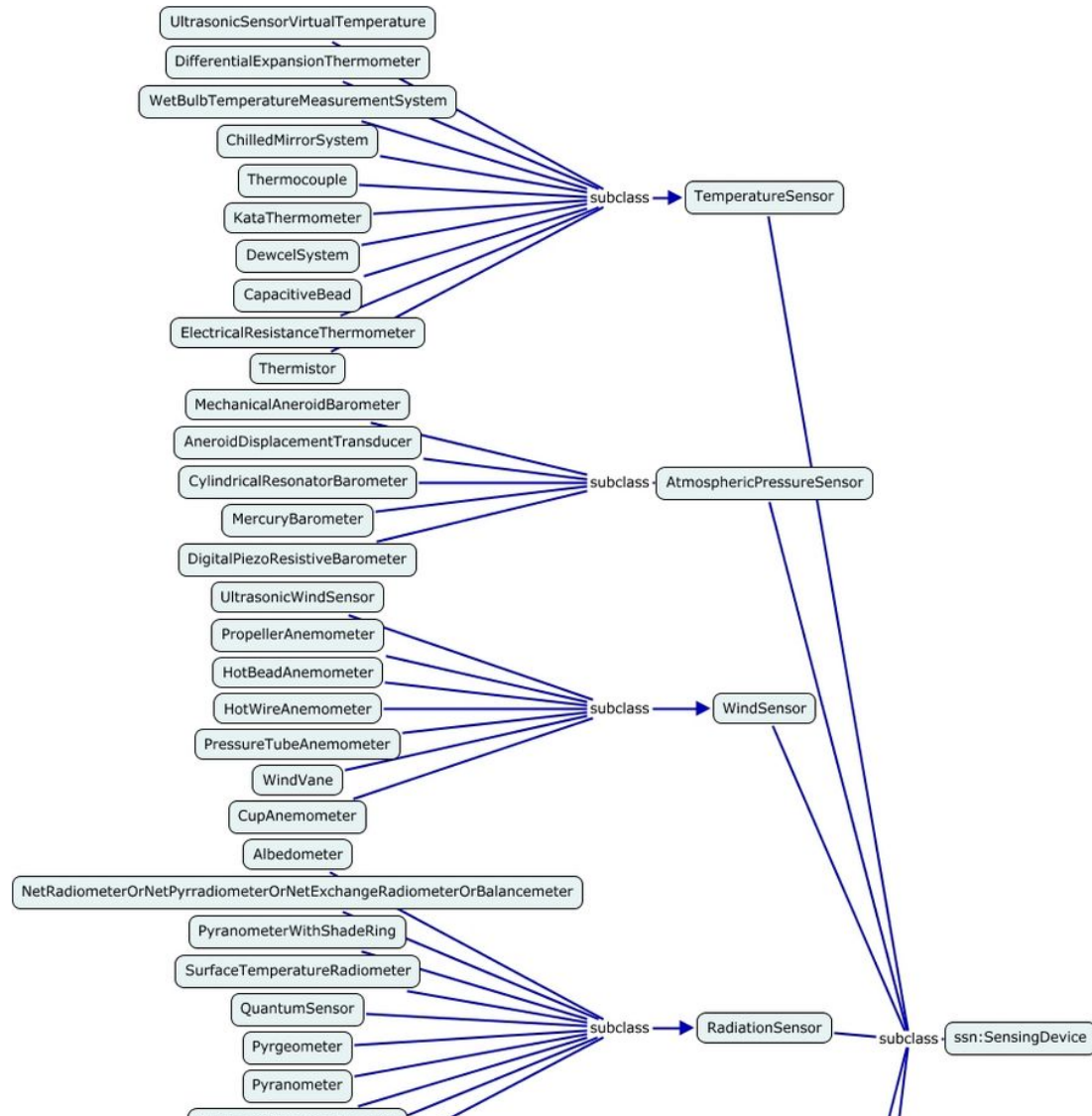


Figure 80. Partial illustration of the Automatic Weather Station ontology (Barnaghi et al., 2011).

Another extensive sensor taxonomy was done in an ontology to enable metadata exchange between automatic weather stations (Barnaghi et al., 2011). The ontology includes a taxonomy of sensors used in the agriculture meteorology. On the first level they categorize

the sensors based on the physical phenomenon to be measured, e.g., Radiation, Temperature, Atmospheric pressure. In the second level, the sensors are categorized based on the technique how they measure the physical phenomenon, e.g., temperature measurements can be done by thermistor, electrical resistance thermometer, and capacitive bead.

Different ways of organizing sensors and actuators shows that in different situations and application domains, some criteria are more useful for the application domain. There is no “correct” way to organize them without considering the application requirements.

In addition, ontologies could be used to describe sensors and actuators with Meta information in different granularity from a very coarse to very detail granularity. Several ontological approaches for modeling devices have left out the categorization of the devices (Bandara et al., 2008; Togias et al., 2010) since they claimed that this level of categorization must be defined according to the applications by the subject-matter experts (W3C, 2011). However, some works involving concrete application implementations have included the categories of devices that are relevant for that application domain in an ontology (Kostelnik et al., 2008; Jingjing et al., 2009). These different approaches have shown a clear indication that a generic IoT ontology may be used across application domains, however, to be useful for the implementation they must be linked to more detailed ontologies that are more specific to the application domains as depicted in Figure 79. However the tradeoff between the efforts required to provide a very detailed information, the performance issue caused by processing a huge amount of information in the ontology, and the advantages that could be gained by adding more details in the ontology must be considered carefully.

The Semantic Sensor Network Ontology (SSN Ontology)

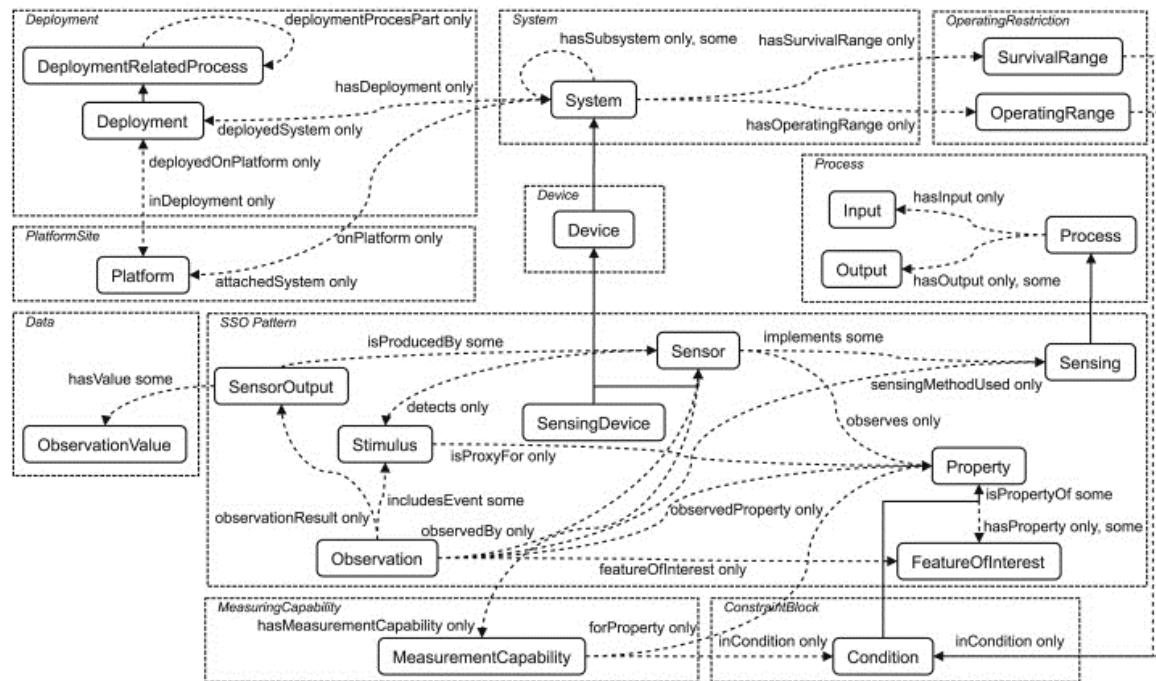


Figure 81. Overview of the SSN Ontology classes and properties (W3C, 2011)

SSN Ontology has been proposed by several organizations that are involved within the W3C Semantic Sensor Network Incubator Group (SSN XG) as a result of studies and merging several similar sensor ontologies such as CSIRO, OntoSensor, MMI, CESN and (OGC) Sensor Web Enablement (SWE) standards (Lefort et al., 2011).

The Semantic Sensor Network Ontology is a "formal OWL DL ontology for modeling sensor devices (and their capabilities), systems and processes" (W3C, 2011). It is able to describe the process of sensing and how sensors are deployed or attached to the platforms. It describes as well systems of sensors and sensing methods. The ontology "leaves the observed domain unspecified" (Lefort et al., 2011), but when it is instantiated, it allows domain semantics, units of measurement, time and time series, location ontologies and mobility ontologies to become attached to it.

As depicted in Figure 81, SSN Ontology consists of several modules to describe an observation system, including Deployment, System, Operating Restriction, Process, Device, Platform Site, Data, Skeleton, Measuring Capability, and Constraint Block. Each module has properties and classes to represent particular aspects of a sensor or its observations (W3C, 2011).

6.2.2.1 Capability and Unit of measurement

The device capability denotes the gain that can be obtained from the applications by using the device. In the sensor and actuator context, a more concrete definition should answer the question of "what kind of data and information that the sensor could deliver" and "what kind of influence / effect that the actuator could do to its environment". Sensor capabilities might include the physical or virtual phenomenon that is observed by the sensors. For instance, in a building automation domain, the ability to measure physical phenomenon such as e.g., "power consumption", "humidity" or "temperature", "occupancy" could be described as the capability of the sensor". In the SSN Ontology, this can be mapped to the relationship between Sensor to Stimulus and Sensor to Properties, which belong to a FeatureOfInterest. Similarly, the actuator capabilities in building automation domain, include "power switch (on/off) electronic devices", "open/close door, window, window's blind", "increase/decrease temperature, fan, escalators". There is no existing equivalent of actuator capability in the SSN ontology since, it only covers concepts related to sensors. In section 6.2.3, this work proposes the extension of the SSN ontology for dealing with actuators

When a device is a sensor, it is necessary to describe the unit of measurement so that the applications know the meaning of the data it receives. Moreover, when the conversion is required ontology could provide the information to perform the conversion. Unit of measurement itself presents a broad topic that has been captured in several ontologies such as Measurement Unit Ontology (MUO) (Berrueta et al., 2008), Quantities, Units, Dimensions, Values (QUDV) (OMG, 2009), Quantities, Units, Dimensions and Data Types in OWL and XML (QUDT) (Hodgson & Keller, 2011), These ontologies do not contain instances of units but they provide schemas that could be used to describe the unit in standardized vocabularies. A more comprehensive ontology for mathematic and physic containing units of measurement has been presented in (Gruber & Olsen, 1994).

The quality of measurement

Similar to the device categorization, it is practically impossible to limit the vocabularies for describing the capability of the device. Since the required granularity to describe device capabilities depends on the context of use and applications. For instance, for some applications, it is quite sufficient to describe the capability of a thermometer is to measure the temperature. However, in another context the quality of the measurement might be essential for the application to calculate the trustworthiness of the data. In this case a detailed information about the capabilities could include the quality parameters such as the accuracy, measurement frequency, the lifetime of the device (e.g., able to operate for a year), sensitivity range (e.g., between -35 to 150 degree Celsius), and relevant observation (e.g., the temperature of room XYZ). In the SSN ontology, these quality parameters are referred as the MeasurementCapability. The MeasurementCapability concept provides an abstraction for a collection of measurement properties that represents a single quality parameter of a sensor measurement. The provided measurement properties are shown in Figure 82, which include between others drift, sensitivity, selectivity, accuracy, precision and latency.

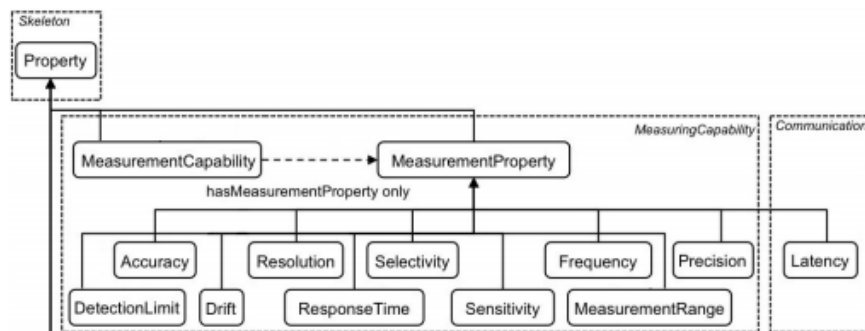


Figure 82. Enumeration of Measurement Properties in the SSN Ontology (W3C, 2011)

6.2.3 The design of the Discovery Manager knowledge base.

The main advantage of using ontologies to store device information is the possibility to derive a logical deduction based on incomplete facts. This feature allows the applications to retrieve the necessary information about devices, although they are not explicitly described by the device developers. Although there exist diverse ontology reasoners, most reasoners support RDF/RDFS and OWL reasoning, which, between others, include reasoning over transitivity, reflexivity, disjointness, and equivalent. Table 7 shows examples of logical deduction over the transitive relations between RDF triples.

In Table 7, the first device has a type of “kWh-Meter”. Assuming that in the ontology there is already a statement stating that Power-Meter is a subclass of EnergySensor, the reasoner could derive that PowerX123 has also a type of EnergySensor. In the second example, let’s assume that in the ontology ElectricityConsumption and PowerConsumption are marked as identical, the reasoner could derive that EltakoXX01 is able to detect PowerConsumption. In the third example, the capability of the third device could be derived from its type by defining a customized rule that is used by the reasoner. In the example, PlugwiseXX01 is able to detect

PowerConsumption since its type (Smart-Plug) is able to detect it. Alternatively, this deduction could be achieved through SPARQL queries.

Table 7. Transitive inference of device types

No	Device Description	Explicit axiom
1	Id="PowerX123" Type="Power Meter" SensingCapability="Electricity"	:PowerX123 rdf:type :Power-Meter :Power-Meter rdfs:SubClassOf :EnergySensor →→:PowerX123 rdf:type :EnergySensor
2	Id="EltakoXX01" Type="KWh-Meter" SensingCapability="Electricity"	:EltakoXX01 rdf:type :KWh-Meter :ElectricityConsumption owl:sameAs :PowerConsumption →→:EltakoXX01 ssn:detects :PowerConsumption
3	Id="PlugwiseXX01" Type="Smart-Plug"	:PlugwiseXX01 ssn:detects :Electricity :EltakoXX01 rdf:type :Smart-Plug :Smart-Plug :detects :PowerConsumption →→:PlugwiseXX01 :detects :PowerConsumption (custom rule)

Because of these features, to maintain the knowledge about the available devices in the network, the Discovery Manager uses an ontology as its knowledge base. The initial design of the ontology was very simple and can be extended at runtime to describe device metadata with diverse numbers of attributes. The ontology captures the main IoT concepts such as sensors and actuators or any other aggregated devices. Since the Discovery Manager might be used in different application domain, it needs to be able to work with numerous methods of categorizing devices.

The initial scheme includes several basic properties that require each device to have, e.g., device id, type, location, capability, and optionally a link to another device (Figure 5). In the initial version, the devices are categorized based on these attributes in a flat structure. While flat structure is quite simple to understand and implement, this approach has a drawback which eliminates the possibility of having a device taxonomy. Organizing the devices in a taxonomy allows the Discovery Manager to estimate the semantic similarities between two devices based on the topology e.g., edge based approach (Pekar & Staab, 2002; Cheng et al., 2004) and node based approach (Resnik, 1995; Couto & Silva, 2011).

Given the different terms could be used by the device developers to describe the device types and capabilities, the Discovery Manager must be able to detect when identical devices are described with diverse terms and organize these instances under the same device type. Enabling this feature each time a device is discovered, the Discovery Manager must first check if the device type and capability of the device are available in the ontology. When it could not find a device type or capability with the given names, it retrieves the synonyms of the given names from an online Bighugelabs dictionary and check again if their synonyms are already available in the ontology. When they are available, the Discovery Manager classifies the new device type and capabilities under that synonym set. Additionally, it stores the initial terms given by the device developer as a synonym of the existing device type and capabilities in their synonym set (synsets). Although the Bighugelabs dictionary is able to provide an extensive list of synonyms, it assumes that these terms have the same semantics. This is not always the case when in linguistic, some words could have super-subset relations e.g.:

Bighugelabs assumes that “Physical Property” and “Light” are synonymous, however “Light” is a subset of “Physical Property”. These relations are known as hyponymy and hypernymy. These cases are discussed more detail in section 6.3.1.1 which describes the second iteration of the Discovery Manager.

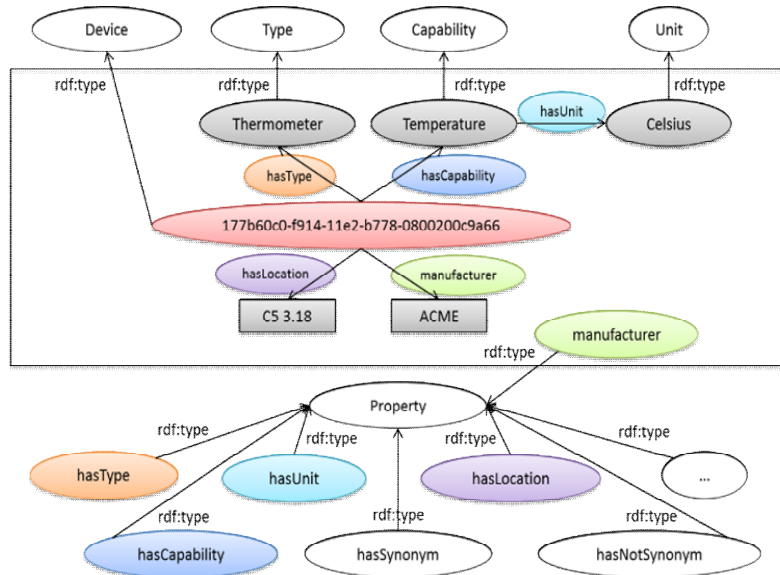


Figure 5. The ontology used for maintaining the knowledge of the available devices (Pramudianto et al., 2014).

The design of the ontology also allows devices to be linked to another device which can be used to model a device that observes a property of another device. The properties “linkToID” and “linkToCapability” can be used to link sensor devices to the objects that are being measured by the sensors. For example, a printer can be linked to a smart-plug that measures its power consumption. Since the order when the connected devices become available cannot be guaranteed, the link between two devices must be stored as two independent string literals (Figure 83) which can exist in the ontology independently. If this link was made as ontology edge, the system must guarantee that the two instances are always available at the same time.

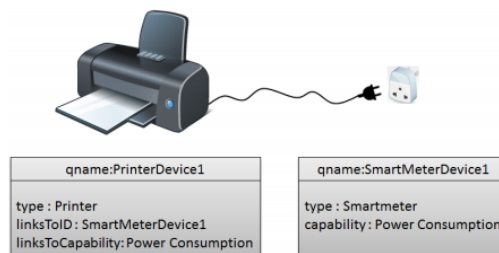


Figure 83. Linking devices in the ontology (Pramudianto et al., 2014).

6.2.4 Adding arbitrary device attributes at runtime

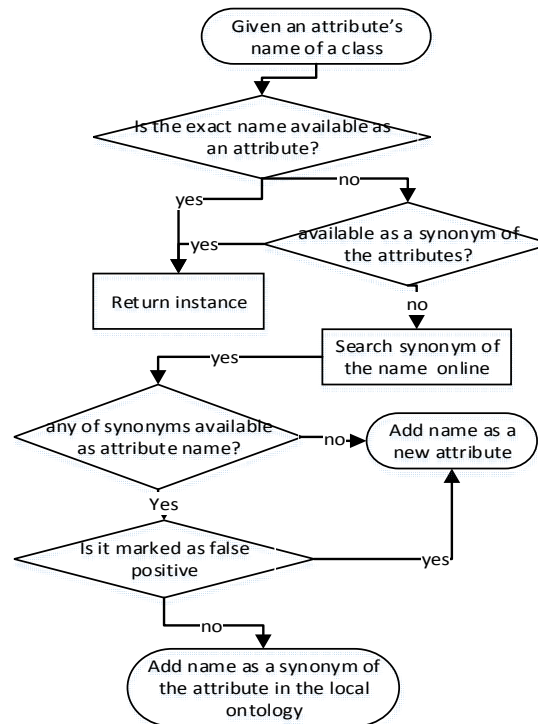


Figure 84. The logic used to homogenize terminology used for describing devices (Pramudianto et al., 2014).

Since all possible device attributes may not have been known from the beginning, there are two ways to resolve this. First the arbitrary attributes that are not yet available can be stored as string literals in the ontology. Second the base ontology can be extended to store them as classes which can be instantiated when other devices with the same attributes become available.

Given that different terminologies may be used to describe equivalent device attributes, the Discovery Manager should check if the semantics of the attributes already exists in the ontology. This requires the Discovery Manager to homogenize the terms used by the metadata that are newly discovered with the terms available in the ontology. This homogenization is done with a best effort solution to associate together terms that have equivalent meanings according to online dictionaries. The detailed process of the terms homogenization is depicted in Figure 84.

To keep the record of terms that have already been recognized as synonyms by the online dictionary and to avoid subsequent repeated online requests, the Discovery Manager includes these found synonyms in local synonym sets (synsets) which are stored as axioms in the ontology. This enables the Discovery Manager to learn how different terms can be used to address equivalent devices every time new devices are connected. Each term in the local ontology that can be used to describe the metadata of a device will have its own synset. During the homogenization, these local synsets will always be checked first, and only if they do not return a match for a pair of terms, the Discovery Manager will proceed to query the

online dictionaries. However, the online dictionaries are not always aware of terms that can be used interchangeably in specific application domains, and they will fail to recognize certain terms as equivalent in this context. Conversely, they could recognize terms as synonyms while they are not equivalent in that specific context. In these cases, the Discovery Manager will require a human intervention (administrator) to correct these homogenization errors through a graphical user interface (GUI) shown in Figure 8.

In cases where there exist a controlled vocabulary that can support the homogenization of different terminologies, this system should allow an administrator to pre-load that vocabulary from a file to the ontology through the GUI. In doing so, when the term is searched locally for matches, it will likely be found, thus avoiding the need to search online dictionaries and to decrease the chances of errors in the homogenization.

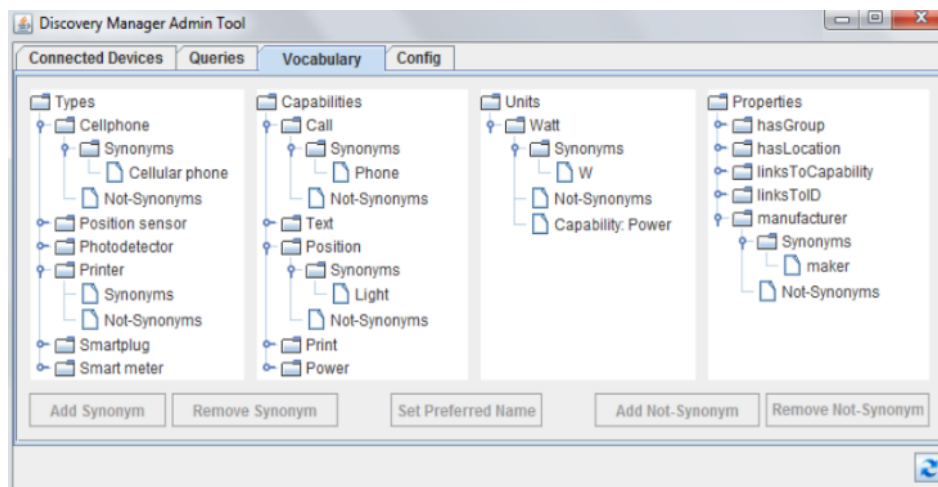


Figure 85. The administration tool to modify the synsets (Pramudianto et al., 2014).

6.2.5 Web Service interface for the applications

In order to allow applications to query data from each local ontology, the Discovery Manager offers a Web Service interface that includes all the relevant methods that applications require invoking. Given that application developers might also use different terminologies for the same domain of interest, the Discovery Managers should be flexible to homogenize as well terms coming from application queries, in order to provide query responses that are as accurate and complete as possible. When the system concludes that a given term used as an input parameter does not exist in the ontology, it tries to find the synonyms in the local synsets and if required from the online dictionaries.

The Web Service provides methods to retrieve all connected devices with the specified type, capability, location, link to other devices, and any properties that are defined by the device developers. In addition, application developers could also query the ontology using SPARQL. Upon the queries by the application, the Web Service methods return a list of devices with their properties.

6.3 Using SSN Ontology and WordNet

Although the design of the ontology in the first iteration is quite simple, some drawbacks were identified. First, it uses a non-standardized ontology schema. Consequently, to query any information from the ontology, the application developers must learn and understand the vocabularies and relations used in the ontology. Using a standardized ontology schema allows users who already familiar with it to query information without having to learn the available concepts and properties to be used in specifying the queries. This enables the application model and the generated knowledge to be exposed as a machine-readable linked data which can be automatically processed by third party applications (Atemezing et al., 2013). Moreover, using standardized vocabularies opens a bigger possibility to an adoption by the IoT community. Therefore, in the second iteration of the Discovery Manager's development, this work aimed at migrating the Discovery Manager's ontology to widely accepted ontology standard. As mentioned in section 6.2.2, SSN Ontology has emerged as a well-accepted standard for describing a sensor observation, and it is well maintained by the W3C Semantic Sensor Network Incubator Group.

The core of the SSN ontology follows a Stimulus-Sensor-Observation Pattern, which is introduced in (Janowicz & Compton, 2010). Figure 86 shows an example of using the pattern to describe an observation. In this example, ears act as a sensor that detects sound waves and produces an observation.

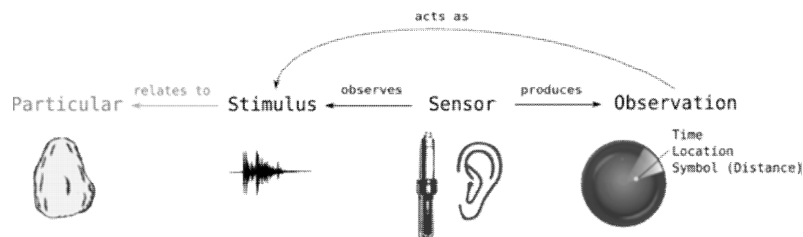


Figure 86. Stimulus-Sensor-Observation pattern (Stasch et al., 2009).

The SSN Ontology does this slightly different as depicted in

Figure 87, “Sensors” are linked to “Stimulus” through “detect” object property while the “Observation” is first coupled to “Property” that belongs to “FeatureOfInterests”. To describe a concrete sensor observation system, the Sensors must be linked to the Properties. The link between a sensor and a stimulus reflects the ability of the sensor to detect a physical event in general, while the link between the “Sensor” with a “Property” describes a concrete implementation of what the sensor is used for. This allows sensors to be re-purposed for a different kind of observations when it is possible. An example is illustrated in

Figure 88, a temperature sensor is able to detect temperature changes in the room. When it is deployed to provide an input for an HVAC system, it is linked with temperature property of the room, but it can be re-purposed to detect a possible fire event. In the latter case, the sensor is linked with the fire-state property of the room.

In contrast to the initial design the Discovery Manager’s ontology, the SSN Ontology could describe sensor observation more detail, however, it lacks concepts and relations that are able to describe an IoT system. As depicted in

Figure 87 on the right, the initial ontology is centered on a Device concept which can be categorized according to the Types related through synonym sets. A finer categorization exists in the SSN ontology through the taxonomy of subclasses, e.g., (ssn:Sensor is a subclass of ssn:Device). The SSN Ontology approach allows the application to take advantage of transitivity reasoning through the device taxonomy.

Unlike the Discovery Manager’s initial ontology, the SSN ontology does not have a generic Capability concept since it assumes that the sensor capability is always detecting stimuli. However, an IoT system may consist of sensors and actuators or any aggregation of them to form more complex functions. Sensor capabilities could be described with a “ssn:detects” relation between Sensors and Stimuli. The Stimuli are proxies for the Properties belong to a FeatureOfInterest under the observation.

For describing an actuation, it must be extended with an “Actuator” concept and a relation to the objects which could be influenced by the actuators. The simple solution is to follow a similar pattern used to describe the observation. As illustrated in

Figure 87, first an actuator is linked to a stimulus by an object property “canAffect” to describe the ability of the actuator in general. Secondly, to describe a concrete system implementation, the sensor is linked to a property. This enables the application developers to describe when an actuator could affect several objects of interest, e.g., a centralized air conditioner that affects all rooms in a building. In addition, to describe the relation with the applications using these devices, an “Application” concept is added which is linked with “Observation” and “Actuator” through “performs” and “actuates” object properties respectively, which enables to supervise the usage of devices by third party applications.

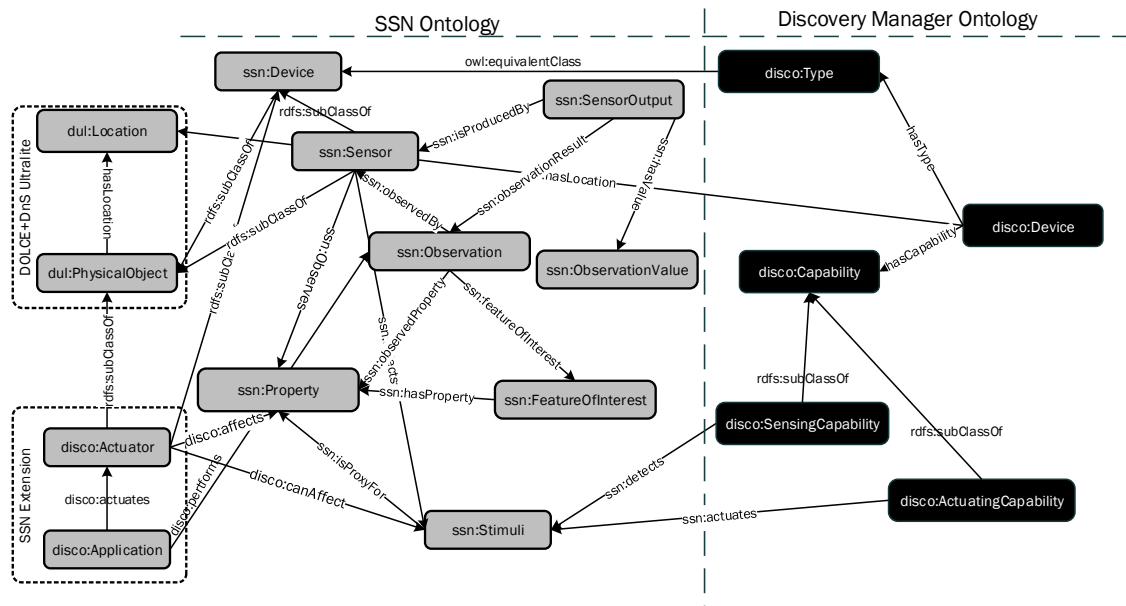


Figure 87. Ontology mapping between the SSN ontology & Base Ontology

Similar to other ontological approaches, SSN Ontology assumes that vocabularies used in an application domain must be standardized. As explained in the beginning of section 6.2, this work claims that IoT is a field which is used by many domains and therefore it is uncertain that vocabularies could be standardized at all and therefore a significant contribution of the Discovery Manager is its ability to handle the diversity of terms. When migrating to the SSN ontology, this feature must be implemented differently since SSN Ontology does not have concepts such as Synonym like the initial design.

In addition, some drawbacks are identified, e.g., the online dictionary (bighugelabs) that was used by the Discovery Manager only provides synonyms and antonym relations. Although, there are other relations in linguistic that we often find in the real world. For instance, hyponymy and hypernymy which denotes a subset/superset relations between words (e.g., tree is a hyponym of a plant) (G. A. Miller et al., 1990).

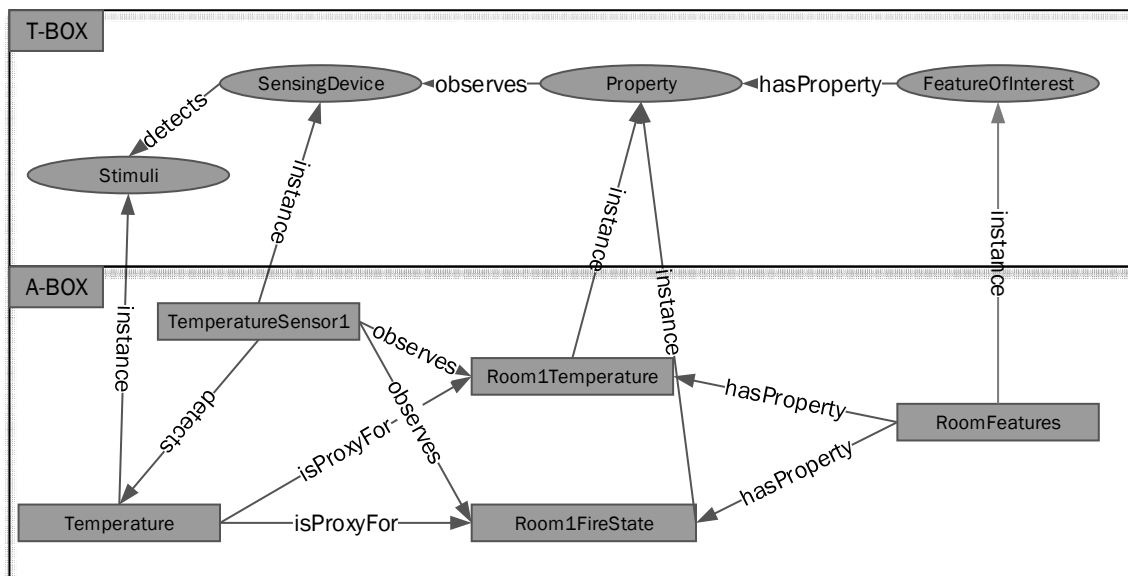


Figure 88. An example of instantiating SSN ontology for describing an Observation

In the second iteration, the implementation of this feature uses a more standardized vocabulary such as `owl:equivalentClass`, `owl:sameAs` to imply the synonymous classes and instances respectively. In addition, the Discovery Manager is able to estimate the taxonomy of device types based several criteria including the relations between lexical terms. A more detailed description of the implementation is discussed in the next section (6.3.1.1).

6.3.1.1 Estimating a taxonomy of device types

Having a device taxonomy enables application developers to find devices based on granular device types as well as their subclasses. However, this requires a lot of manual work to enter the relations between devices in the ontology. The device taxonomy could be built semi-automatically by applying rules, how the newly discovered device types must be positioned in the ontology. Many approaches have been discussed for building a taxonomy based on diverse criteria e.g., the relation instance-of relations based on Wikipedia categories and the pages, is-a relation based on the lexical head (e.g., British computer scientist is a scientist)

(Ponzetto & Strube, 2007), Lexico-syntactic based methods which identify subset-superset relation based patterns such as (Hearst, 1992) :

- *Such NP₀ as (NP₁) * (or | and) NP_n (e.g., such poultry as chicken or turkey)*
- *(NP_n) * or|and other NP₀ (e.g., chicken, turkey or other poultry)*
- *NP₀ including (NP_n) * or|and NP_n (mammals including pigs or monkeys)*

Another approach uses a statistical method such as latent semantic analysis (Landauer et al., 1998) to estimate the similarity between entities (Widdows, 2003).

To ensure that building the device taxonomy automatically would still be able to perform well when several devices are discovered at the same time, this work chooses to use simple rules based on the lexical analysis instead of using statistical techniques. The rules used by the Discovery Manager can be seen in the workflow illustrated in

Figure 89 which is explained as follows:

1. When a device type is not yet available, the Discovery Manager assumes that it has a “is-a” relationship with devices that have identical capabilities therefore it should be linked as an equivalence of that device.
2. When the capabilities of the new device are a subset of an existing device, the existing device should be linked as a subclass of the new device.

In addition, the following rules are applied to the device type and each stimulus that are linked to the capability of the device:

1. When the name of the device type/stimuli is a hyponym of an existing type/stimuli, it should be linked as a subclass of that existing type/stimuli.
2. When the name of the device type is a hypernym of an existing type/stimuli, the existing type/stimuli should be linked as a subclass of the new device type/stimuli.
3. When the name of the device type/stimuli is in the same synset as an existing type/stimuli, they should be linked as an equal.

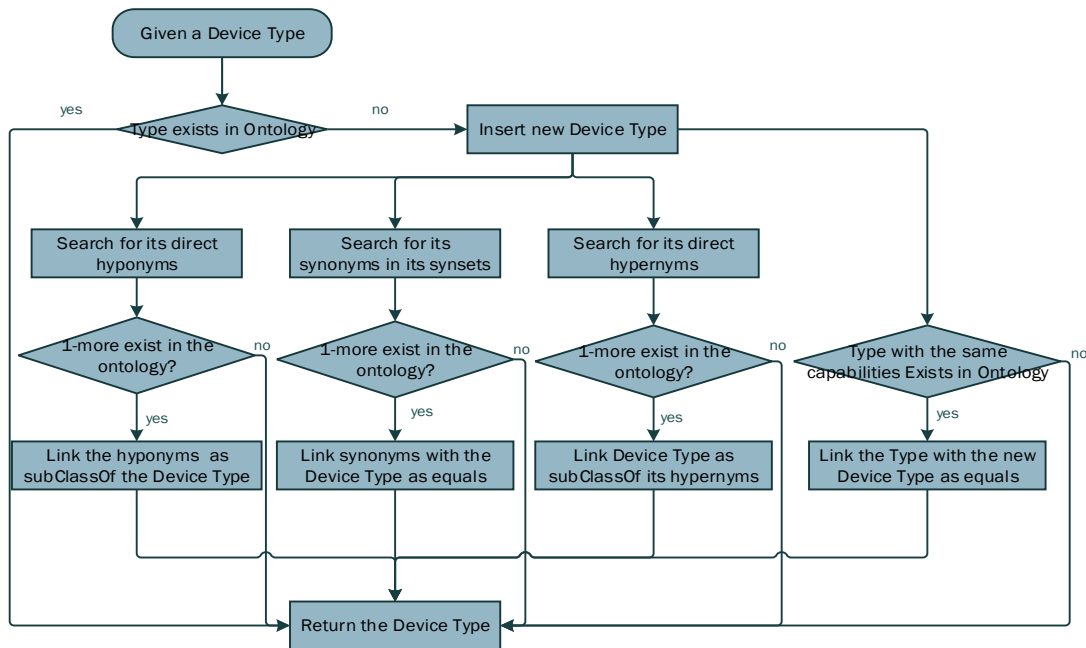


Figure 89. Workflow used to build the device type taxonomy.

The Discovery Manager requires a dictionary which contains relationships between English lexical. One of the open dictionaries which could provide such a complex structure between words is WordNet (Fellbaum, 1999). WordNet is available as RDF/OWL⁶⁶, offline dictionary, and prolog. The Discovery Manager uses the offline version to ensure its performance. The offline database could be updated when the new version of the database is made available by its maintainer.

WordNet was initiated in Princeton. It contains a lexical database of English containing Nouns, Verbs, Adverb, and Adjectives which are grouped into synonym sets (synsets) based on their senses or meanings (depicted in Figure 90). This is necessary since there are words that can be used interchangeably without changing any meaning. And there are words that look the same, but have different meanings which can only be recognized from the context of use (homographs). These synsets are linked with other synsets according to granular relations such hypernymy and hyponymy, meronymy (a part-whole such as a finger to the hand), troponymy (increasing activities such as move-jog-run). Hyponymy and hypernymy can be exploited by the Discovery Manager to provide an estimation of subclass and super-class relations between device types and their capabilities when new device types are discovered. Although, it provides the necessary structure, WordNet does not contain domain specific vocabularies and relations that allow the Discovery Manager to perform an accurate estimation of the device taxonomy. This is where a manual work by subject-matter experts are still required. Therefore, this work still provides the possibility for the Discovery Manager to work with a domain specific dictionary.

⁶⁶ <http://www.w3.org/TR/wordnet-rdf/> (Retrieved on June 7, 2014)

<p>S: (n) light, <u>visible light</u>, <u>visible radiation</u> ((physics) electromagnetic radiation that can produce a visual sensation) <i>"the light was filtered through a soft glass window"</i></p> <p>S: (n) light, <u>light source</u> (any device serving as a source of illumination) <i>"he stopped the car and turned off the lights"</i></p> <p>S: (n) light (a particular perspective or aspect of a situation) <i>"although he saw it in a different light, he still did not understand"</i></p> <p>S: (n) <u>luminosity</u>, <u>brightness</u>, <u>brightness level</u>, <u>luminance</u>, <u>luminousness</u>, light (the quality of being luminous; emitting or reflecting light) <i>"its luminosity is measured relative to that of our sun"</i></p> <ul style="list-style-type: none"> ◦ <u>direct hyponym / full hyponym</u> <ul style="list-style-type: none"> • S: (n) <u>illumination</u>, <u>illumination</u> (the luminous flux incident on a unit area) • S: (n) <u>incandescence</u> (light from heat) • S: (n) <u>luminescence</u>, <u>glow</u> (light from nonthermal sources) ◦ <u>attribute</u> <ul style="list-style-type: none"> • S: (adj) <u>bright</u> (emitting or reflecting light readily or in large amounts) <i>"the sun was bright and hot"; "a bright sunlit room"</i> • S: (adj) <u>dull</u> (emitting or reflecting very little light) <i>"a dull glow"; "dull silver badly in need of a polish"; "a dull sky"</i> ◦ <u>direct hypernym / inherited hypernym / sister term</u> <ul style="list-style-type: none"> • S: (n) <u>physical property</u> (any property used to characterize matter and energy and their interactions) ◦ <u>derivationally related form</u>
--

Figure 90. Examples of some synsets, hyponym, hypernym of "light" exposed by WordNet⁶⁷

To maximize the advantage of RDF/RDFS and OWL reasoners, the Discovery Manager must translate the lexical relations into relations that can be understood by RDF/RDFS and OWL reasoners. Secondly, Since the higher the number of axioms in the ontology, the more time required by the reasoners to deduct a conclusive result, the Discovery Manager only inserts relations between the terms that already available in the knowledge base for the new device type and its capabilities. The Discovery Manager translates these lexical relations as an RDF triples i.e. when the available device types exist in the same synset as the new device type they are inserted as owl:equivalenceClass, and for the capabilities are inserted as owl:sameAs. When the available Device Types exist as hyponyms of the new Device Type they are inserted as rdfs:subClassOf.

6.3.1.2 Estimating a taxonomy of device capabilities

The device metadata may include the device capabilities which are divided into sensing capabilities and actuating capabilities. Sensing capabilities correspond to physical events that the device could sense. In the SSN ontology, these concepts are represented by instances or subclass of "ssn:stimuli" linked to the sensor instance by a "ssn:detects" object property. Since the stimuli could be expressed with any arbitrary vocabularies, the Discovery Manager must again verify if the newly discovered stimuli or its synonyms already exist in the ontology. If they do, they are linked with "owl:sameAs" relations.

Figure 87, actuation capabilities are represented by the actuator linked to stimuli by an object property derived from "disco:canAffect" property. In this case, the object property denotes the effects that the actuator causes to the stimuli which semantically has a broader scope than the

⁶⁷ Taken from the result of querying "light" in <http://wordnetweb.princeton.edu> on June 11, 2014.

relation between the sensor and stimuli (i.e. `ssn:detects`). e.g., an actuator that is being able to switch a light and dim the brightness level could be expressed as the following axiom:

- `disco:Eltako12 disco:switch disco:light`
- `disco:Eltako12 disco:dim disco:light`

This means that a diverse terminology may be used in both the property and stimuli which could have different semantic meanings. Once again here the Discovery Manager links the newly discovered stimuli with its synonyms that already exist in the ontology. However to estimate the taxonomy of the properties that link the actuator and the stimuli, it must be dealt slightly different from estimating the taxonomy of device capability since they are verbs. In WordNet, verbs that are a subset of other verbs are named troponyms. Troponymy relations between verbs indicates “a more precisely the manner of doing something by replacing a verb of a more generalized meaning” (WordSense.eu, 2014). The Discovery Manager could take an advantage of these relations to build a taxonomy of the properties which allows applications to retrieve actuators that are able to fulfil their requirements but described using verbs with a broader sense e.g., an actuator described with “switch” capability should be able to do switch on and off therefore they must be discoverable by keywords such as “turn on”, “turn off”, “switch on”, and “switch off”. Moreover, property taxonomy allows applications to find actuators that could only fulfil partial requirements e.g., applications that require a device with a dimming capability could find a switch actuator that still could be used to perform its function partially which is dimming the light to 0% and 100%. This means in practice, when a software was designed to work with a dimmer is deployed in an environment that only has a switch, the software could still be able to run its function without needing to be reprogrammed and recompiled.

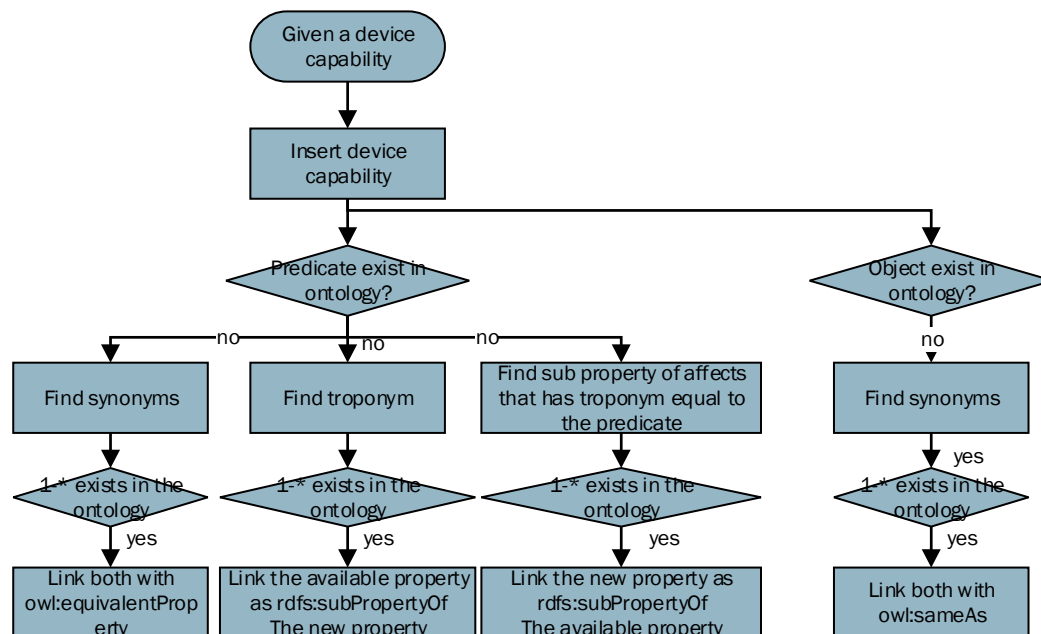


Figure 91. Inserting the capability of devices and linking to available concepts

To build the property taxonomy, after inserting the new properties, the Discovery Manager verifies the WordNet database if its synonyms are already available in the ontology. If they

are, they are linked together with the new property using owl:equivalentProperty relation which tells OWL reasoners to treat them as equals. Secondly, it checks if its troponyms are available in the ontology if they are available, they are linked to the new property with rdfs:subPropertyOf. Linking in the opposite direction is a little tricky, since WordNet does not have an opposite link for troponym, as a work around the Discovery Manager has to examine the troponyms of all available properties and check if the new property is a troponym of the available properties in the ontology. This approach might not scale well if the number of the available properties in the ontology is massive. As another approach, one could first calculate the distance between the available property with the new property in the WordNet that denotes the likelihood they are related (Pedersen et al., 2004; Yang & Powers, 2005). Alternatively, one could use normalized Google distance (Cilibrasi & Vitanyi, 2007) to measure the similarity between the properties.

6.3.1.3 Retrieving device information

Using standardized RDFS/OWL relations to build the device taxonomy enables application developers finding the shared devices without having to know the exact device types. Because the device types classified under the subclasses of the keyword used in the query, are automatically deducted by the ontology reasoners. For instance, if the application developers would like to retrieve all devices that consumes electricity in a home, assuming “Kitchen_Appliances” and “Home_Entertainment” are subclasses of “Electrical_Device”, the application is able to retrieve the devices under these two categories only by querying the instances of “Electrical_Device” with a SPARQL Query :

```
SELECT ?device WHERE{  
    ?device rdf:type :Electrical_Device .  
}
```

When the keywords used by the application developers to search for devices exist in the ontology, the Discovery Manager is able to retrieve the relevant device instances in the taxonomy. The ontology reasoner is able to retrieve the instances under the given terms and their synonyms since they have been marked as equivalence classes when the devices were discovered by the Discovery Manager as explained in 6.3.1.1

However, the Discovery Manager must consider that the application developers might use keywords which are not yet available in the ontology, but they are synonymous with a device type that exist in the ontology. In this case, the Discovery Manager must still be able to present the intended devices in the query result. This case requires the Discovery Manager to consult the synonyms of a given keyword to a dictionary (WordNet). The Discovery Manager first assumes that the given keyword exists in the ontology, if it could not find any result under the given keyword, it assumes that the given keyword is a synonym of an available device type, but it has not been inserted to the ontology since so far there has not been any device developer who describes the device type with the given keyword. Thus, it needs to retrieve all synonyms in the synset of the keyword from WordNet and query the ontology for any device type described by any of its synonyms.

The implementation of the Discovery Manager provides a Web Service with to retrieve device instances based on its type. First, when the application developers already know in advance

the terms used to describe the device type, they could search the device using the exact term used to describe them. When application developers do not know the exact term, they may use the synset address, which considers that a given term might only be a synonym of a device type in the ontology. Table 8 illustrates an example. Let a device instance with the “Light_sensor” type exists in the ontology. As other devices with “LightSensor” and “IlluminationSensor” are discovered, these types are marked as same types by the Discovery Manager since they are grouped in the same synset in the WordNet database. Using the REST link on the left of Table 8, the application developers would find these devices using the keyword “light” or “Illumination”, but when the application developers use “brightness” with the left link, it would not return any results. With the right link, the keyword “brightness” returns the same result as the keyword “light” or “Illumination”.

The application developer might also search devices based on their capabilities. A feature that has been provided by semantic discovery approaches to disambiguate the search result (Akkiraju et al., 2003; Kostelník et al., 2009). The Discovery Manager also provides a Web Service that could be used to find devices by their capabilities.

The application developers might use diverse keywords to express the intended stimuli. To be able to achieve the intended results regardless of the diverse keywords that may be used by the application developers, the Discovery Manager applies the same techniques as the one it uses to find devices by its type. When it could not find any result using the given keyword, the Discovery Manager has to verify whether the given keyword is a synonym of the available stimuli in the ontology by obtaining all synonyms of the keyword from WordNet database and use them to query the ontology.

Table 8. Example of searching device with keywords

Possible keywords : <i>light, illumination</i>	Possible keywords : <i>luminosity, brightness, brightness level, luminance, luminousness, light</i>
HTTP://129.26.162.117:9124/rest/device/device/light	HTTP://129.26.162.117:9124/rest/device/synset/brightness
<pre><SemanticDevice> <id> HTTP://www.linksmart.eu/ontologies /disco.owl#Photoresistor </id> <SensingCapabilities> LightIntensity </SensingCapabilities> <SensingCapabilities> Brightness </SensingCapabilities> <SensingCapabilities> Light </SensingCapabilities> <SensingCapabilities> Illumination </SensingCapabilities> <Type>Light_sensor</Type> <Type>Illumination_sensor</Type> <Type>LightSensor</Type> </SemanticDevice> <SemanticDevice> ... </SemanticDevice></pre>	<pre><SemanticDevice> <id> HTTP://www.linksmart.eu/ontologies /disco.owl#Photoresistor </id> <SensingCapabilities> LightIntensity </SensingCapabilities> <SensingCapabilities> Brightness </SensingCapabilities> <SensingCapabilities> Light </SensingCapabilities> <SensingCapabilities> Illumination </SensingCapabilities> <Type>Light_sensor</Type> <Type>Illumination_sensor</Type> <Type>LightSensor</Type> </SemanticDevice> <SemanticDevice> ... </SemanticDevice></pre>

The capability property might also be expressed with diverse terms that could be used interchangeably, such as the following query examples:

```
SELECT ?device WHERE ?device disco:switchOn disco:Light.
```

```
SELECT ?device WHERE ?device disco:turnOn disco:Light.
```

As these two queries are semantically equivalent, the Discovery Manager must be able to present the same results. Therefore, when a given keyword has not yet available in the ontology, the Discovery Manager has to consult to the WordNet database. Let assume if in the ontology, the available statement is as the following:

```
disco:device123 disco:switch disco:Light.
```

The Discovery Manager must be able to know that the instance “disco: device 123” fulfils the criteria of the device being searched by the application since its capability is a subset of a given keyword. Thus, when the Discovery Manager could not find the intended devices using the given keywords nor any combinations of their synonyms, it tries to find devices with a broader capability than what the applications requested by querying the ontology for devices that has a capability whose predicate is a troponym of the given keyword. The following shows the troponyms of “switch” as listed in the WordNet database.

```
switch
      shift, break, switch on, turn on,
      switch off, cut, turn off, turn out
```

6.4 Conclusion

This chapter describes the conceptual design and implementation of the IoTLink discovery component. The main contribution of this work includes:

- A discovery architecture that enables sharing IoT devices across the Internet.
- Algorithm that uses ontology and WordNet dictionary for discovering similar devices described with diverse terms.
- Algorithm to semi-automatically estimates the device taxonomy based on the lexical semantics.
- Increase interoperability by adopting SSN ontology and a proposal to extend it for describing IoT systems.

Discovery of IoT must be addressed at different levels. First, the syntactic discovery of devices in different network layer must be addressed. To ensure the user experience, there is a need of a holistic discovery from the lowest network layer to the application level that could work together seamlessly. However, IoT Discovery involves heterogeneous network protocols, and discovery mechanism, therefore IoT discovery should have an abstraction at the semantic level, which encapsulates different discovery technology at different layers and present the physical objects’ semantic properties such as functionalities and quality parameters. Semantic discovery has been discussed to discover services and devices (Akkiraju et al., 2003; Klusch et al., 2006; Mokhtar et al., 2008; R.-C. Wang et al., 2009). However,

none of the approaches anticipate diverse terminologies that might be adopted by different developers to describe the devices' capabilities and types. This presents a risk that the intended devices may not be discovered by other developers who use synonymous keywords to find them. Therefore, to solve this problem, this author proposes to use a dictionary to identify the terms diversity in device descriptions or the search terms. Initially, the Discovery Manager took an advantage of an online dictionary which was able to detect if synonymous terms are used to describe identical devices. In the second major iteration, the dictionary was changed to WordNet dictionary which offers more detailed relations between lexical terms such as hyponymic (relations between terms that has narrower meaning e.g., "dog" is a hyponym of "animal"), hypernymy (terms that have broader meaning e.g., "fruit" is a hypernym of "banana"), and troponym (verbs that signify increasing intensity of actions e.g., "walk" is a troponym of "run"). Having detailed relation between lexical terms allows the Discovery Manager to estimate the taxonomy of devices based on terms used to describe the device types and capabilities.

The discovery architecture requires the physical objects to be represented by proxies that carry their Metadata in a JSON formatted file. The proxies find the Discovery Manager through the WS-Discovery protocol. Then, they register themselves and send their metadata to the Discovery Manager. The Discovery Manager extracts the metadata and verifies whether the related terms already exist as device category and capabilities in the knowledge base in order to relate the existing terms with the terms used by the newly discovered device. From WordNet database, the Discovery Manager is able to recognize the following semantics:

- Synonymous terms and translate them as OWL equivalent relations.
- Hyponyms as subclasses of their corresponding Hypernyms
- Troponyms as subclasses of the verbs with more intensive actions, e.g., devices with the capability of "switch on" are subclasses of devices with the capability of "switch".

Although Dictionaries such as WordNet offer complex semantic relations between lexical terms, they do not always provide the semantic structure as needed by specific application domain. Within specific domains, terms which are not officially synonymous could be used interchangeably. Therefore, the Discovery Manager provides a user interface that could be used by an administrator to edit the relations between device categories and capabilities as well as adding domain specific relations between lexical terms that can be used to automatically determine the taxonomy of device category and capabilities.

Chapter 7.

Case Studies

7.1 Evaluation methodology

According to how software engineering experiments are carried out, they can be classified as follows (Kitchenham et al., 1995):

- Studies that focus on a single project, preferably called case studies since it does not resemble a formal experiment that requires replication.
- Studies that involve many projects or a single project which is replicated multiple times may be categorized as case studies or formal experiments depending whether the experimental subjects and objects are chosen randomly according to the constraints required by the experimental design.
- Studies that involve many teams and projects could be classified under a formal experiment or survey depending whether the selection of teams and projects follows the experimental design or it was decided incrementally depending on the result of the previous experiment.

As summarized in Table 9, each method has its own advantages and disadvantage as well as its main strength for achieving a specific goal.

Table 9. Summary of software engineering experiments classifications.

Method	Strength	Advantages	Disadvantages
Case study	Assess broad effects of a change / treatment.	Easier to plan. Reveals the effects on typical situations on a larger scale than formal experiments.	The results are harder to interpret / harder to pinpoint the cause. Results cannot be generalized.
Formal experiment	Choosing between several competing methods or tools	The results are easier to interpret and generalized. Allow to experiment on different possible situations based on small samples.	Must be done on a small scale to control the effect of specific variables. Difficult to conduct when the degree of control is limited.
Survey	Confirm the benefits of the change.	Capture the applicability to real-world projects with the strength of formal experiments which can generalize the results based on replications.	Done on a larger scale. Take a great deal of time to collect data. Results are not available after projects are completed.

These classifications are quite generic and can be applied to investigating different variables of interest in software engineering such as software quality, usability, development efforts, and maintainability.

After reviewing Table 9, to be able to accomplish comprehensive results that can provide answers to research questions #2 and #3, a combination of several approaches must be done. Thus, the author decided to perform the evaluation through two methods. First, the evaluation was performed through case studies that assess the practicability of IoTLink within typical IoT developments. Secondly, as described in Chapter 8, the evaluation was done through a set of controlled experiments measuring the usability of IoTLink against a typical IoT development approach which is done through middleware and textual programming languages such as Java.

7.2 Case studies

These case studies provide real-world use cases with a larger set of requirements and features than a controlled experiment. In the case studies, three prototype developments from the three European projects SEEMPubS, BEMOCOFRA, and ebbits are discussed. SEEMPubS presents a case study in integrating IoT for building automation and energy saving. BEMOCOFRA presents the second use case in integrating IoT for monitoring a prototype of a flexible manufacturing line. Ebbits presents a use case in integrating IoT for meat traceability.

7.2.1 SEEMPubS: Building automation

SEEMPubS is a European research project, which aims at reducing energy consumption in public spaces. The project tried increasing the users' awareness to their energy consumptions by providing a continuous feedback to encourage the occupants saving energy. The final prototype of the SEEMPubS improves the lighting and HVAC automation in the polytechnics of Turin by considering contextual information such as the room occupancy, daylight, and outdoor temperature. The campus consists of newly built buildings equipped with a BMS. It is able to control the lighting and HVAC centrally based on a schedule that is maintained by the facility manager. Additionally, the historical castle is used for the administration staff and some architecture classes. Since the castle is preserved by the government, it may not be physically modified to preserve its historical values. In this building, the existing HVAC and lighting systems are not well integrated into the building, and any additional equipment installation may not change the structure of the building, therefore, wireless solutions was the best option.

The prototype of SEEMPubS was developed for six rooms with different characteristics, including a pair of a single office, a pair of a shared office, and a pair of a shared lab room. The selection of pairwise was done to compare the different strategies applied to a room vs. the control subject (the room without the control strategy). The consumption data is stored in a database and can be accessed through desktop computer through a web portal as well as the smartphone of the occupants. To acquire the consumption data and enabling control to appliances by the occupants, SEEMPubS takes advantage of sensors and actuators that are built into the buildings. They are Fieldbus devices that are able to communicate through a bus

network called BACNET. In addition, EnOcean wireless sensors are used to measure the occupancy, daylight, temperature and indoor light. Since the built-in sensors and actuators did not provide sufficient measurement and control details, wireless plugs based on ZigBee that are able to measure energy consumption and switch each appliance were installed.

The main challenge in the SEEMPubS development was integrating heterogeneous devices with different communication protocols and constraints (E.g., wireless sensors cannot send data frequently to preserve the battery life). To integrate these devices, three types of gateways were installed, including ZigBee gateways, EnOcean gateways and an OPC server that allows Fieldbus devices to communicate with TCP/IP network. On the software level, the integration was done through LinkSmart middleware. Each communication protocol is represented by a software proxy that offer Web Services for retrieving sensor data and sending a control command. The proxies also publish the sensor data every interval of time when changes have occurred and the events are subscribed by the applications and transported by the LinkSmart event manager, which go through the LinkSmart P2P network as depicted in Figure 92.

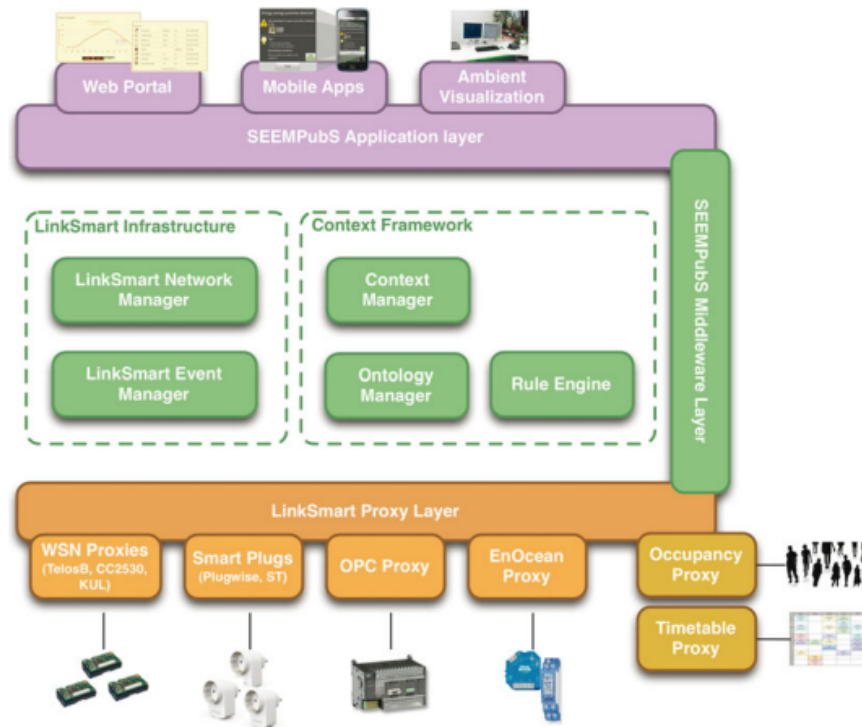


Figure 92. SEEMPubS Architecture (Osello et al., 2013)

Although applying the LinkSmart middleware were quite successful, the development of the device proxies and linking them to the applications required tremendous time and efforts since all components must be crafted manually by hand. According to the developers of the SEEMPubS prototype, the prototype development took approximately 10-12 months to achieve the initial version and another 6 months to perform live tests and fixes. The prototype was developed by six developers who worked approximately 50%-75% of their time on the project. Please note that these numbers are quite difficult to be quantified precisely since their

efforts were also invested for documenting the system, conducting research on new technologies, learning different new technologies, as well as the overhead required for context switching between tasks e.g., to recall what they did last time. It still provides an overview how much efforts are required to build such a prototype in a research project.

Experimenting with different ways of saving energy was quite difficult since the changing the system requires a significant effort. This has inspired the development of IoTLink, which is able partially to generate software artifacts required for research projects such as collecting sensor data, and connecting sensors actuators into application rapidly.

Studying SEEMPubS use cases provides valuable inputs for evaluating IoTLink. Therefore, this section discusses how the SEEMPubS system could be developed using IoTLink. This case study replicates integrating the rooms used in the project into a web application. In this case study, we only measure the energy consumptions of the lighting and the appliances using wireless power meters are installed in the sockets which are normally used by computers or floor lamps. In addition, the temperature is measured, and the control of the HVAC system is made available. For simulating the SEEMPubS system, a KEPWARE OPC server that is able to simulate the sensor data is used to generate the random energy consumptions of the ceiling lamps as well as data from the HVAC. The ZigBee and EnOcean data are generated by real devices that are installed in the office of the author. The efforts required to replicate the system was approximately 3 weeks of work done by two developers with 70% time for the project.

Modeling the system, a domain model to represent the offices and devices was designed. The domain model was designed to capture the energy consumptions on different granularities including the appliances and the total energy consumption of all appliances in the offices. The single office usually has a desk while the shared office has two desks and two ceiling lamps, and a lab is shared by eight students therefore have eight desks and two ceiling lamps. These details are captured in the domain model as illustrated by Figure 93. Then the classes are instantiated in the main canvas, and the properties are linked with OPC client, Plugwise, and EnOcean components which retrieve the sensor data from the gateways. When the new system is to be used, only the network addresses of the OPC server and other devices need to be configured with the addresses of the sensors and actuators in the polytechnic of Turin.

After the classes are defined, and the relations between classes are linked, the concrete objects were generated and linked to the corresponding sensors and actuators. The objects to represent offices were created using static objects as the sensors and actuators will have static relations to the physical objects in the room. Then the input components were added to the input container. The energy consumption of the ceiling lamps can be obtained from the OPC server, therefore, in the input container OPCInput components are added. The other appliances are connected to Plugwise power meters, which can be accessed by PlugwiseInput component. In addition, EnOceanInput was added to communicate with the other EnOcean devices. All power sensors are connected to a sensor fusion module that calculates the total energy consumption per room. In the output section, LinkSmartEventOutput are added so that external applications such as the Web Portal and Mobile application could subscribe to the events that they require. Moreover, several actuators are added and connected to “switch” function of the lamps. And finally, a Drools rule engine is added to the output container.

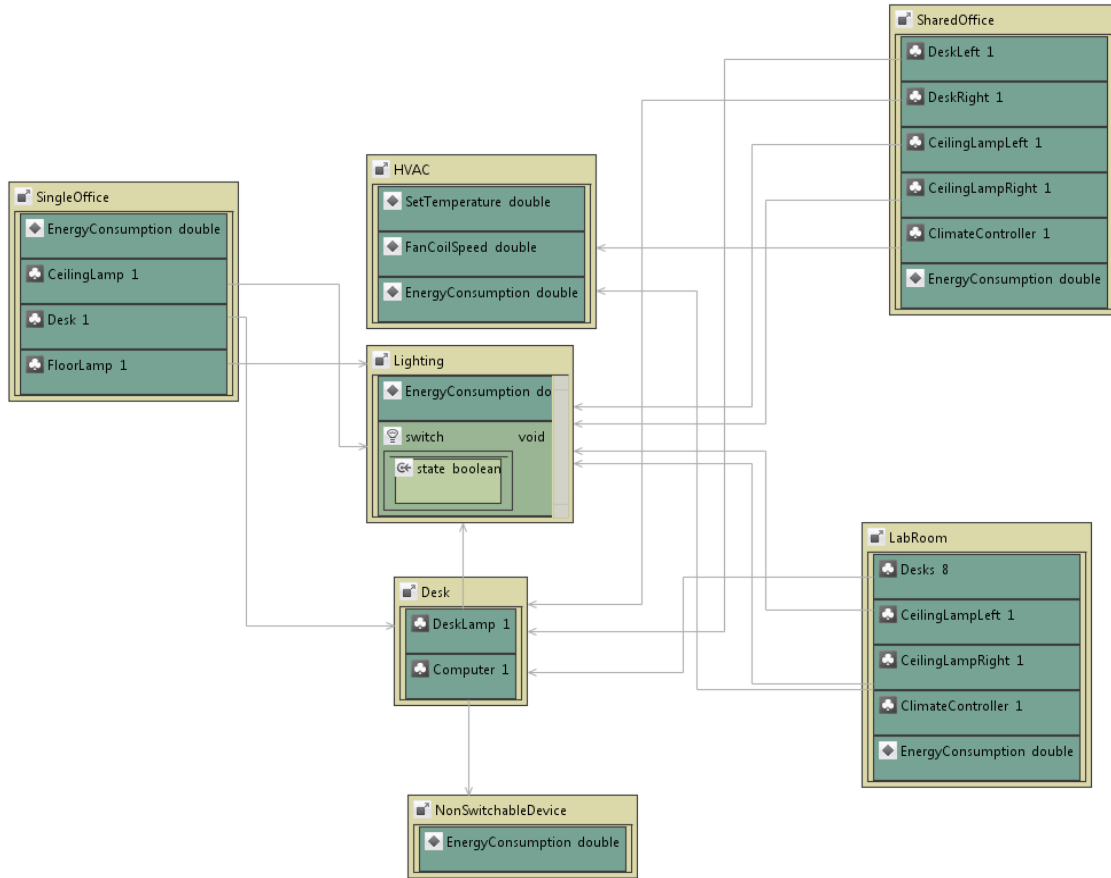


Figure 93. Domain model of the SEEMPubS system

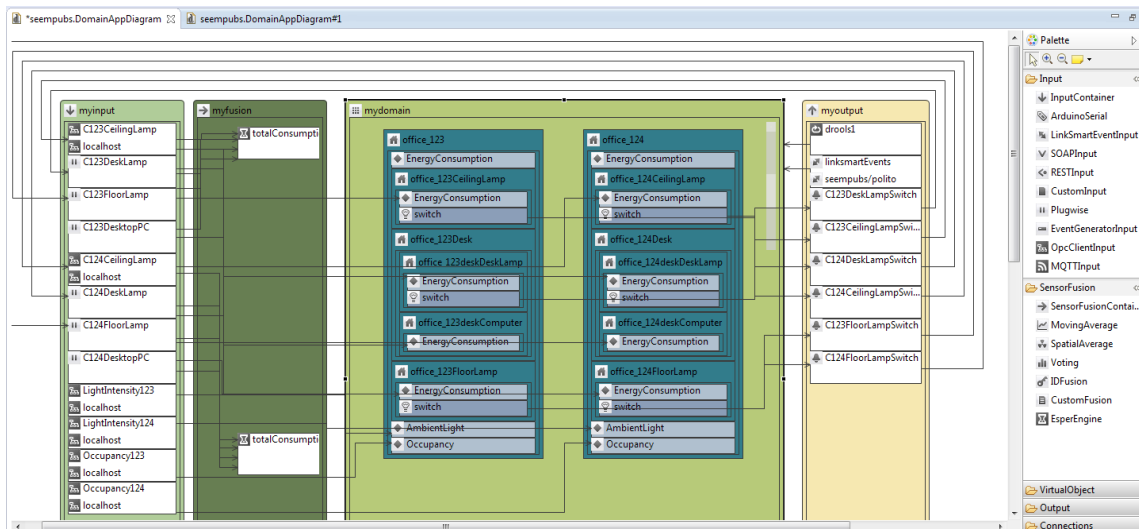
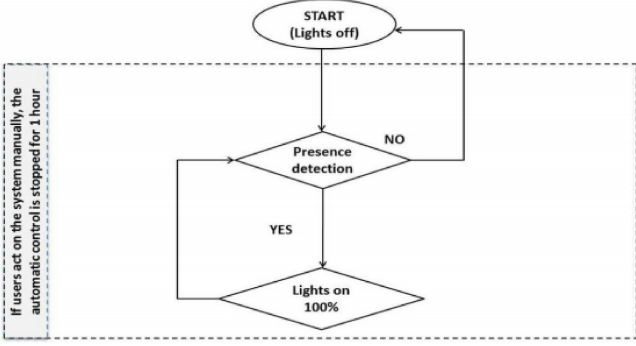
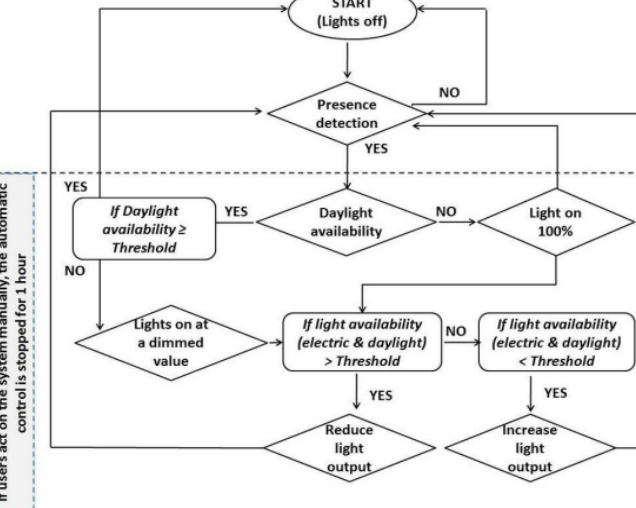


Figure 94. Linking the power meter that measure the energy consumptions and actuators in the two single offices

Table 10. Control strategy for non-dimming lighting

Flowchart control strategy (Osello et al., 2013)	Drools Rule
 <p style="text-align: center;">Non-Dimmable Lighting</p>	<pre> rule "Lighting On Strategy" when SingleOffice(occupancy == true && AmbientLight < 300, \$ceilingLight:CeilingLight) then \$ceilingLight.switc (true); end rule "Lighting Off Strategy" when SingleOffice (occupancy == false AmbientLight >= 350, \$ceilingLight:CeilingLight) then \$ceilingLight.switch (true); end </pre>
 <p style="text-align: center;">Dimmable Lighting</p>	<pre> rule "Lighting Up Strategy" when SingleOffice (occupancy == true && AmbientLight < 300, \$ceilingLight:CeilingLight) then \$ceilingLight.dimUp(50); end rule "Lighting Down Strategy" when SingleOffice(occupancy == true && AmbientLight >= 350, \$ceilingLight:CeilingLight) then \$ceilingLight.dimDown(50); end rule "Lighting Off Strategy" when SingleOffice(occupancy == false, \$ceilingLight:CeilingLight) then \$ceilingLight.switch(false); end rule "Lighting Off Strategy" when SingleOffice(occupancy == true, \$ceilingLight:CeilingLight) then \$ceilingLight.swith (true); end </pre>

The Drools rule engine allows the developers changing different control strategies at runtime which gives a great flexibility for a research project. The rules are maintained centrally through drools Guvnor, which allows developers to define new rules, activate and deactivate the rules, and apply versioning through a web interface. As the initial prototype, the lighting control strategies for single offices which are proposed in the SEEMPubS project were translated as Drools rules as depicted in Table 10. The control strategies for lighting in single offices consider the non-dimmable lighting system which can only be switched on and off and dimmable lighting system which should adjust the light intensity according to the amount of daylight or other light sources in the room. By changing the rules in the Guvnor, e.g., adjusting the threshold when the lights should be switched on and off, the developers could perform experiments how much energy could be saved while preserving the user comfort level. This would save the efforts of redeploying the software as required by the previous system.

7.2.2 ebbits: Enabling traceability in meat production

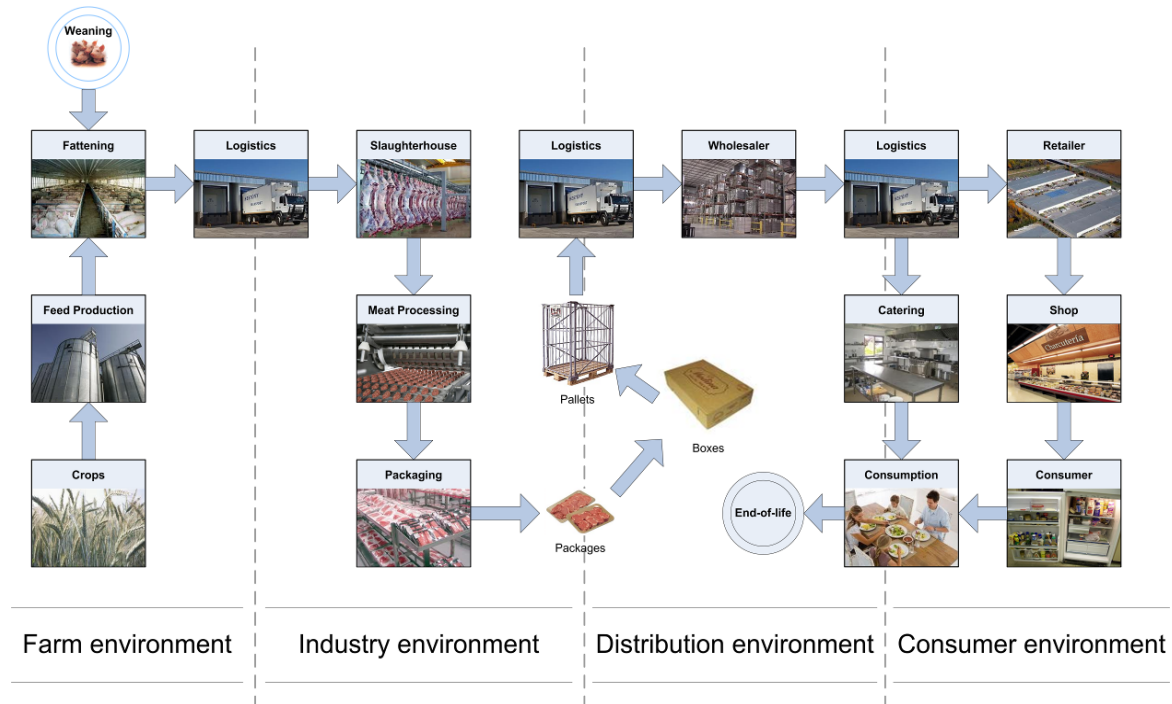


Figure 95. The lifecycle of pork meat product (Udsen et al., 2010)

In the ebbits project, several approaches to enable product traceability were investigated in order to improve transparency of the products for the customers. Moreover, traceability is useful when a product is found defected and need to be recalled. Having the possibility to trace the origin and distribution of the product electronically reduces the time required to isolate the potential problems and recall the products that are sold on the shelves. To enable a traceability along the supply chain network an uniform identification scheme must be used, and related product IDs must be linked (e.g., The id of the cow is linked to its meat). Furthermore, the production data must be collected, aggregated, and made available to the

parties that are authorized to retrieve the information about the products. The traceability scenario in ebbits describes the potential of IoT technologies to collect information from “farms to fork” and made them available for actors along the chain. Figure 95 shows an example of meat production processes in the pork meat industry. The production is started in a farm where pigs are born and raised with their mother until they reach a certain age where they are able to eat industrialized feed. The pigs are then moved to another location or farm, in which they are fed with livestock feed until they reach a certain weight.

When the pigs have reached a certain weight, they are sent to abattoirs. In the abattoirs, their information is recorded into the abattoirs’ ERP system. Once they are slaughtered, depending on the customer request, the carcasses are cut into smaller pieces which then are put into packages and labeled with barcodes. The packages are organized on pallets then delivered to wholesalers, retailers, or restaurants. A few supermarkets in Germany have provided an additional QR code that can be used by the customers to retrieve traceability information such as the location of the abattoirs. However, more detailed information that can be obtained by applying IoT technologies are not yet available such as how much carbon footprint is required to produce the meat, whether the pigs were fed with organic food, etc.

The pigs are not tagged with any RFID at the moment, since the technology is still too expensive in relation to the price of pork meat, some farms however use tattoo branding. Only the sows are tagged with RFID since the farms in Europe are required to know the ancestry information. The regulatory requirements differ from country to country. On the farm, the health and feed information about the pigs are recorded in a farm management system. However, these feeds and medical information is usually applied in batches. That means if a pig is ill; the whole pigs in the same pen will be treated with the same medication.

The main challenge of enabling traceability on a meat production chain is, the interoperability within the organization and between external organizations must be achieved. It requires the slaughterhouse, distributors, and retailers to be able to collect data internally from different subsystems, as well as tracking information along the chain to the farms. The farms must be able to trace the feed and medicine that have been given to the pigs to be able to localize the problem when it arises. Moreover, they need to keep a record of the distribution to be able to recall the affected meat.

Collecting and tracking these chain of information requires a seamless communication between devices and software in the local sites where information about the livestock or the meat is collected as well as seamless communication between organizations involved in the production chain. The main issue for enabling a seamless communication is the existence of heterogeneous systems within the organizations and between the organizations. There exist no well-established standard which regulates how these different information systems should share their data. On the farm level, there exist different suppliers of feeding systems, climate controls, and farm management, on modern slaughterhouses usually a more standardized industrial automation system connected to an ERP system is used to keep track the meat production. The data from the farms are often handed over to the logistic companies through forms and papers. This information is handed over to the slaughterhouses, then entered manually into their system. There is different information that could be included in the end product. For instance, since there is not a consistent standard of organic farming, the consumers might not know the quality of feed, whether antibiotic and hormone are used for

growing the livestock. The consumers who have more environmental conciseness may want to know the emission produced in transporting the meat. Moreover, the modern logistic vehicles are able to keep track the temperature of the meat during the transport to ensure that the cold chain is not violated, and the meat shelve life is consistent. When the animals arrived at the slaughterhouse, they are checked by veterinarians then slaughtered. The slaughterhouse usually assesses the meat quality in terms of fat contains and add this information for the retailers.

A proposed approach in ebbits (Brizzi et al., 2013) utilizes a centralized application which retrieves information from different stakeholders along the production chain. To enable the communication between the central application and various systems used by the stakeholders, bridging applications must be built which are responsible for collecting the input from each system using the required protocols and data format and transform this into a uniform protocol and data format which is able to be understood by the central application.

In ebbits project, a simulation of a beef production chain was implemented to demonstrate the proposed solution. In this demo, a beef production was chosen since most countries have required that cattle must be individually tagged, which makes the solution more realistic to be implemented. The flow of the data along the production chain is depicted in Figure 96. Along the chain, the cattle are represented by DigiCows which are software objects that are interlinked. When they have been slaughtered, they are represented by Bulk Beef that are linked with the DigiCow that represents its origin. In this demo, the cows were simulated using RFID tags which were scanned using RFID readers on each stakeholder. The data on the farm such as the weight of the cows and the amount of feed are simulated through physical sensors connected to Arduino boards.

The initial implementation done without specific tools except for Java and C# programming. The efforts required for the development were approximately 1.5 months of work done by two developers with 75% time for the project.

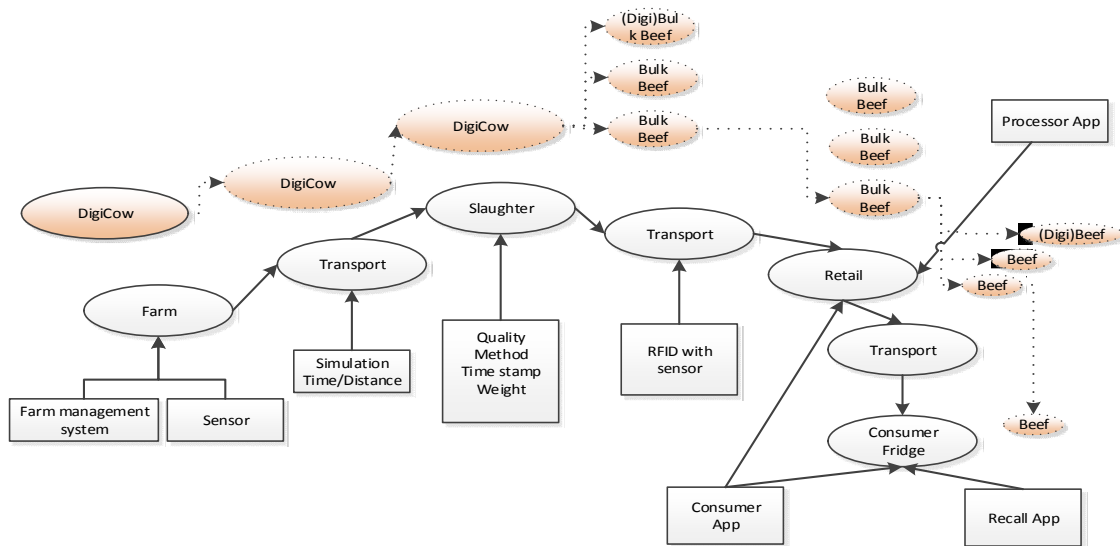


Figure 96. The data acquisition for enabling traceability in meat production through the ebbits platform (Madsen et al., 2013).

Replicating this development, IoTLink was used to capture the DigiCow and BulkBeef that are sent from one stakeholder to the next one and expose this information to the interested stakeholder in the production chain through different network protocols. On the farm, the each DigiCow must be correlated with the feed object that is consumed by the cattle. To log the feed consumed by the cattle, an RFID reader and two scales to weigh the feed and the cow could be installed. In ebbits, these devices were simulated using Arduino boards that are connected to an RFID reader and digital scales. When a cow approaches the feeding station, its tag, the feed id, the amount of the feed consumed by the cattle, the weight of the cattle are published through an MQTT broker through the Arduino proxies. In this case, the cow was simulated through RFID tags. The amount of feed was simulated through the data delivered from the scale connected to the Arduino. IoTLink was used to generate an application that captures these events and log them into a database.

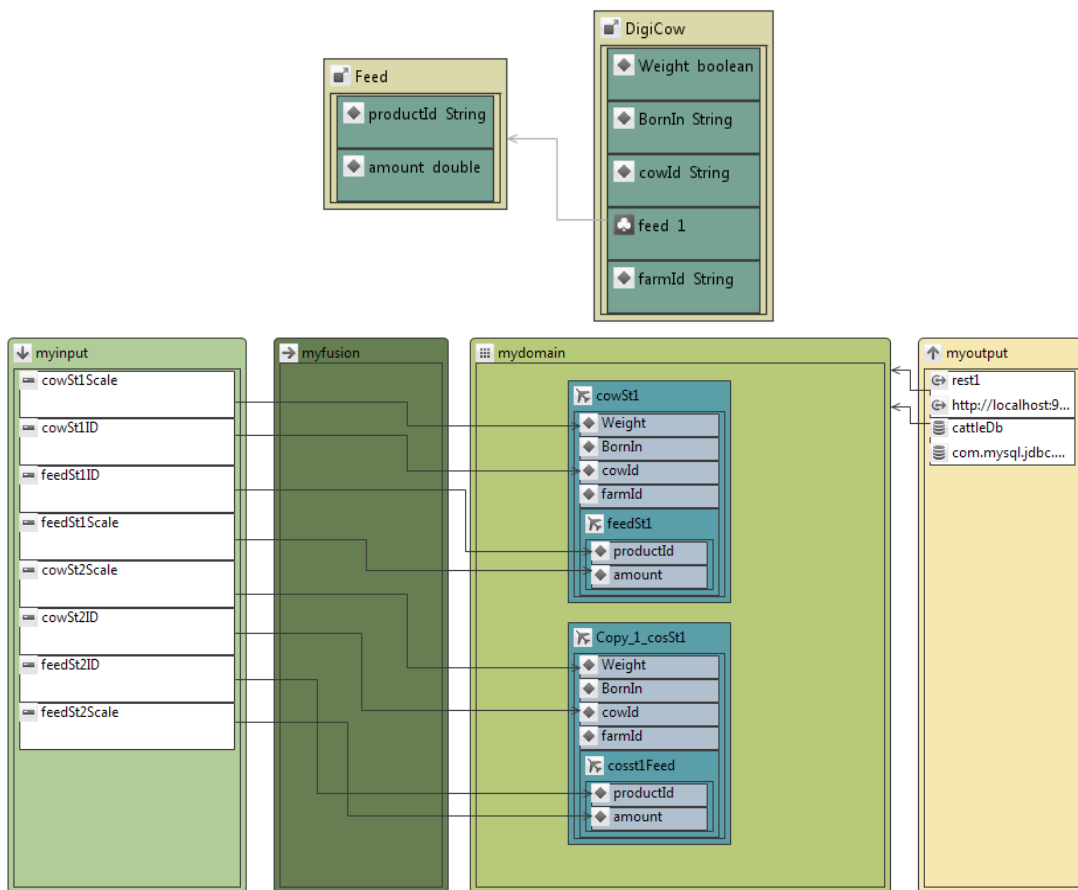


Figure 97. Domain model for recording livestock feed.

As illustrated in the Figure 97, to log the feeding events from all cows, the cows were considered as moving objects that have dynamic relations to the sensors attached to the feeding stations therefore we only need to have a DigiCow with the type of Moving Object at each feeding station. Since the prototype application only needs to log the feeding events, we do not need to model the stations.

Each time a cow feeds at the feeding station, the generated application retrieves the corresponding DigiCow from the VirtualObjectFactory. When the factory could not find the corresponding virtual object, it creates a new virtual object to represent the cow. However, when the object containing the id of the cow already exist, the factory retrieves that object and hand it over to the MainApp where its properties are updated with the data coming from the sensors and sensor fusion modules.

Using a simulated data, the feeding station data is published through an event generator. The DigiCow objects are stored in a memory as well as MySQL database. When the DigiCow objects are updated, e.g., their weight has increased, before they are updated, the history is stored in MySQL database. The DigiCow objects are published through a REST-based service which follows the structure of the domain model as depicted in Figure 98 on the left side. On the right side, the link to each DigiCow object is encoded in QRCode which can be used to tag the physical cow right before it is sent to the slaughterhouse.

The efforts required for the development with IoTLink was approximately 2.5 weeks of work done by two developers with 75% time for the project.

The screenshot shows a REST client interface. On the left, the XML response for a DigiCow object is displayed, including details like LastUpdate, BornIn, FarmId, Feed, Id, and Weight. On the right, a table titled 'Class : DigiCow' lists the same attributes for two different DigiCow objects, with a QR code generated for each object's ID.



Class : DigiCow	
LastUpdate	2014-07-22T13:35:36.273+02:00
Born In	Germany
FarmId	DE-122344
Feed	[object Object]
Id	X000162
Weight	89
	
LastUpdate	2014-07-22T13:35:40.291+02:00
Born In	Germany
FarmId	DE-122344
Feed	[object Object]
Id	X000163
Weight	97
	

Figure 98. DigiCow database which is accessible through REST (left) and generated link encoded in QRCode (right)

7.2.3 BEMOCOFRA: Monitoring a flexible manufacturing prototype

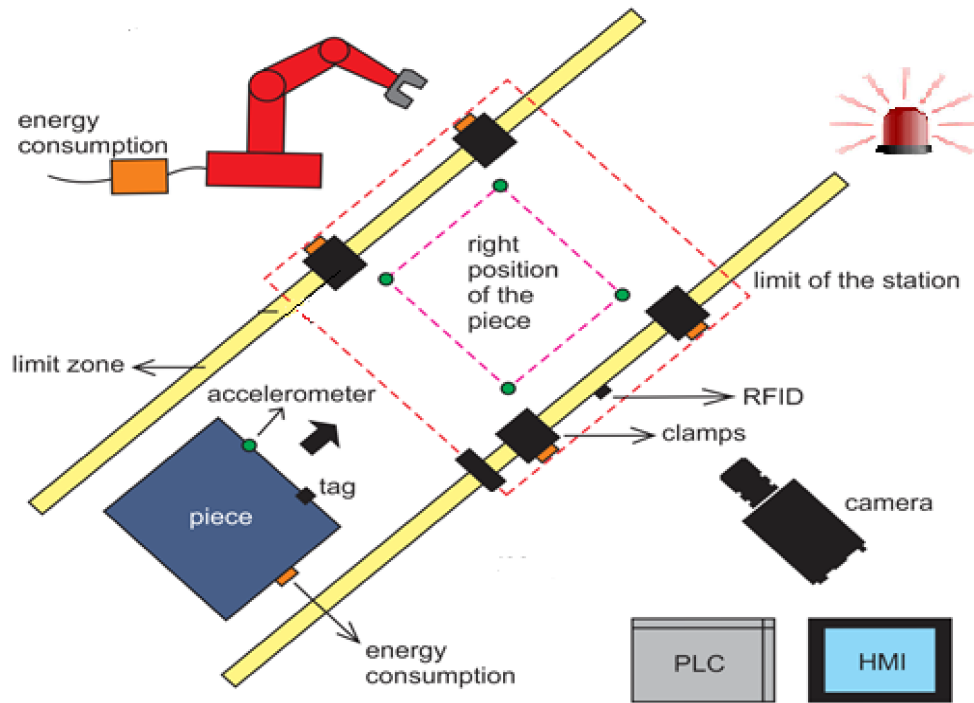


Figure 99. Flexible manufacturing testbed at COMAU's site

In the BEMCOFRA project, a prototype for monitoring energy consumption and flexible car manufacturing was created to demonstrate the potential of IoT in the factory automation domain. Developing software for car factories involves a very complex hierarchical structure, which comprises more than hundred thousand of distributed devices. Normally, developing an application for a car factory must undergo an iterative process to minimize risks of interrupting the production process which as well as the security of the workers.

To review possible approaches, the prototype application was planned to monitor a smaller scale of a manufacturing line as depicted in Figure 99. The software part to connect the robots and the iPad GUI was directly implemented using IoTLink, which required 2.5 weeks of work by two developers who worked 75% for the project.

The main challenge of the development is, the specification of the system changes rapidly as different approaches are to be investigated. Moreover, the integration between industrial devices, wireless sensor networks, and the iPad should be achieved rapidly in order to acquire sensor data and analyze the effects of the system design to the energy consumptions.

The application was designed to capture the domain objects involved in a test bed. The station is responsible for welding the rooftop of a sedan. The station contained a robot with a welding gun, a conveyor system with a skid and clamps that fixed the roof in a position where the robot was able to work. Additionally, wireless sensors were used to measure the state of the clamps and the acceleration of the rooftop entering the station. Wireless cameras were used to check whether the position of the rooftop in the station was correct and to check the quality of the welding spots.

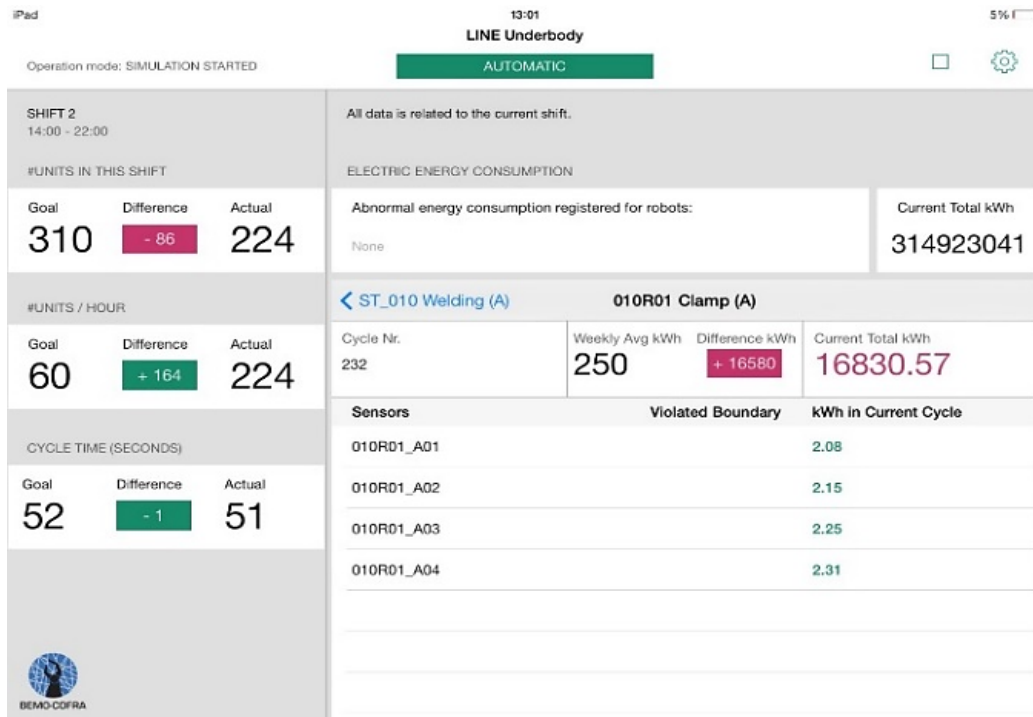


Figure 100. The iPad user interface for monitoring a manufacturing line by a line manager

In the physical station, a PLC was used to coordinate the interaction between devices e.g., when a side body had been detected in a station by a sensor connected to the PLC, the PLC instructed the robot to perform the welding process. After the robot had finished and returned to the starting position, the PLC was informed which then allowed the skid to move the roof out of the station. To monitor the energy consumptions in different level, the prototype application stored the sensor values in a database which then could be aggregated and shown on an iPad to the line manager. As illustrated in Figure 100, the iPad application shows the power consumption of each device, robot, station, and the whole line as well as other sensor values to identify abnormal behavior of the devices in the manufacturing line.

Since the project was only able to provide a limited number of devices, to show the scalability of the prototype some input data had to be emulated using an event generator that replicated the measurement data taken in the earlier phase of the project.

7.2.3.1 Application model

The high-level component architecture of the prototype is shown in Figure 101. On the left side, distributed industrial devices were connected to a programmable control logic (PLC). The PLC stored the sensor values and state of the devices in the station, which could be read from the OPC Server containing the manufacturer’s specific PLC driver. A bridging application was designed to read the sensor values from the OPC server and publish them as MQTT events. Although the generated application is able to communicate directly to the OPC server, the bridging component was introduced since it was useful to emulate a number of data simulating four stations in the line and four robots per station. Each OPC variables was published with a different topic to an MQTT broker (Mosquito). The generated application subscribed to all of these topics by using MQTTInput components.

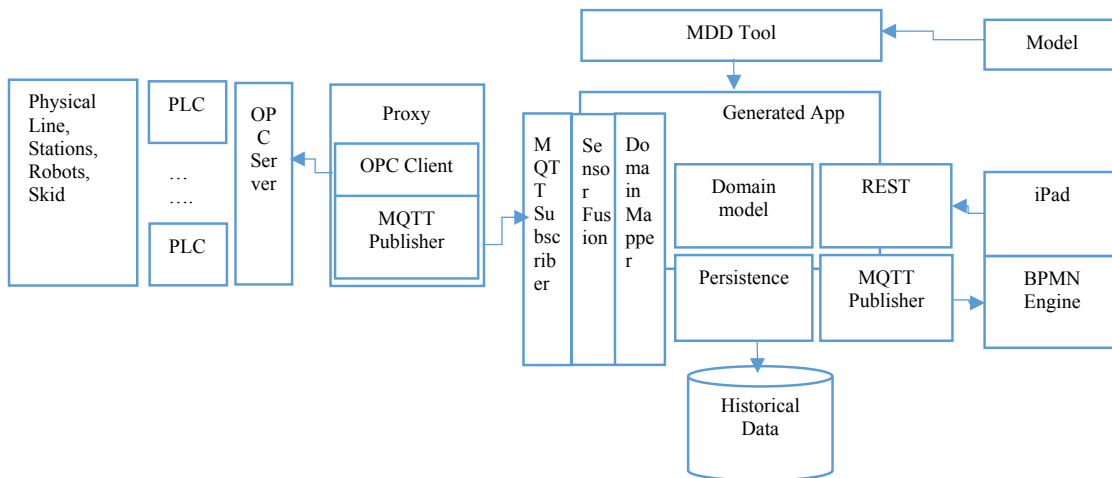


Figure 101. Architecture of the monitoring application

The sensor values had to be correlated directly to the powerConsumption property of the robot axes. And the consumption of the robot axes belongs to a robot had to be fused and assigned to the powerConsumption property of the robot. The power consumptions of the robots in the station must be fused and assigned to the power consumption of the station and

finally the power consumptions of the stations are aggregated and assigned to the power consumption of the line.

The actual and historical consumptions of the devices were stored in a relational database to be analyzed offline in order to create energy consumption profiles of the devices. In addition, the actual power consumption was exposed through a REST service with a JSON data format which is used by the iPad application also to configure the number of objects to be displayed on the screen. The actual state of the entities is also required by the BPMN engine to annotate the data with the actual business process that the data belongs to. This allows the manufacturing processes to be analyzed and improved according to the power consumption data first to reduce cost as well as to reduce the emission of the factory which has been demanded by the regulatory bodies in the last decade e.g., EC Directive 2010/75/EU Integrated Pollution Prevention and Control (Directive, 2010).

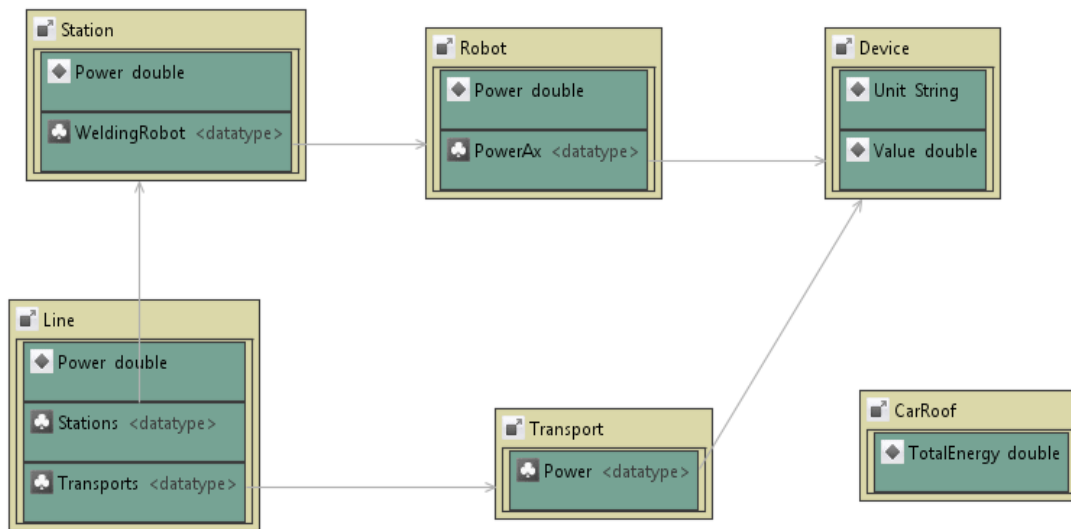


Figure 102. Classes used to represent the entities in the domain.

Developing the prototype was started by creating the application model using IoTLink. In the model, each entity is represented by a class that has properties with the type of list referencing to another class. These references are similar to UML's composition link in a class diagram. Figure 102 illustrates the links between classes in the domain model. The most complex class is the "Line" class that represents a manufacturing line. Each Line has a power property to hold the summary of energy consumptions in the line. It also has a list of stations and a list of transports. The transports and station properties have a reference to Transport and Station classes respectively. IoTLink allows the developers to specify the number of objects that must be instantiated initially. These objects are generated in the main canvas that reflect the concrete implementation so that they can be linked to the data sources as depicted in Figure 103. In the application the each Line is defined with four Stations, 5 Transports that represent the transport from the line entrance to the station 1, from station 1 to station 2, from station 2 to station 3, from station 3 to station 4, and from station 4 to the exit. Each station has four robots, and each robot has four axles. Each axle has a mechanical motor and a power sensor,

however, in the model only the power sensor is included since the application is not interested in controlling the motor.

After the classes are modeled, the virtual objects must be instantiated in the main canvas and linked with necessary the input, output, and sensor fusion components. In the input compartment, an MQTT Input for subscribing to each event topic published by the OPC proxy is created. In the sensor fusion compartment, several fusion components are created first to aggregate the axle power consumption into the robot power consumption, secondly to aggregate the robots' consumptions into the overall station's consumptions, and finally from stations' consumption into the overall line's consumptions. The model uses Esper's CEP engine which can be configured with a domain specific language (EPL) to accumulate the consumption events in batch time interval. Time window based aggregation works well for discrete manufacturing processes which usually has a deterministic execution time. The sensor values are published by the proxy every second, which are summed up to retrieve the energy consumption in watt/second.

SELECT sum(value) from SensorData.win:time_batch(52 sec)

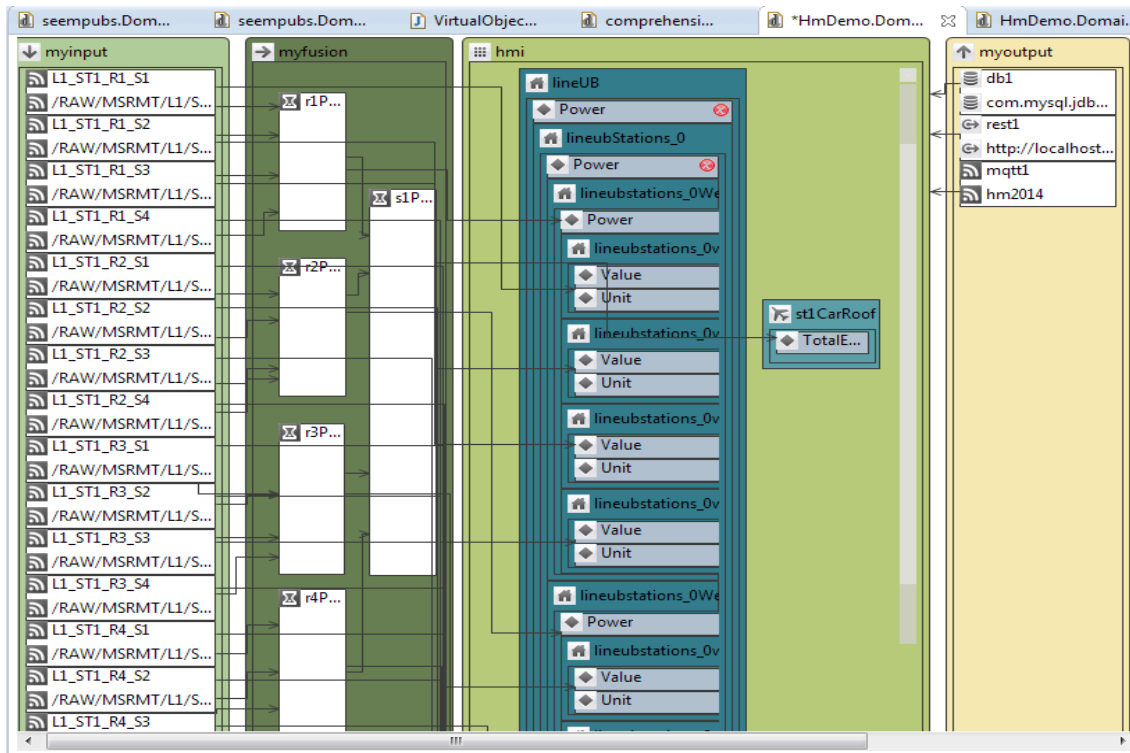


Figure 103. The concrete implementation model where instances of virtual objects are linked to sensor fusion modules and connections to the sensors.

In the virtual object area, a static object to represent the whole Line is required. The content of the Line such as the stations, and the robots are generated automatically based on the number instances that are defined in the class model shown in Figure 102. In addition, a Moving Object is required for every station to represent the product being processed at the station. Each of the moving objects has a TotalEnergy property which is assigned to the VirtualObject

that enters the station. This allows the VirtualObjects to accumulate the energy data from each station providing an overview how much energy is required to produce the roof of a car.

Moreover three output components including the DatabaseOutput, RestOutput and MQTTEventOutput are instantiated. The DatabaseOutput generates the necessary Java Persistence API (JPA) annotations which are used by the EclipseLink to generate the database schema and map the objects into the entries in the database tables. The RESTOutput generates Java classes with JAX-RS annotations that are used by CXF to provide a REST-based service with the following addresses:

- HTTP://localhost:9123/rest/virtualobject/line/{lineId}
- HTTP://localhost:9123/rest/virtualobject/station{stationId}
- HTTP://localhost:9123/rest/virtualobject/robot/{robotId}
- HTTP://localhost:9123/rest/virtualobject/transport/{transportId}
- HTTP://localhost:9123/rest/virtualobject/robot/{robotId}
- HTTP://localhost:9123/rest/virtualobject/device/{deviceId}

The MQTTEventOutput publishes an event every time a property of the virtual objects is updated. For each property, the MQTTEventOutput generates event topics with the following format:

Topic = baseTopic/virtualobject/{Line Id}/{Station Id}/{Robot Id}/{Device Id}/Power

Where the string in the bracket depends on which entity is being subscribed, e.g., the following event topic is generated for publishing the power consumption of station 1:

Topic = baseTopic/virtualobject/LineUB/Stations_1/Power

Generated Database Schema

The DatabaseOutput component annotates the classes in the domain models to determine how these classes must be mapped onto relational database. The annotations contain description of which classes should be mapped onto tables, and how the properties of the classes should be mapped into columns in the tables. Moreover, the annotations also determine the cardinality constraints between the classes.

The DatabaseOutput component uses an EclipseLink persistence framework, which is able to generate the database based on the annotated Java classes. The generated classes are annotated in a way that it configures EclipseLink to generate a table for each class that store the actual values of the VirtualObjects and another “shadow” table for each class that stores the historical data of the objects. Separating the actual state and the historical data of the virtual objects is necessary to keep the cardinality constraint simple.

The DatabaseOutput component also listens to the property changes event of all virtual objects. Upon any property changes, it updates the tables that store the actual values and add a new record to the table that store the historical values.

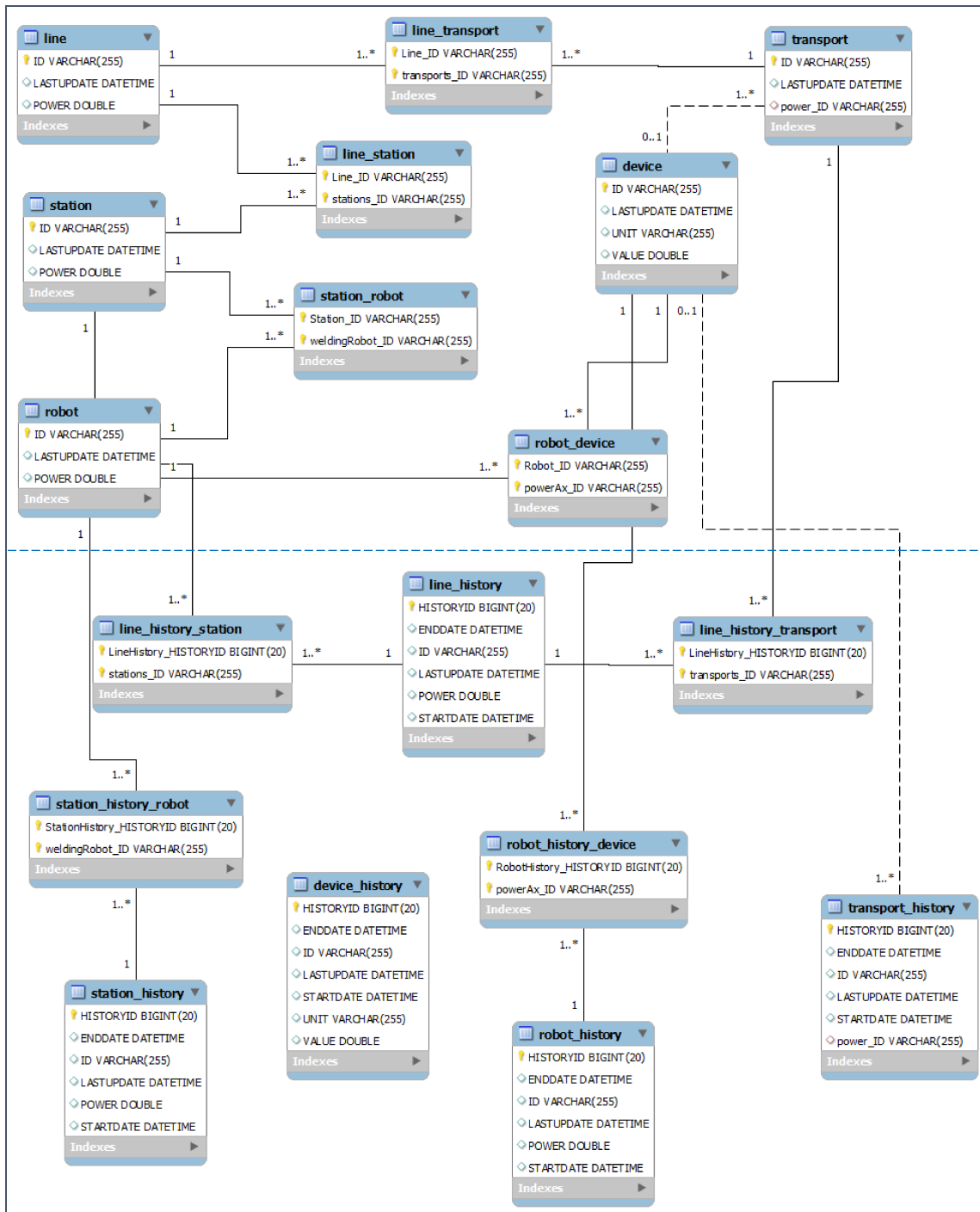


Figure 104. Database schema generated by the DatabaseOutput component.

7.3 Conclusion and Lesson learned

Applying a model driven approach supported by IoTLink to replicate the development of SEEMPubS and ebbits prototypes were able to provide an insight how far IoTLink could be useful for a prototype development in a context of a European research project. It was useful to improve IoTLink iteratively before it was used to develop the BEMOCOFRA prototype.

Replicating the SEEMPubS prototype using the proposed approach took only 3 days and two developers who worked around 70% of their time for the project. The developers were able to come to an initial implementation, which is already able to store the measurement data of the two rooms in a MySQL database, providing the web and mobile applications the state of the rooms through a Web Service, and connecting the state of the rooms to a rule engine. In addition, the generated code must be extended to accommodate the desired behavior, which was not anticipated by the design of IoTLink e.g., the event topics were automatically assigned by IoTLink and cannot be changed visually. Therefore, the generated code must be edited manually to publish the events on the corresponding topics. However, the modification of the code only took 3 days. IoTLink was able to generate many features that are required for the SEEMPubS prototype, particularly the communications to the sensors, summing the energy consumption, providing a Web Service for the application, as well as publishing the system changes through an event broker. The REST API generated by IoTLink is very convenient to use, particularly when accessed from apps running on small devices such as smartphones and tablets. The connection to the Drools rule engine and drools Guvnor was very easy and fast to set up. It provides a more flexible system than the initial prototype. Now, it allows the policy to save electricity to be changed without having to reboot the system.

IoTLink was used in ebbits for developing a proof-of-concept IoT application that collects information about livestock in the farm. Although the prototype was simulated using Arduino sensors, it illustrates how traceability can be supported by IoT technology. IoTLink is able to simplify the development by rapidly integrate sensors that collect the feeding information and weight of the livestock. Moreover, the data is stored in a database and exposed with a RESTful service as virtual objects that can be accessed by abattoirs, supermarkets, and the consumers. After the specification had been finalized, the prototype of the software took three working days and a developer that worked full time. Transferring the specification into IoTLink model was quite straightforward, and no modification to the generated code was required since the prototype was only a proof of concept that is built from scratch.

In BEMOCOFRA, IoTLink was used to develop a more complex prototype. It enables the integration between an iPad user interface and a prototype of the flexible manufacturing station as well as several generated events to simulate a manufacturing line with several stations. IoTLink was able to accelerate the development from connecting the necessary sensors and event generator to the virtual objects, storing the historical states in the database, as well as exposing the virtual objects through a RESTful service. The development was done by two developers with 75% time dedicated to the project and took approximately 10 working days excluding the time used for planning and creating the specifications. The development took much longer than the two previous demos since in BEMOCOFRA; the specification had to be refined after the development had been started.

Overall, IoTLink is definitely able to support a rapid prototyping development when the number of virtual objects is small. Setting up a large number of similar objects arguably could be achieved more effectively with a conventional programming through control loops which are not yet supported by IoTLink. Nonetheless, for developing small scale prototypes that are often required in research projects, IoTLink was quite fast with the tradeoff of less flexibility in customizing the implementations through the visual diagrams. Further customizations can still be done in the generated code, but requires highly skilled developers that already have experience in IoT programming since using Java libraries require the developer to define a more detailed information to use them. For instance, when using Apache CXF to provide a RESTful service, developers are required to define a service class that provides methods returning the virtual objects. IoTLink is able to automate these steps by generating the necessary details from high-level abstractions defined in the visual model.

Table 11. Comparison of estimated efforts in project developments with and without IoTLink

Case Study	The estimated efforts required without IoTLink	The efforts required with IoTLink
<p>SEEMPubs</p> <p>Capture the data from an OPC server, ZigBee plugs, and EnOcean sensors. Aggregate and store the data to a database and automate the lighting based on several rules</p>	6 developers who worked around 50% - 70% for 10-12 months (including architecture design)	Two developers who worked 70% for the project within 3 weeks.
<p>BEMOCOFRA</p> <p>Capture the data from a PLC and a simulator through MQTT, aggregate data, store the data to a database, expose the data through REST and MQTT interfaces and display them on an iPad</p>	2.5 months two developers with 75% time for the project ⁶⁸	3 weeks, two developers with 75% time for the project.
<p>ebbits</p> <p>Capture the data from Arduino scales, RFID reader, store data to a database, and expose the data through a REST Interface.</p>	1.5 months two developers who worked 75% for the project.	2.5 weeks and two developers who worked 75% for the project.

⁶⁸ Estimated using work breakdown structure based on expert judgment (Institute, 2013)

The developers were quite satisfied with the time they needed to create a model and get the generated code. They think that IoTLink could be used to support their work defining the initial prototype which then can be extended with textual programming when complex customization is required. For them, the concept was clear and easy to learn. Nonetheless, when using the Esper sensor fusion and Drools engine they were required to go through long documentation. They would prefer that examples were given so they can learn them quickly. They also mentioned that it could be improved by providing shortcuts to fill the required information. The default GMF notations still have some usability issues. For instance, the target where the users must drop the cursor to create a link was too small and therefore requires developers to move the mouse cursor slowly to create a link. This should be improved to enable a more rapid interaction. Moreover, a developer complains that when clicking the text field on the notations to enter the required information, it does not always directly work. The developers also suggested that they have more control to the output component e.g., they are able to select the virtual objects to be exposed and define the security aspects which might be needed to develop real applications e.g., to prevent unauthorized application accessing the REST API.

The closest use case to a real life development and deployment was the SEEMPubS demonstrator that follows a living lab approach (Macii & Osello, 2011). It involves a significant amount of devices within several rooms that are used every day. Using IoTLink to replicate the demonstrator was straightforward. However, the author also finds that IoTLink is missing a feature to replicate virtual objects automatically as the one that can be done using iteration control logic in programming language. Moreover, a large number of notations and the density of the diagram could overwhelm the users. Therefore, for the future work, IoTLink should be able to partition the diagram to simplify the user interface. The performance of the generated code was quite acceptable to process 32 sensors, but could be improved to reduce latency and the hardware requirements to host the application on an embedded system. The security aspect was missing, which needed to be implemented manually. In the case studies, simple HTTP authentication was added.

On the positive side, the author finds MDD is applicable for real-world development. It accelerates the development significantly and helps keeping the consistency of the code which could easily be a problem when dealing with a large number of software instructions. Furthermore, the proposed architecture and the domain specific modeling language that reflect this architecture is quite simple to understand even by developers that have electrical engineer. Therefore, it was able to facilitate the cooperation between software- and electrical engineers.

Chapter 8.

IoTLink Formal Evaluation

The author believes that controlled usability experiments are essential for finding answers to the research question #3 (*“To what extent a Model-Driven Development approach could support IoT prototyping?”*) since controlled experiments are able to examine the effects of particular variables.

The author performs small usability studies in each iteration for minimizing the development risks and gain a high users’ acceptance at the end of the development. However, this dissertation must keep the development efforts and evaluation efforts in balance in order to achieve the final results on schedule. Therefore, the author decided to apply usability walkthrough which is useful for rapid iterations since it provides “faster, cheaper” method to recognize usability problems early in the design cycle (Hollingsed & Novick, 2007). Additionally, the number of participants was minimized while still able to find around 80% of usability problems. Based on the following mathematical model for finding usability problems, 5 users would be able to find approximately 85% of the usability problems (Nielsen & Landauer, 1993).

$$N(1 - (1 - L)^n)$$

Where N=total number or usability problem and L is the proportion of usability problems discovered while testing a single user with a typical value of 31% (Nielsen, 2000).

Note that the validity of the formula depends on the distribution of the L values between the individuals, which refers to the individual differences between test users (Woolrych & Cockton, 2001). Ignoring the individual differences could risk finding only 55% of the usability problems (Faulkner, 2003).

Because of these concerns, a larger evaluation is required to ensure the validity of the results. Therefore, as IoTLink became more mature, its usability was evaluated again with 24 users to obtain more comprehensive and representative results.

8.1 Initial formal evaluations

To get an initial insight to the answer of research question #3 as early as possible, two usability evaluations were performed in conjunction with two master theses that was done

under the author's supervision. The first study evaluated the initial prototype of IoTLink as well as the conceptual design (Rusmita, 2012). The second study evaluated the concept design of the Discovery Manager (Avila, 2013).

8.1.1 Usability test of the initial IoTLink

The first study consists of a usability inspection to obtain qualitative feedback from the users, and a formal quantitative experiment (usability test) to compare the users' satisfaction between IoTLink and a similar tool based on UML.

Usability walkthrough could be executed in many different ways. However, after reviewing them, the author concludes that the Streamlined Cognitive Walkthrough technique offers the best solution to keep the balance between the efforts required and the results was the main concern in the initial phase since it is designed to tackle a time pressure in industrial environments (Spencer, 2000). Moreover, the users' satisfaction when using IoTLink was measured and compared to a model driven tool that uses a simplified version of UML named ECoreTool⁶⁹.

8.1.1.1 Qualitative inspection

Before executing the Streamlined Cognitive Walkthrough the following required artifacts (Spencer, 2000) were prepared:

1. Define inputs to the walkthrough.

In this step, the goals of the walkthrough were defined, including:

- Evaluating whether the proposed metamodel and layered architecture (depicted in Figure 46 and Figure 50) are easy to understand and adequately designed.
- Evaluating whether IoTLink including the user interface and the workflow are easy to understand and adequately designed.
- Evaluating whether the generated source code is easy to understand and adequately designed.

Then, the target users are identified which are the software developers who have to create rapid functional prototypes, involving IoT communication and data processing.

Moreover, the tasks are defined in a scenario in which the participants were requested to display sensor values from a temperature sensor and a light sensor attached to an Arduino board as well as power consumption of a desk lamp measured by a power meter. Then, the action sequence to complete the tasks are defined in the instruction.

2. Convene the walkthrough

This step was done during the evaluation to provide a clear process and working agreement to execute the walkthrough. First, the goals of the walkthrough are explained, including the scope of activities that will be executed and what will not be executed. Then the roles are assigned which in this case half participants played a role as expert developers and half of the participants played a role as novice developers.

3. Walkthrough the action sequences

⁶⁹ <http://www.eclipse.org/ecoretools/> (Retrieved on Oct 25, 2014)

Then the review team walkthrough the action sequences with IoTLink and for every step, they answer the two questions with a plausible story:

- Will the user know what to do at this step?
- If the user does the right thing, will they know that they did the right thing, and are making progress towards their goal?

4. Record critical information

When the team is able to tell a plausible story for both questions, nothing is recorded. Otherwise, it is recorded. Moreover, design flaws that are able to be identified directly by the review team are recorded.

The cognitive walkthrough involved seven participants between 25-35 years old and have 2-4 years of experience in Java development. Three of them have been involved in at least one IoT research project.

During the process, the review team identified some design flaws e.g.:

- Not all sensor data must be processed by sensor fusion modules before they can be abstracted as virtual objects, therefore, the architecture design should allow cross-layer interactions.
- Copy-paste function was not working inside the compartment; they find it as a fundamental function that IoTLink must have.
- The users would like to be able to define templates for the objects which enable them to change the structure of the objects from the template.
- IoTLink should be able to generate code into the same project every time the code must be regenerated.
- There was no warning when the users use the same object name which then generates an erroneous Java code.
- IoTLink did not provide sufficient error messages to prevent the users generate erroneous Java code, e.g., the user forgot to add the required connections between components.

8.1.1.2 Quantitative evaluation

The quantitative study was executed to compare the users' satisfaction of using IoTLink. This study is quite important to justify the author's choice of using a DSL since the author claims that using a DSL which resembles the proposed architecture would be easier to grasp by the users than using a generic modeling language, such as UML which requires the users to map the solutions designed in the proposed architecture onto the generic modeling language.

Testing this hypothesis, the author designed a within-group experiment that compares modeling an IoT functional prototype using UML and the proposed DSL. To reduce the overhead of the study, the same participants involved in the cognitive walkthrough are used as the subjects. They are given another task to model an application that monitors energy consumption of a DVD player, a table lamp, and a floor lamp in the room. The participants created the two models, in an alternating order, with IoTLink using the proposed DSL (Figure 105A) and EcoreTools using a simplified version of a UML's class diagram (Figure 105B). After performing a task with a modeling language, the participants filled an After-Scenario Questionnaire (ASQ) which "address three important components of user satisfaction with system usability: ease of task completion, time to complete a task, and adequacy of support information"(Lewis, 1991).

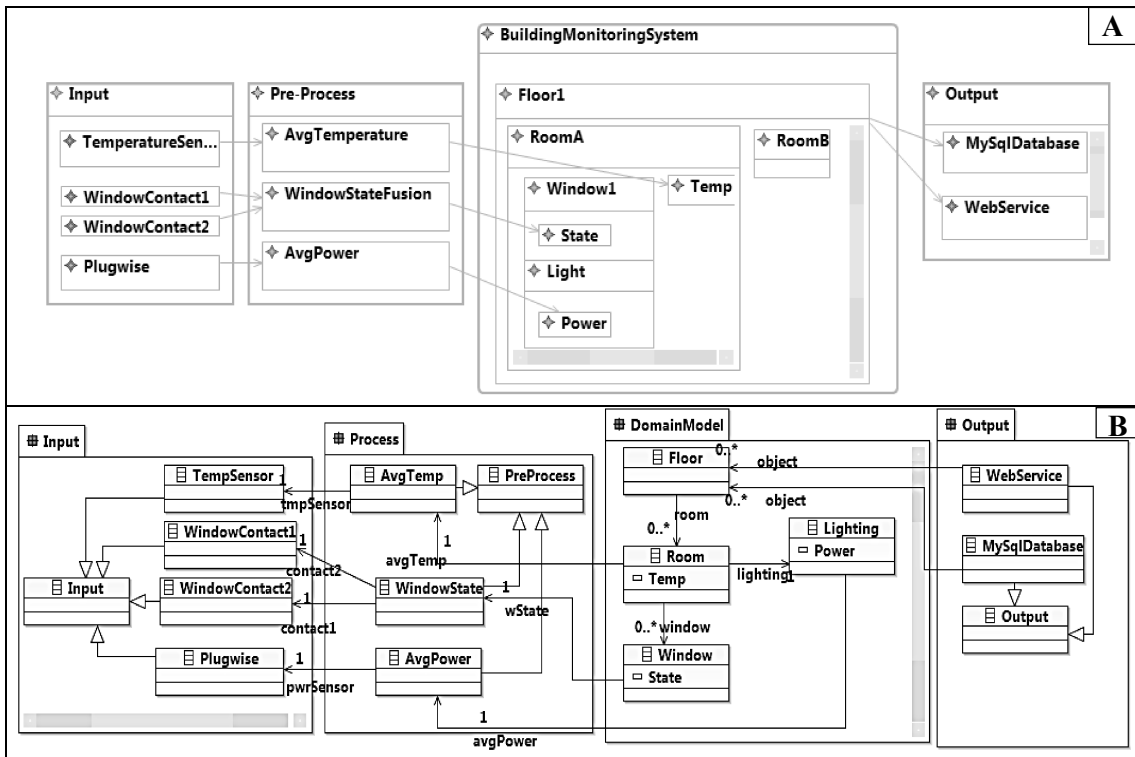


Figure 105. Illustration of UML’s class diagram notation in EcoreTool (A) compared to the DSL notation in IoTLINK (B) (Pramudianto, Rusmita, et al., 2013)

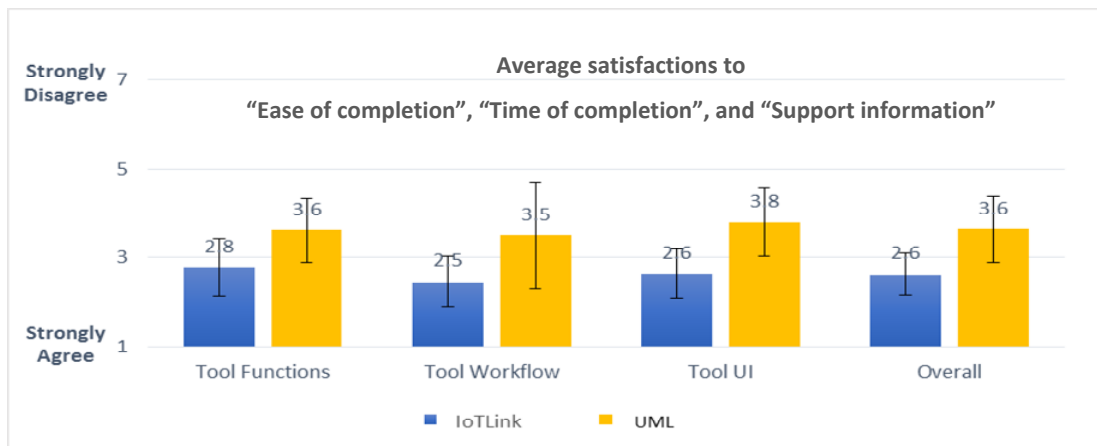


Figure 106. Comparison between EMF tool & IoT Modeling Tool (Pramudianto, Rusmita, et al., 2013)

The satisfaction of the seven participants in three areas (“Ease of completion”, “Time of completion”, and “Support information”) are averaged and summarized in Figure 106. Even though IoTLINK scored a better means in all categories, paired the T-Test analysis shows that the user satisfaction for the Functions, Workflow, and Overall are not significantly different. Conversely, the users’ satisfaction to the IoTLINK’s interface was significantly higher than

UML [$T(6)=2.66, p<.5$]. This finding indicates that a simplified DSL serves a better purpose for simple prototyping tasks than complex modeling languages such as UML since the users are faced with limited options that they have to decide. (E.g., UML has more types of connections, while IoT has only one.)

This confirms that the choice of using a DSL for IoTLink is adequate and able to facilitate the developers performing IoT development tasks. However, the usability of IoTLink needed to be improved in the next iteration, e.g., by using clearer visual cues for different parts of the models and by implementing clearer error messages.

The metamodel was also adequately designed and could be understood quite fast by the participants. The participants pointed out that the sensor fusion modules is not always necessary since many devices have become smarter and able to deliver not only data but also contextualized information. Therefore, the users should be able to skip the sensor fusion and directly link sensors with the virtual objects.

8.1.1.3 Usability Test of the Initial Discovery Manager

The second study was performed to obtain an early insight to answer the research question #2 related to the IoT discovery. The evaluation follows the similar approach used in section 8.1.1 which performs a usability inspection using a cognitive walkthrough approach to obtain the qualitative feedback from the users. The study was then continued with a quantitative evaluation by measuring the users' satisfaction through ASQ (Lewis, 1995).

The cognitive walkthrough was organized with six participants. Four of them have 3-5 year experiences in building automation and IoT projects. Two of them have limited development experiences. The participants were given a four page manual four days in advance. The document contains a brief introduction of the Discovery Manager and the roles of device developers, administrators, and application developers. At the beginning of the cognitive walkthrough, the goal of the study was explained, the Discovery Manager was briefly presented, and then the task sequences for the three user types including device developer, application developer, and administrator were performed by the review team. First, the review team simulates the device developer role. They go through the steps required to provide a device description in JSON file, load the file, and publish the device description through a Web Service. Second, they took the role of the administrators. They had to explore the administrator's user interface to find and correct a device type that was mistakenly assumed as a synonym of another device type. Thirdly, they played the role of the application developer in which they are required to perform a simple query for finding temperature sensor devices. As they performed the task sequence, several usability issues were identified, such as:

- The participants would prefer to have a Java library or in other programming languages that is able to generate the device description instead of defining a JSON formatted description by hand.
- Providing a Web Service and the communication with the Discovery Manager should also be transparent to the developer, which can be wrapped in a software library.
- The Discovery Manager is the one that should provide a method to register rather than having the devices providing a Web Service that is too cumbersome when the device proxies to be deployed in a resource-constrained hardware.

- Handling synonyms should have been done with a standard owl:equivalentClass & owl:sameAs relations which are more natural for ontology experts.
- A REST based service should be provided since it simplifies testing the service through a web browser as well as consuming the service through mobile platforms.

At the end of every task, the participants were also requested to fill an ASQ to measure their satisfaction to the current approach quantitatively.

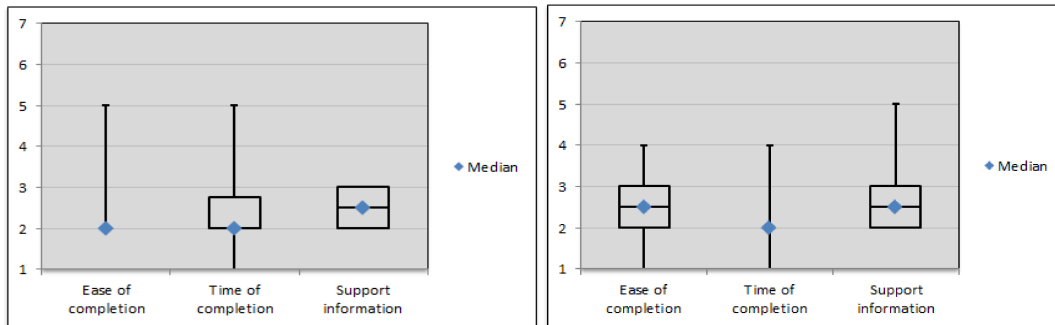


Figure 107. Users' satisfaction of (A) Device developer's role and (B) The administrator's role (Avila, 2013).

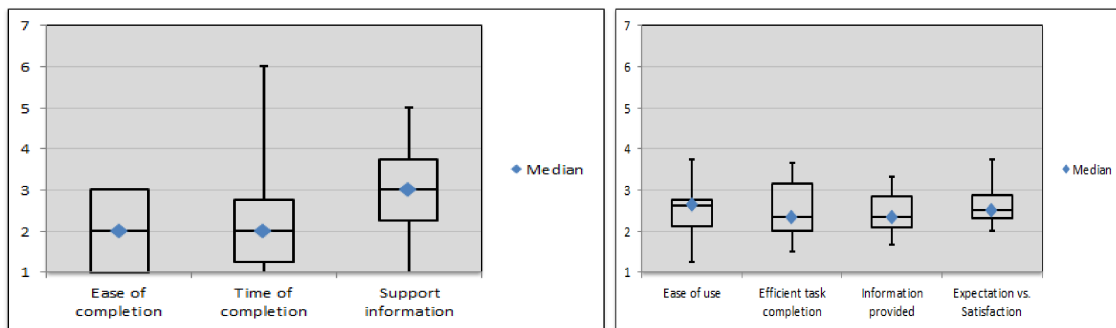


Figure 108. Users' satisfaction of (A) Application developer's role and (B) The overall system (Avila, 2013).

As depicted in Figure 107A (left side), the users' satisfaction when performing the device developer tasks shows that, the JSON file approach was rather easy to understand and fast to complete and scored with a median of two out of 7, while its documentation was rated between 2.5 out of 7. The result of the questionnaires for the administrator's role (Figure 107B) was found easy (2.5 out of 7) and fast (2 out of 7) to complete by the participants. The documentation was rated with 2.5 out of 7. The result of the questionnaires for the application developer's role (Figure 108A) shows that the tasks were rather easy and fast to complete which were rated with a median of two out of seven for both categories. The manual for application developers was rated with a median of three out of seven since the Web Service documentation did not provide sufficient details as expected by the users. The participants rated the overall concept (Figure 108B) of three roles was easy to use (2.5 out of 7) and could help them efficiently sharing and discovering IoT device (2.5 out of 7). The information provided through the user interface and manual was sufficient (2.5 out of 7) and the prototype was working as expected (2.5 out of 7).

These findings confirm that the proposed discovery approach can be understood easily by the three user groups. The efforts required to make devices discoverable, as well as the efforts to discover devices from the application developers are perceived acceptable for them. The participants were excited to be able to discover similar devices that were described with diverse terms. The usefulness of this approach is confirmed by a survey that shows, even developers within the same working group use diverse terminology to describe devices (Avila, 2013). Additionally, the participants also find the possibility to edit manually the mapping between synonymous terms very useful since common dictionaries do not contain domain specific terms. Therefore, the identification of similar terms is not always accurate.

8.2 The Final Usability Test of IoTLink

After the users' suggestions to improve IoTLink and the Discovery Manager were incorporated, and the final usability test was done to provide clearer insights and answers to research the question #2 and #3. The final usability test examined three factors, including the efficiency IoTLink, its effectiveness, and the users' satisfaction. The three factors that resemble usability metrics as described by ISO 9241-11 (ISO, 1998). Various methods exist to investigate these variables and have been discussed at the beginning of Chapter 7 and Chapter 8 including a formal experiments that measure these metrics quantitatively and other methods that are focused on qualitative feedback such as case studies, cognitive walkthrough, and cognitive dimensions (T. R. G. Green & Petre, 1996). A framework to become a standard for evaluating model-driven tools has been proposed (Condori-Fernanndeز et al., 2013). Since it was designed only for MDD, the method is not suitable for comparing IoTLink with middleware approach. Moreover, the method requires a tremendous amount of efforts to prepare and execute. Therefore the author decided to use customized study design which is able to compare the two approaches with a reasonable amount of efforts.

8.2.1 Usability Study Design

To investigate whether IoTLink would improve the current state of IoT prototype development, it must be compared with the common approaches used in IoT software developments. A current survey (S. Bandyopadhyay et al., 2011a) reveals that the most IoT developments were supported by middleware such as LinkSmart (formerly called HYDRA) (Markus Eisenhauer et al., 2009), ASPIRE (Kefalakis et al., 2009), (Katasonov et al., 2008), SOCRADES (de Souza et al., 2008), SMEPP (Brogi et al., 2008) are developed using object-oriented textual languages such as Java, CSharp, XML.

Comparing these two approaches, one must consider that IoTLink relies on visual modeling language while the current IoT middleware approaches rely on textual programming languages. A quite extensive overview of related studies comparing visual languages to textual languages is presented by Navarro-Pietro et al (Navarro-Prieto & Cañas, 2001). However, they mainly focus on program comprehension. They categorize pictorial aids for programming into program visualization systems and visual programming languages. The pictorial aids seem to be effective in enhancing program learnability particularly when the programmers are not trained in advanced. Moreover, it helps improving the programmers' mental model to emphasize the semantic relationship, therefore, visual languages are

particularly beneficial when the tasks require integration and semantic information (Navarro-Prieto & Cañas, 2001). Another study claims that flowcharts have a clearer effect for tasks such as tracing a control flow of software instructions (Curtis et al., 1989). Navarro-Pietro et al performed an experiment comparing C and spreadsheet programming and able to confirm that visual cue on a spreadsheet were able to help developers developing a better data flow mental representation. These findings are aligned with IoTLink's goal in supporting inexperienced developers without extensive trainings in IoT development as well as the design decision to use visual language for organizing the semantic relations between diverse IoT components.

Conversely, there exist findings that oppose to the usefulness of visual aids in programming. For instance, a series of five experiments (Shneiderman et al., 1977) and two stage experiment by Ramsey (Ramsey et al., 1983) could not find any benefit of using flowcharts as a documentation tool to aid a textual source code. When comparing textual and LabVIEW's logic gates and boxes for expressing conditional logic revealed that graphical notations was causing a longer response time for solving the tasks, although some of the participants have experiences in logic gates as well as LabVIEW (T. R. Green et al., 1991). However, their findings show that the experts distinguished from the novices in their ability to use secondary notations.

These findings indicate that the benefits of using visual notations depend on the programming tasks since every notation may highlight a specific type of information while obscuring the others (T. R. Green, 1989). Therefore, the author believes that the mixed results could be caused by the design of the tasks used in the study and the notations. When the tasks that are able to exploit the strength of the notations lead to positive results while the negative results might be caused by the tasks that are not able to exploit the strength of the notations. Nonetheless, programming requires different cognitive functions such as comprehension, structuring, expressing algorithms, in some of which visual aids could be beneficial and textual language might be faster in another part.

Measuring the efficiency and effectiveness of IoTLink for the IoT software development, the tasks used in the study must be designed to measure whether there is an increase of program comprehension. Secondly, the tasks must reveal whether the notations affect the time required to create and modify software instructions as well as the number of mistakes made by the users. As a baseline for the comparison, the time required and mistakes made when using typical development approaches must be measured. These results are then compared to the efficiency and effectiveness of IoTLink when doing the same tasks.

Therefore, the study is designed with two types of tasks that must be solved within two hours to prevent the participants losing their concentration. The first part of the study comprises program comprehension tasks that are designed to measure how well the developers' understand the relation of the classes and domain objects on the given notations. These tasks show related domain objects visualized in a UML notation (Figure 110), the proposed domain specific notation (Figure 111), and Java source code (

Figure 112). They show a few objects and classes that are related to each other. The developers were asked to choose the correct statements representing the relation between the classes and objects. In addition, they were asked to mention the number of specific objects

that are available in the picture. The complete questions presented to the participants can be found in Appendix 1.

The author chose these tasks since they convey the fundamental tasks required in developing IoT applications. The developers must be able to quickly analyze the relations between IoT components in a program that may be done by themselves or other developers. Identifying IoT components required the developers to analyze the semantic relations between the classes and the objects.

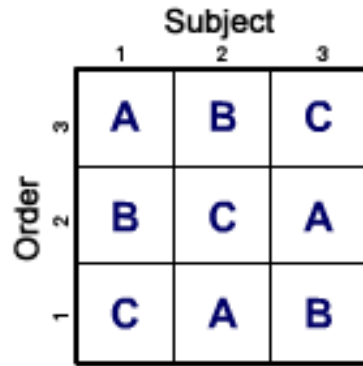


Figure 109. 3x3 Latin square design⁷⁰

The study is designed with the within-subject approach in which each participant had to answer similar questions for the three notations. The time required to answer the questions were measured and the number of the wrong and correct answers were recorded. To balance the fatigue and learning effects across the tasks, the order of the notations for every participant is alternated according to a 3x3 Latin square as depicted in Figure 109.

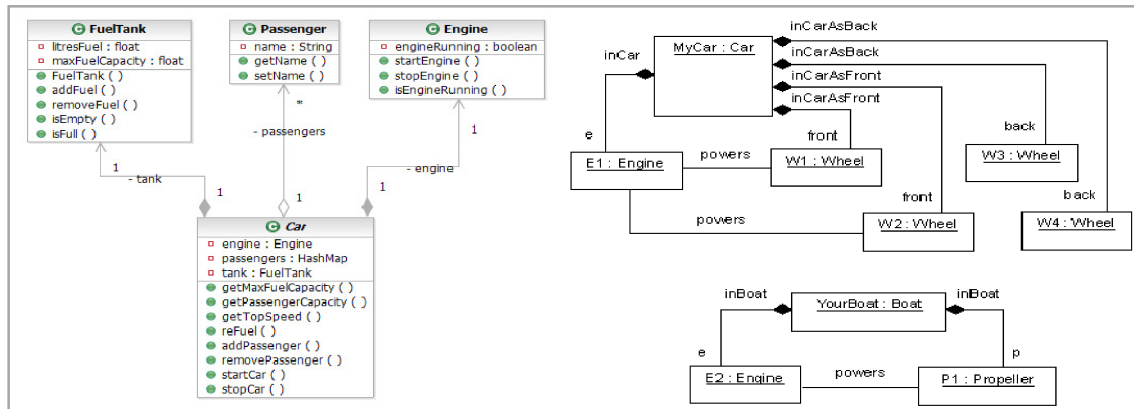


Figure 110. The UML diagram used for the comprehension task.

⁷⁰ <https://onlinecourses.science.psu.edu/stat503/node/24> (Retrieved on July 18, 2014)

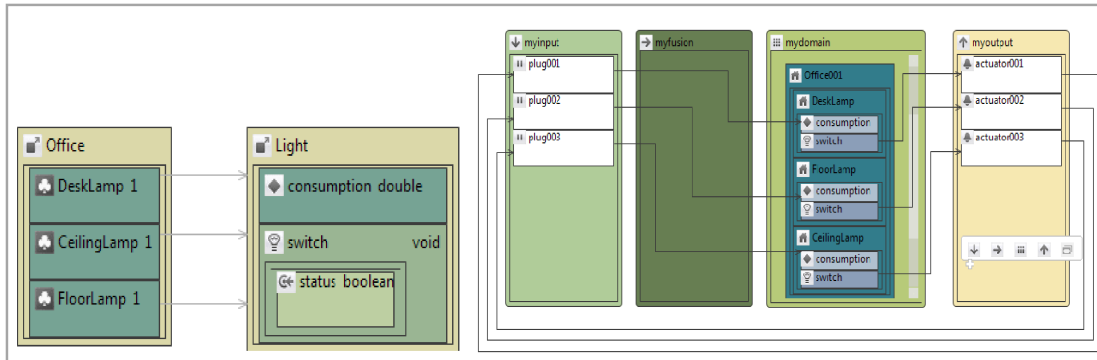


Figure 111. The IoTLINK diagram used for the comprehension task.

```

class Room
{
    List<Window> windows;

    public List<Window> getWindows()...
    public void setWindows(List<Window> _windows)...

    List<Door> doors;

    public List<Door> getDoors()...
    public void setWindows(List<Door> _doors)...

    List<Light> lights;

    public List<Light> getLights()...
    public void setLights(List<Light> _lights)...
}

class Window...
class Door...
class Light...
class SmartPlug
{
    public void switches(Light light) ...
    public void measures(Light light)...
}

public static void Main(string[] args)
{
    Room myOffice = new Room();

    Window leftWindow = new Window();
    myOffice.getWindows().Add(leftWindow);

    Window backWindow = new Window();
    myOffice.getWindows().Add(backWindow);

    Light spotLight = new Light();
    myOffice.getLights().Add(spotLight);

    SmartPlug plugX0000 = new SmartPlug();
    plugX0000.switches(spotLight);
    plugX0000.measures(spotLight);

    Light floorLamp = new Light();
    myOffice.getLights().Add(floorLamp);

    SmartPlug plugX0001 = new SmartPlug();
    plugX0001.switches(floorLamp);
    plugX0001.measures(floorLamp);

    Light ceilingLamp = new Light();
    myOffice.getLights().Add(ceilingLamp);

    Light emergencyLight = new Light();
    myOffice.getLights().Add(emergencyLight);
}
    
```

Figure 112. The source code used for the comprehension task.

In the second part of the study, the efficiency of IoTLINK was measured through the time required by the developers to create a program with it. To provide a baseline for the measurement, the time should be compared with a similar tool such as a UML tool that is able to generate Java code. However, according to the author's knowledge, the available UML tools are only able to generate Java interfaces from a class diagram. In contrast, IoTLINK offers a step further from the existing MDD approaches by allowing developers to work directly with concrete objects and generate the complete Java implementation. Therefore, a fair comparison between the IoTLINK and a UML tool in the second phase of the study was not possible. In the second stage, IoTLINK was only compared to Java programming supported by a middleware library that provides a similar abstraction to IoTLINK's components.

The author chose tasks that are required to monitor the temperature and light intensity in two rooms using IoTLINK and Java programming. These tasks were chosen since they represent

the most common scenario in IoT development, which involves monitoring the states of physical objects. The author did not present any control scenario since it requires the participants to define the application logic in Java or Drools rules. This would require significantly more time and therefore could introduce undesirable effects during the study such as loss of concentration and fatigue.

To measure the time in different step of the development, the program was decomposed into a set of smaller tasks as follows:

1. Defining a domain model comprises a class and two objects that represent two rooms.
2. Subscribing to four MQTT events and update the domain objects based on the values.
3. Perform an average of the light intensity values before they are assigned to the property of the rooms.
4. Publish the objects through REST-based service that expose the room objects with their temperature and light intensity as the properties.
5. Publish the objects as MQTT events when the temperature and light intensity are updated.

These smaller tasks examine the advantages of IoTLink when used for defining different components within the proposed architecture. For instance, task 1 identifies the ability of IoTLink for defining objects and classes and task 2 identifies the ability of the connection components to abstract communication with different sources.

Since the study follows a within-subject design, each participant had to do the tasks with IoTLink as well as with the Java library. Similar to the first part of the study, to avoid learning and fatigue effects, the order of the development tool used by the participants was alternated.

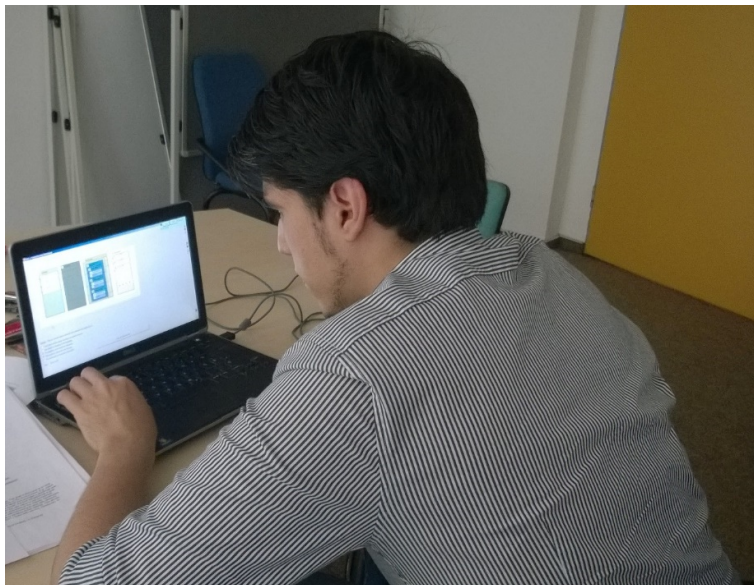


Figure 113. A participant was doing a comprehension task.

After performing each individual task the participant was asked to fill an ASQ that inspects the user's satisfactions to the three factors, including (1) the time required to complete the task, (2) the support information, and (3) the overall satisfaction (Lewis, 1991). After the five tasks were done with a development tool, the participants are asked to express his satisfaction

from four different aspects including (1) the overall satisfaction score (OVERALL), (2) system usefulness (SYSUSE), (3) information quality (INFOQUAL) and (4) interface quality (INTERQUAL) that are presented as 19 questions in the Post-Study System Usability Questionnaire (PSSUQ) (Lewis, 1992). In addition, the time required by the participants to perform the tasks are recorded.

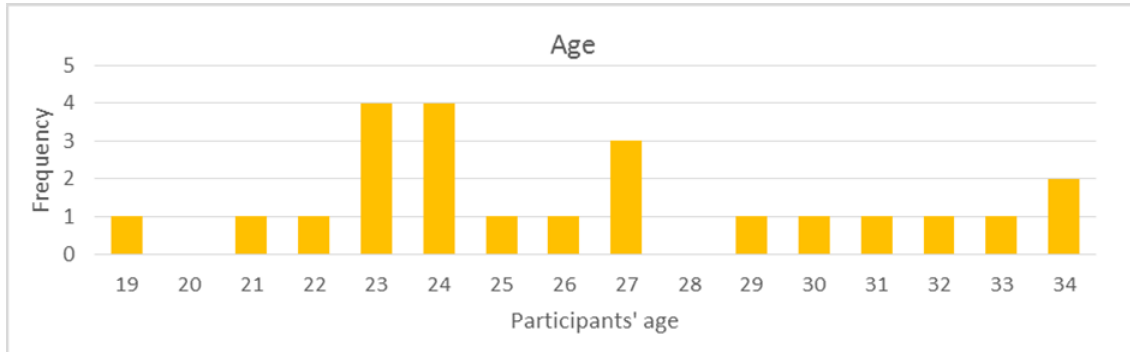


Figure 114. Age distribution of the participants

The study was performed by 24 participants (23 males and one female) who were randomly chosen from project partners and colleagues. The participants’ age ranged from 19 to 34-year-old with the median age of 24.5. Their experience profile is depicted in Figure 115. The participants’ object-oriented experiences range from 0.5 to 17 years with a median of 5 years. The participants have none to 12 years experiences in UML with a median of 2.5 years. In the network programming area, the participants none to 12 years experiences with a median of two years. The participants have none to six years of IoT experiences with a median of one year.

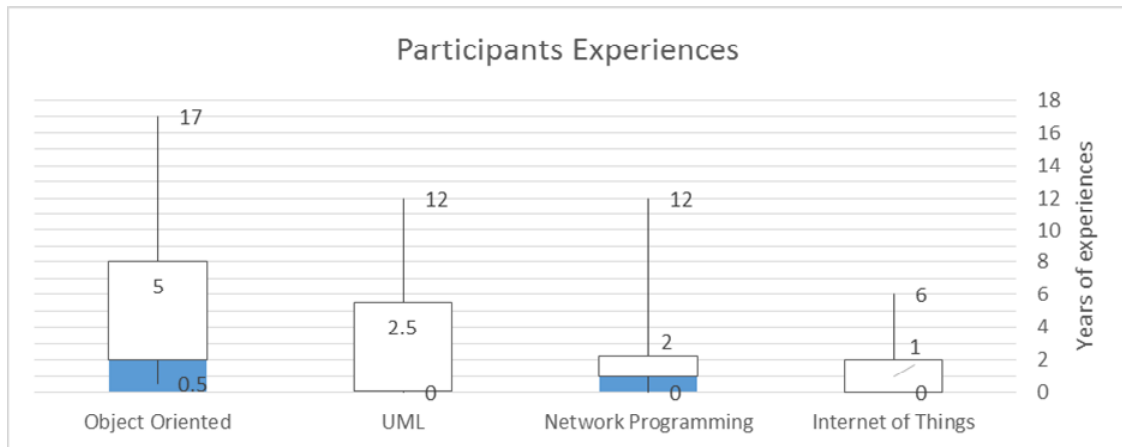


Figure 115. Participants’ experiences in Object-Oriented, UML, Network Programming and IoT.

The participants' profile shows that most participants have little to none experiences in the area of internet of things as well as network programming which are the target user of IoTLink. However, IoTLink should be able to support users with extensive, as well as little

knowledge of object-oriented programming and modeling (UML). This profile is also fulfilled by the participants' profile.

The evaluation was performed on a Windows-XP virtual machine that was run on a dell latitude E6230 with core i7, 16GB Memory, 256 SSD hard drive connected to a desktop keyboard and mouse. Moreover, the participants are given with 30 pages documentation for the IoTLink and the Java library that can be used when performing the tasks.

8.2.2 Final Evaluation Results

8.2.2.1 Results of the comprehension tasks

Figure 116 shows the average time required by the participants to answer the questions, determining the relation between classes, the number of objects and the relation between concrete objects. The classes and objects are presented in three different formats including UML diagram, Java source code, and IoTLink diagram. The result shows that the participants required the least time when analyzing the UML class diagram (M=1:34; SE=21.7), followed by the IoTLink diagram (M=1:56; SE=12.4), and the source code (M=2:08; SE=11.9). However, repeated measures ANOVA reveals that there are no significant differences on how fast the participants could understand the relations between classes that are presented in UML, source code, and IoTLink diagram.

When the participants were asked to analyze the number of the objects in the three different presentations, the time required for analyzing the IoTLink diagram (M= 0:48; SE=5.5) was 20 seconds (29%) faster than interpreting the source code (M=1:08; SE=4.6) [T(23)=3.8, p<0.05]. Compared to the average time required to interpret the UML diagram (M=00:54; SE=5.3), IoTLink was 6 seconds (11%) faster. However, paired T-Test shows that there is no significant difference between IoTLink and UML diagram.

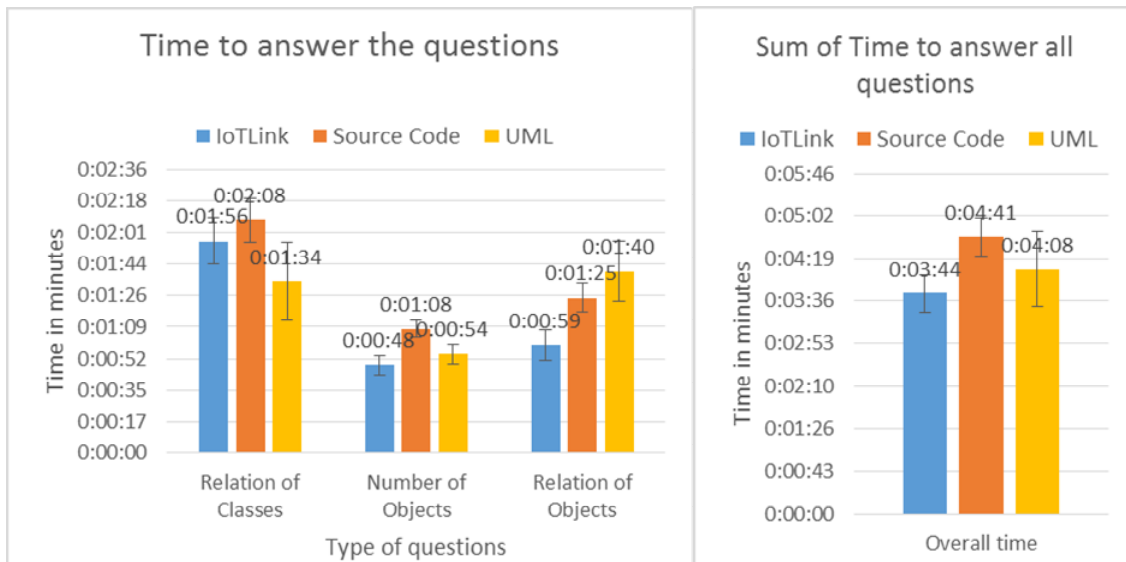


Figure 116. The average time required by the participants to answer the questions from three different presentations of the classes and objects.

In the last tasks where the participants had to analyze the relations between objects, the average time required for interpreting the IoTLink diagram (M=00:59; SE=8.2) was able to outperform the time required to analyze the source code (M=1:25; SE=7.9) by 26 seconds (31% faster) [T(23)=3.8, p<0.05] and the time required to analyze UML (M=1:40; SE=16.6) by 41 seconds (41% faster) [T(23)=2.3, p<0.05].

Overall, the average time required to analyze IoTLink (M=3:44; SE=19.6) diagram was 57 seconds (20%) faster compared to analyzing Java source code (M=4:41; SE=19.6) [T(23) = 3.3; p<0.05] and 37 seconds (10%) faster than UML diagram. However, a paired T-Test shows no significant difference between the total time required to analyze UML and IoTLink diagrams.

Analyzing the number of mistakes that the participants made when answering the comprehension tasks, the results show that the participants made the least mistakes when interpreting the UML diagram (M=0.38; SE=0.12). The participants made the most mistakes when determining the relations between classes, particularly when the classes are presented with the source code (M=1.2; SE=0.12), followed by the IoTLink diagram (M=1; SE=0.19). A paired T-Test shows a significant difference between the mistakes done when interpreting IoTLink and UML diagrams [T(23) = 3.3; p<0.05].

When the participants identified the number of objects presented in a UML diagram, the participants did not make any mistake. When the objects were represented in the source code, two participants made a mistake. And one participant made a mistake when the objects were presented in an IoTLink diagram. Repeated measures ANOVA reveals no significant differences between the means.

When performing the third task, in which the participants analyzed the relations between objects, the least mistake was made when they analyzed the source code (M=0.08; SE=0.06) followed by IoTLink (M=0.25; SE=0.09) and then by UML (M=0.29; SE=0.14). Repeated measures ANOVA reveals no significant differences between the means.

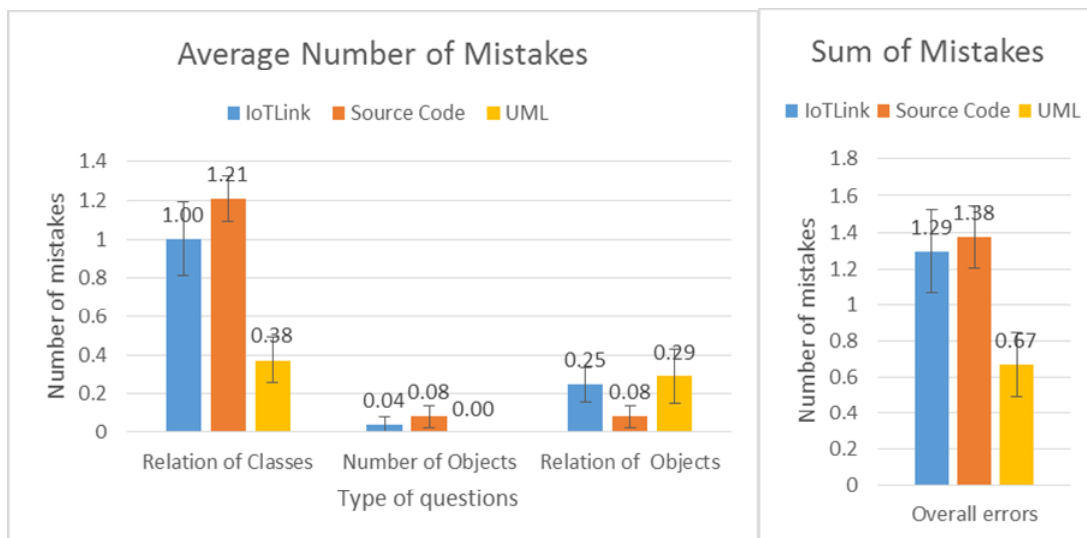


Figure 117. The average mistakes found in the participants' answers.

Overall, there was a significant difference when comparing the average mistakes made by the participants when analyzing UML diagram and IoTLink [$T(23) = 2.1$; $p < 0.05$] as well as when comparing UML diagram and source code [$T(23) = 3.2$; $p < 0.05$]. However, in terms of mistakes made by the participants, there was no significant difference between IoTLink and source code. Overall, the participants made the least number of mistakes when analyzing the UML diagram ($M=0.7$; $SE=0.2$) followed by IoTLink ($M=1.3$; $SE=0.2$) then source code ($M=1.4$; $SE=0.2$).

Discussion

The evaluation results of the first phase provide insights on how the participants are able to comprehend object-oriented artifacts on three different notations. Some participants claim that diagrams are able to present class relationships more clearly than source code. Consequently, the UML diagram presenting the relations between classes can be scanned significantly faster by the participants than the source code. The participants were not familiar with the IoTLink class diagram and therefore they needed more time to understand the information presented by it.

The result shows that, as the diagrams became more complex, they required more time to analyze particularly when the diagrams are not systematically structured. The structure of the diagrams influences the time required to comprehend the content. When the diagram has no clear structure or if the users are not aware how the diagram is organized, the users first have to find the starting point of the diagrams. Since there is no clear convention on how to read unstructured diagram, the users may scan them randomly to find the starting point that results in a longer time to analyze. Once the starting point is found, they scan the relations between the objects of interest. In contrast, the presented source code has a clear structure that can be skimmed systematically from left to right and from top to bottom.

When the users have to analyze a complex diagram that have a clear structure and the density of the objects is not too overwhelming for the users, they require less time to scan the diagram since it provides an overview of the presented information. This is confirmed by the IoTLink object diagram that requires the least mean time to analyze. Since it was arranged systematically from left to the right, it helps developers to analyze the relations between the concrete components systematically. Moreover, the number of mistakes made by the participants is also consistent with the time required to analyze them since the time represents their hesitation when not knowing the correct answers. However, the differences were not significant. Therefore, the author cannot draw any conclusive arguments.

Some of the participants who have some years of experience with UML diagram argue that UML diagrams are very helpful to provide an overview of the system architecture compared with tracing the source code. In the contrary, some developers who have long experiences in Java admitted that they prefer to work with textual language since they are more familiar with textual language and tracing codes nowadays has been simplified by the code editor which allow programmers to find links between objects and classes. This shows that developers have their own preferences on programming or modeling language depending on their experiences and how productive they could be when using the language.

8.2.2.2 Results of the programming tasks

Time for the task completion

The second part of the study required the participants to develop an application that monitors the light and temperature in two rooms. Each participant must build the same application twice, with Java programming (supported by a middleware library) and IoTLink. The average time required by the participants to finish the tasks and the number of errors are shown in Figure 118. The number of errors was determined by whether or not the users could provide a complete solution for the tasks. Each incomplete solution is counted as one mistake.

Performing analysis on the time required to complete all programming tasks using paired T-Test shows that there was a significant difference when the participants used IoTLink (M=35:3; SE=3:48) and Java (M=1:03:53; SE=5:33) [T(23)=4.8, $p < 0.05$]. Furthermore, the Cohen's d effect size value ($d = 1.3$) suggested a large practical significance.

Figure 118 shows using IoTLink to perform all tasks was in average 28 minutes 23 seconds (44%) faster than using Java libraries. Moreover, the number of mistakes made by the participants when using IoT (M=1; SE=0.22) was on average 48% less than the mistakes made by participants when solving the tasks with Java (M=1; SE=0.22) [T(23)=3.1, $p < 0.05$]. Furthermore, the Cohen's d effect size value ($d = 1.4$) suggested a large practical significance.

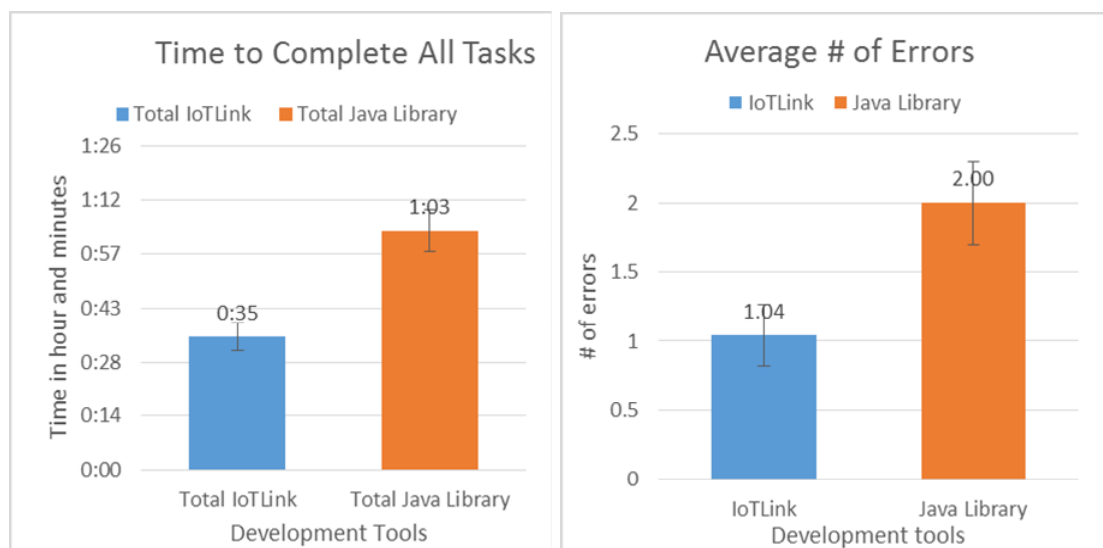


Figure 118. Total time required by the participants to perform all programming tasks (left) and the number of errors made by the participants (right).

The time required for solving each task by the participants is illustrated in Figure 119. The average time required to solve task 1 with IoTLink (M=10:40; SE=1:30) was 25 seconds (4%) more than using the Java library (M=10:15; SE=1:19). Nonetheless, a paired T-Test analysis reveals no significant difference. In contrast, a paired T-Test analysis of the time required by the participants to perform task 2 shows a significant difference [T(23)=3.7, $p < 0.05$]. The average time required when using IoTLink (M= 09:32; SE=00:57) was 9 minutes 17 seconds (49%) faster than when using the Java library (M=18:49; SE=02:11). Furthermore, the time required to solve task 3 with IoTLink (M=5:54; SE=0:50) was 4 minutes 4 seconds (41%) less

than using Java (M=9:58; SE=1:28) and paired T-Test shows a significant difference [T(23)=2.3, p<0.05]. Additionally, significant differences could also be found in the times required to perform task 4 [T(23)=5.8, p<0.05] and task 5 [T (23)=4.6, p<0.05]. IoTLink (M=5:20; SE=00:55) was able to outperform Java libraries (M=14:05; SE=01:47) in average by 8 minutes 45 seconds (62% faster) when it was used to solve task 4. When it was used for solving task 5, IoTLink (M=4:03; SE=00:45) was again able to outperform the Java libraries (M=10:45; SE=01:25) on average by 6 minutes 42 seconds (62% faster).

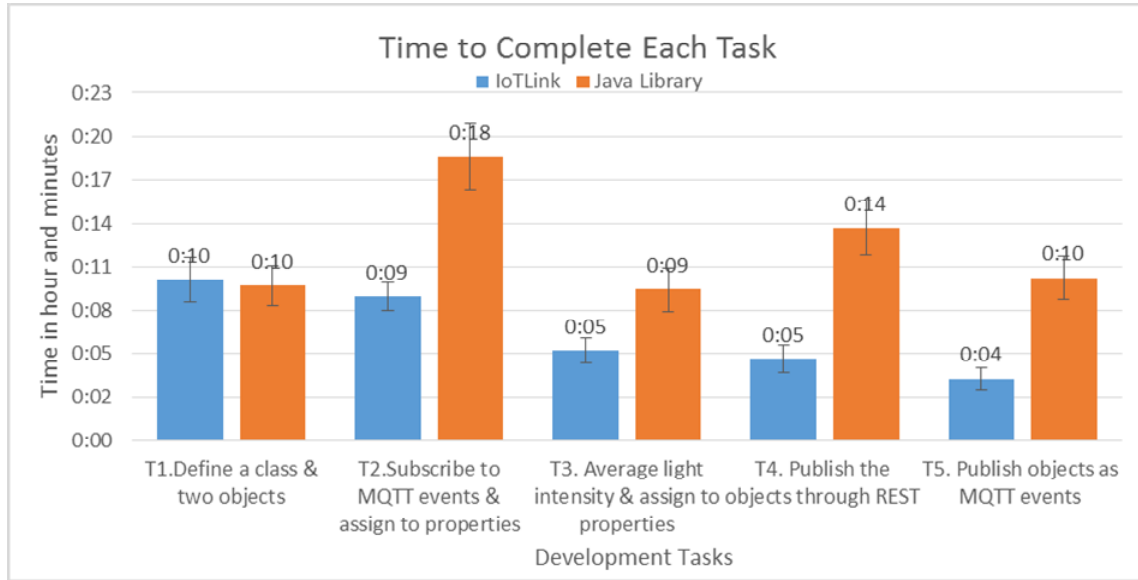


Figure 119. Time required by participants to complete the tasks.

Discussion

The results show that in general, IoTLink was able to support the developers building IoT applications faster and with fewer mistakes. We could see a pattern from these tasks where IoTLink is faster for linking components than using the Java library. For instance, task 2 requires the users only to select the components and drop them in the input container. To link the input components to the domain objects, the users only need to draw lines from the input components to the property of each object. In opposite, using Java, the users must instantiate the connection components. Since there are different possibilities to link objects in Java, the users have to find out how to link them (E.g., by creating a listener and finding the correct method to register the listener). The input components in Java require the users to create a listener object which is notified when the data from the data source arrives. Then the users could set the properties of the domain objects based on the data received by the listeners. These steps are reduced significantly when the users use IoTLink. Moreover, IoTLink provides a limited possibility how to link the objects compared to Java, which also make developers understand and decide what to do faster when using IoTLink.

In task 3, the average time of IoTLink shows a significant improvement over Java. Although IoTLink could not automate some parts of the process by code generation, it might have been caused by the effect of visual component which is straightforward to use and therefore the developers gain a better confidence as they use it. Meanwhile, when using Java code in this

task, the developers have some doubts to use the library and some of them even went through the source code of the sensor fusion component trying to understand the inner workings of the module. Another factor that shortens the duration of task 3 done through Java was that the user either reused the listener that they already did for task 2, therefore they were able to finish the task 3 much quicker than task 2 when creating the listener. Had the users did not reuse the listener code from the task 2, the time would have been greater.

Using IoTLink to perform task 4 and 5 was also significantly faster since the users were only required to select the components, put them in the corresponding containers, and draw lines from the output components to the domain model container. In opposite, using Java to expose the objects through REST requires the users to annotate the Java Beans and create a service class with methods to be executed when the REST service is accessed. This process can be simplified much further by IoTLink since it is able to generate the necessary service classes and annotate them automatically. In task 4, IoTLink was able to generate the necessary code for publishing events to an MQTT broker which reduces the process further than what an abstraction through a Java library could provide. Therefore, although the Java library and the IoTLink component may provide the same abstractions, a code generation brings further advantages that simplify the development.

When IoTLink is not able to simplify the process to solve the task, it could not reduce the time required by the developers as we can see from the result of task 1. IoTLink was even slightly slower since some participants had difficulties on clicking the text fields on the notations. Moreover, some experienced participants were able to take advantage of the Eclipse’s wizard to generate a class. Some more experienced users were able to write code manually quicker than interacting with visual user interface.

Interaction Effects between the Tools and Experiences

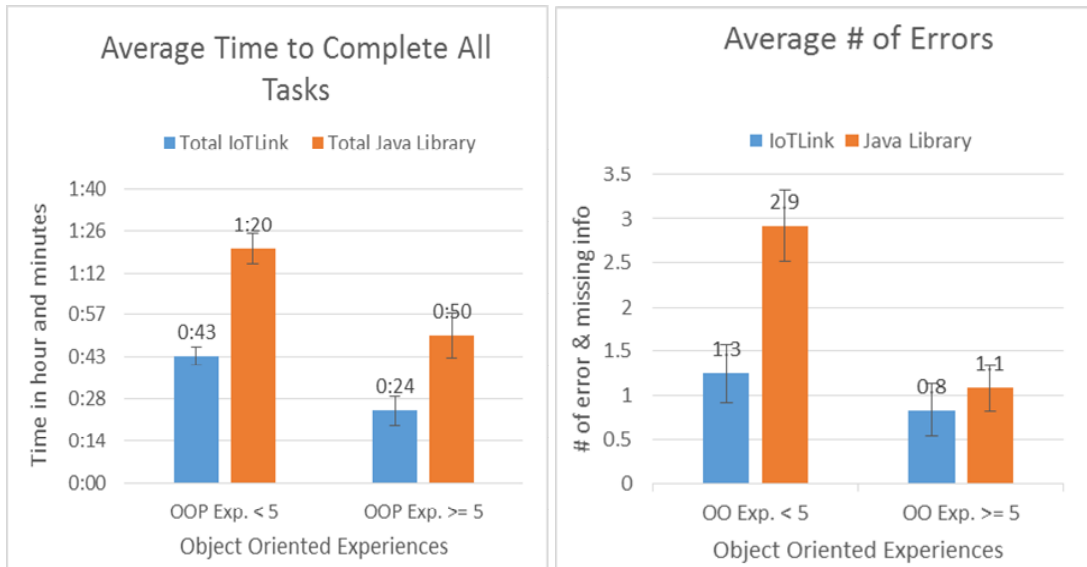


Figure 120. Total time required by the participants to perform all programming tasks (left) and the number of errors made by the participants (right) categorized by the object-oriented experiences.

To understand if IoTLink has different effects to the less experienced participants, the results of the experiment are divided into two groups including the participants who have experiences less than five years and the participants who have five or more years of experiences with object-oriented. Since there are only 11 participants who have at least five years experiences in object-oriented, the same number of participants from the second group must be selected in order to be able to analyze their difference using statistical method such as T-Test and ANOVA.

Analyzing the mean time required by the participants when using IoTLink, the result shows that the participants with object-oriented experiences less than 5 years ($M=43:33$; $SE=05:17$) were on average 43% slower than the participants who have 5 or more years experiences ($M=24:47$; $SE=03:11$) [$T(16)=3.1$, $p<.05$]. Similarly, when they used Java, the participants who have less than 5 year object-oriented experiences ($M=1:20:16$; $SE=7:36$) were on average 37% slower than the participants who have more than 5 year ($M=50:33$; $SE=5:03$) [$T(17)=3.3$, $p<.05$]. When the participants who have object-oriented experiences less than 5 years used IoTLink, they were 46% faster than when they used Java. In addition, when the participants who have 5 or more years of experiences used IoTLink, they were 51% faster than when they use Java.

Two-way mixed ANOVA confirms that when ignoring the experience factor, the tools have a significant effect on the mean time to solve the tasks, as well as the number of mistakes made by the participants. Additionally, the experiences of the participants also have significant effects on both the mean time and mistakes. However, it does not show any significant interaction between their object-oriented experience and the tool that they used.

In contrast, two-way mixed ANOVA shows that the number of mistakes was significantly affected by the interaction between the tools and object-oriented experiences ($F(1,20) = 5.8$, $p<.05$, $r = .23$). IoTLink was able to reduce the mistakes made by the participants with less than 5 years object-oriented experiences ($M=1.3$; $SE=0.3$) close to the number of the mistakes made by more experienced developers ($M=0.83$; $SE=0.3$). When Java was used the number of mistakes made by the less experienced developers ($M=2.9$; $SE=0.4$) was significantly higher than the experienced developers ($M=1.1$; $SE=0.3$) [$T(19)=3.9$, $p<.05$]. Interestingly, the developers with more than five years experience seem to manage minimizing the number of mistakes when they used both IoTLink and Java. Although their mean time when using IoTLink was slightly lower than Java, T-Test analysis shows no significant difference.

Discussion

IoTLink is clearly able to support both the experienced and less experienced developers. It was able to reduce the mean time required by both groups in solving the tasks. However, it does not have greater or fewer effects on the meantime of any particular group. When using Java, even the more experienced participants needed some time to discover the necessary steps required to use the Java library such as defining the required classes, instantiating the objects, and learn the required methods as well as their parameters. Passing values between the objects could be done differently, e.g., by polling the relevant methods or by applying publish-subscribe pattern

On the other hand, IoTLink offers more uniform patterns to use the components which can be learned quickly by the participants from both groups. When using IoTLink, the participants

only needed to focus on linking the components between different containers that are separated clearly. The only distinctions between the components are the required parameters on the property sheet. The participants are required to understand the expected input by different components. Conversely, IoTLink could not significantly reduce the mistakes made by the more experienced developers further since they were already able to minimize the number of the mistakes when they used Java. The author believes that their experiences made them more careful when writing the program and dealing with different design patterns that are used by the Java library. In contrast, when the less experienced participants developed the solution in Java, they made a higher number of mistakes. Mostly, the less experienced developers forget to link the components after they initialize the objects. Some of them did not fully understand that the Java library was designed with a publish-subscribe pattern, therefore, initially they did not define listeners and tried to poll the values. Although they were provided with the documentation of the library, they did not read the documentation thoroughly because the time was limited. They also asked more questions and needed more help for solving the tasks. This proves that the less experienced participants were not able to map their solution easily in a generic programming language such as Java since the abstractions provided by generic programming languages do not directly resemble domain specific solutions. Consequently, the developers need to perform two stages of problem-solving process. First, they need to develop their domain specific solutions in an abstraction that close to the problem space. Then they need to translate the solution to the abstraction that the programming language requires.

IoTLink was able to reduce the mistakes made by the less experienced developers significantly since IoTLink notations provide an abstraction that is close to the application domain. Thus, it was easier for the less experienced developers to express their solutions in IoTLink notations. In contrast, the more experienced developers were able to translate their solutions to Java and IoTLink only with minor helps since they are accustomed to translating solutions for different application domains into Java.

Users’ perceptions

Analyzing the users’ perception to both methods used in solving the tasks, they were asked to fill a post-study questionnaire. Figure 95 shows the result of the questionnaires.

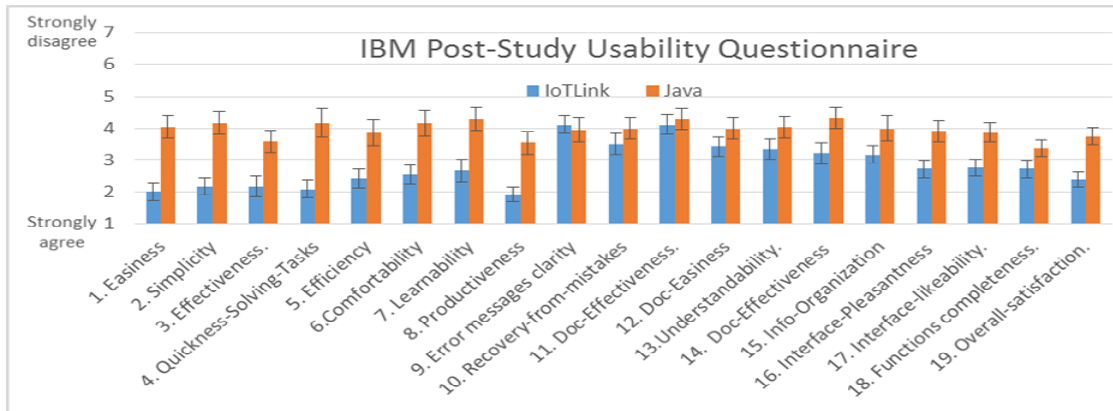


Figure 121. Participants’ satisfactions post-study questionnaire with the complete questions.

As depicted in the picture, IoTLink was rated easier, simpler and more effective for solving the tasks. According to the participants, IoTLink is easy to learn and able to support their productivity. However, the error messages and documentation were the weak points of IoTLink, which affects the ability of the participants to recover from mistakes. The user interface presents a good organization of the information; it was pleasant, and easy to understand. However, the participants feel that it should be improved, particularly with debugging & tracing functions, as well as clear error messages and instructions as tool tips before they would use it for producing large applications. Applying a paired T-Test analysis of the score for each question shows that there are significant differences on the question 1-8 and 14-19. There were no significant differences on the questions 9-12 that are related to the documentation and error messages.

Figure 122 shows the results of the post-study questionnaires for both IoTLink and Java, which are categorized according to the type of questions (Lewis, 1995). The analysis using paired T-Test while assuming unequal variances shows that there are significant differences between IoTLink and Java on the overall satisfaction score (OVERALL), system usefulness (SYSUSE), information quality (INFOQUAL) and interface quality (INTERQUAL). The users perceive IoTLink is superior in all groups.

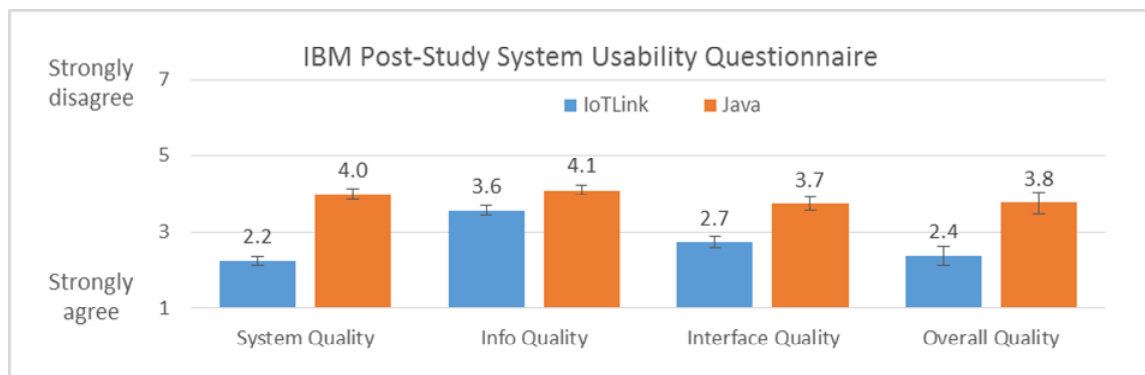


Figure 122. Participants' satisfaction to IoTLink compared to Java in a rapid prototyping.

Discussion

The result of the post-study questionnaires shows that the participants enjoyed using IoTLink since it offers a new alternative approach for developing IoT prototypes. The user interface and the easiness it provides are appreciated, particularly by the less experienced participants who are excited to try new technologies. The following statements are quoted from the participants' comments when they provided the positive aspects of IoTLink:

“This tool is useful to developers. This tool abstracts all the complexity of modeling and help us to do programming of Internet of Things very quickly. It is a good idea.”

“To have a diagram divided into 4 sequential blocks make it very easy to visualize how the data flow in the system. It was way more intuitive than learning how to use Java API, especially because you can see all the relevant

properties that you have to set when you click on a component. Modeling with a drag and drop approach is faster than writing raw code.”

“It is natural and intuitive to establish connections and communication between the elements. Much better than programming with pure Java code.”

However, many participants pointed out that the documentation and tooltips should be improved since they were not utilized optimally. They would like to have a quick-start and examples of creating a prototype from scratch to finish. Furthermore, the participants would like to have clearer error messages and clearer hints to recover from their mistakes. Another participant mentioned that although using IoTLink is quick, it is not very flexible. If the users would like to do something that is not foreseen in the predefined components, it will probably become very complicated since the users are required to modify the generated Java code without too much knowledge how the code is structured.

Overall, the participants were satisfied using IoTLink to solve the given tasks, and they could imagine themselves using it to integrate various IoT rapidly. This shows that MDD has a huge potential for IoT development. However, IoTLink is not ready to be used in the production environment and needs to be further developed to improve the usability as well as incorporating more features to handle different use cases in IoT development.

8.3 Summary and conclusion

IoTLink evaluation was done iteratively to examine its design and implementation at every stage of the development. In the initial phase IoTLink was evaluated with a limited number of people to keep the efforts low while still able to gather the users’ feedback as well as identify up to 85% of the usability issues (Nielsen & Landauer, 1993). The results were very useful to influence the design decisions taken on the further iterations. The results show that DSL was preferred and rated higher by the participants compared to general purpose modeling language. It became clear that the participants struggle with the complexity of UML for modeling small applications. The extensive flexibility offered by UML causes rather large overhead for designing small applications. Additionally, many usability issues and users’ requirements were collected and considered when refining the design of the proposed architecture as well as IoTLink.

The evaluation of IoTLink’s semantic discovery shows a high acceptance from three different user roles including the device developers who provide the device description, the application developers who access the devices based on their requirements, and the administrator that maintain the knowledge of the Discovery Manager. The feedback from the users reveals that they prefer to reuse well-known ontology such as SSN Ontology in order support semantic interoperability with other systems. Moreover, building taxonomy between devices should be achieved automatically to extend the discoverability of the devices based on their semantics.

Based on the feedbacks collected in the initial evaluation, the usability and features of IoTLink were significantly improved, and SSN ontology is used as the backend of the Discovery Manager. The second iteration of IoTLink was evaluated again with 24 users to understand if IoTLink could improve IoT developments and supports inexperienced

developers. The final evaluation consists of two parts which investigate the users' comprehension to the IoTLink diagrams as well as its effectiveness and efficiency compared to the common approach for developing IoT through middleware and textual language.

The results of the first-phase of the study provides insights on how well developers could understand object-oriented artifacts that are presented on different presentations. For simple relations between classes, there were no significant differences in the time required by the participants to analyze UML, IoTLink, and textual source code. However, UML was able to minimize the number of mistakes significantly. The higher number of mistakes on the source code and IoTLink was caused by the cardinality which was not explicitly stated. As the diagram becomes more complex, the users could understand well-structured diagram faster than source code. The time required to analyze unstructured diagram could be longer than analyzing source code since the users have to find a starting point by scanning the whole diagram, then trace the relations.

The second phase of the study provides an insight on how well IoTLink is able to support both the experienced and less experienced developers in IoT software development. IoTLink provides specific functionalities for IoT developments which help developers to gain confidence and decide quickly compared to generic programming language. Additionally, using code generations IoTLink is able to optimize the steps required during the implementation. On the other hand, when the number of steps is comparable, no significant difference between IoTLink and Java could be found. The average time required was even slightly slower than Java. When comparing experienced and inexperienced developers, IoTLink was able to reduce their time to solve the tasks significantly, but neither group was influenced by IoTLink differently. However, IoTLink was significantly more effective for reducing the number of mistakes when it was used by the less experienced developers since the more experienced developers were already able to minimize their mistakes regardless of the tools they use.

Moreover, the participants' perceive IoTLink very positive and could imagine themselves working with such tools for their daily work. However, some usability issues should be fixed, and the documentation, as well as tooltips, must be improved. IoTLink was well received by inexperienced and experienced developers with different level of experiences. It shows that the model driven approach has a huge potential for enabling rapid IoT prototyping. Some steps can be automated by model driven tool by taking advantage of code generation. This could not only reduce the development time but also keep the quality of the code consistent.

Chapter 9.

Conclusion and Future Work

This chapter concludes the thesis with the summary of the contributions and an outline of the future work to extend the knowledge, gained in this research work. The contributions of this work focus on answering the research questions elaborated in section 1.2 which can be summarized as follows:

1. To what extent the internet of thing architectures can be abstracted?
2. How could applications discover “Things”?
3. To what extent a Model-Driven Development approach could support IoT prototyping?

9.1 Summary of contributions

IoT theoretical ground

In the search of answers related to research the question #1, the author investigates the evolution of IoT research and development from the initial phase when the term was first introduced up to now. IoT middlewares have been developed with extensive features including device abstractions, resource management, addressing, and security. Unfortunately, none of the middleware has been accepted widely as the standard middleware for the IoT. Facing this diversity, IoT-A project tries to provide a standardized IoT architecture by unifying the approaches taken by industry and IoT research projects (IoT-A, 2013). IoT-A concludes that IoT should contain physical objects that can be uniquely identified, have properties that can be measured by sensors, and some are able to perform actions through actuators. Moreover, its architectures describe the relations between physical devices, different communication patterns, as well as device abstractions that can be done on the IoT service level and virtual entity level. Surveying the communication patterns, the IoT could communicate using (1) a direct communication when all parties support the same communication protocols or (2) facilitated by gateways that translate the messages back and forth between incompatible parties. The findings from the literature show that IoT definition must explain the “Thing” and “Internet” aspects of IoT. It should explain to what extent physical objects are considered part of IoT and how they interact with other objects as well as the applications. Based on this consideration, the author defines IoT from a rather technical perspective as follows:

IoT is a “vision” of the world in which, physical objects could seamlessly communicate with other physical or virtual objects through electronic media. The objects that do not have communication capabilities or have incompatible communication capabilities could be represented by virtual objects that act as their proxies. The proxies reflect their actual states, contain their information, and able to interact with other virtual entities on the behalf of the physical objects that they represent. The proxies are responsible for ensuring interoperability between things and therefore must provide translation services for the incompatible technologies. These proxies are created with certain goals. Therefore, they may not expose all possible information about the physical objects nor have the ability to bridge all possible services that the physical objects may have. Virtual objects may be created to expose only relevant information and able to represent a set of functions to fulfill certain goals. In the opposite, a proxy may represent a composition of physical objects as a virtual object, which sometimes required to encapsulate the complexity of several physical objects.

Conceptual architecture and implementation of model driven IoT development tool

As the number of connected things increases rapidly, the author believes that in the future, inexperienced developers and end users will contribute significantly to IoT developments and therefore must be supported by appropriate tools that are able to encapsulate the complexity of IoT technology. The model driven approach shows a promising approach to reach this goal. Based on the analysis on current IoT architectures, the author identifies the required components to enable MDD in IoT developments. These components are summarized in a five-layer architecture which has been evaluated in (Pramudianto, Rusmita, et al., 2013). Following the proposed architecture pattern, this work designed and implement a model driven tool, named IoTLink. IoTLink aims to support inexperienced developers by combining the strength of MDD and FBP. The author decided to provide the users with a visual editor for defining the application in platform-independent models since visual notations could enable inexperienced developers as the findings in section 4.2.3 shows. Moreover, the author decided to use Java for implementing the platform-specific model since Java offers extensive open source components for software developments that ease the required efforts to implement IoTLink. Additionally, Java allows us to implement artifacts that can be directly compiled or modified by experienced developers when needed.

As the instantiations of the concept, the author implemented several components for each layer. The input layer currently implements well-known communication protocols such as SOAP and RESTful services, MQTT, and communication via LinkSmart middleware. Moreover, it supports domain-specific technology such as connection to OPC server (a standard middleware in building and industrial automation), and connection to an Arduino, which is often used for IoT hardware prototyping. The input components allow the developers to communicate with physical devices as well as other data sources without having to deal with the complexity of different specific technologies. As instances of the fusion layer to enable sensor data processing, IoTLink implements specific sensor fusion algorithms such as moving average filter and integrates a CEP engine, namely Esper. Esper can be configured

using EPL. Esper allows aggregation and grouping using “group by” and “having” clause, which is useful to perform calculations of values based on particular criteria.

IoTLink implements the domain model layer by allowing the virtual objects being modeled visually, in which each virtual object represents one or an aggregation of physical objects. These virtual objects can then be linked to the sensors that observe them and actuators that perform the action on their behalf. In addition, to expose the virtual objects to external applications, IoTLink introduces several output components that can be used to serialize the state of the physical objects through different data format and network protocols e.g., database entries, REST, and SOAP based services. Allowing applications to get the state changes instantly, IoTLink implements an event publisher that pushes any state changes to an MQTT broker. The MQTT broker then notifies the applications. Additionally, it implements a connection to a Drools rule engine that can be configured to react based on the states of the physical objects. This allows developers to experiment with diverse business logic rapidly. The Drools component can be configured to poll the rules from a central repository called Guvnor (Amador, 2012), which allows developers to deploy and change rules rapidly even at runtime.

Design concept and implementation of IoT semantic discovery

As the research question #2 states, in the IoT development, the developers must be able to find devices that are shared between several applications particularly when they are shared within the internet. Therefore, IoT discovery is one of the most essential components in IoT developments.

In the initial phase of IoTLink development, the author found out that diverse terms are often used to describe the same devices or used as keywords to find them. To solve this problem, this work analyzes the semantics between lexical terms to estimate if the same devices are described with diverse terms. The knowledge about the related terms are stored in the ontology to simplify the required taxonomy reasoning. The ontology schema follows the SSN ontology that covers comprehensive aspects of sensor observations. In addition, the schema was extended by the author to include concepts for actuators and services as introduced by IoT-A’s domain model. Using the proposed approach, the Discovery Manager is able to estimate the taxonomy of devices based on the terms used to describe the device types automatically. It learns the relations between lexical terms from WordNet dictionary that contains relations such as synonym, hyponym, hypernym, and troponym. Hyponym and hypernym indicate subset and superset relations between nouns. Troponym indicates subset relations between verbs. Having a taxonomy of devices, allow developers and applications to work optimistically by taking advantage of the available devices that have most required capabilities. This is quite useful in a dynamic IoT environment where the availability of the devices cannot be known in advance.

Generic dictionaries, such as WordNet, do not contain the semantics used in specific application domains, e.g., terms which are not officially synonymous could be used interchangeably. Therefore, IoTLink allows an administrator to edit the relations between device categories and capabilities as well as adding domain specific relations between terms through a graphical user interface.

The use of semantic and dictionary to solve the terms diversity issue improves the discoverability of devices, particularly in dynamic environments where devices may enter and leave the network dynamically. This approach can also be used to solve language diversity used to describe and find IoT devices. However, a domain specific dictionary is required should be used to improve the effectiveness of this approach. In addition, using a well-known ontology such as SSN ontology to describe IoT systems could increase the interoperability with other systems that are able to process the metadata automatically and understand how these systems work.

Evaluations of model driven approach in IoT development

The research question #3 tries to find out how helpful model driven approach for rapid IoT prototyping is. The results of the studies confirms that MDD approach is able to accelerate the development time, reduce mistakes of inexperienced developers, and ensure the consistency of the code quality. MDD offers several advantages compared to conventional programming languages. First, when MDD is applied using a domain specific modeling language, it offers a level of abstraction that is closer to the problem space, which makes it is easier for the developers to map their solutions to the modeling language. Choosing the modeling language is a very crucial step to ensure the usability of MDD since the modeling language determines how easy the developers are able to express their solutions that they have developed in their mental model. Moreover, it can be used to communicate the system design to different stakeholders. Consequently, in practice the effectiveness and efficiency of MDD are influenced significantly by the functions and usability of the modeling language and the supporting tools.

Although general purpose modeling language, such as UML provides extensive features, its complexity presents a real challenge for different stakeholders, even the ones with a computer science background. Conversely, DSL which are tailored with notations and terms, known in specific application domains offers a better usability (France & Rumpel, 2005; Gronback, 2009; Pramudianto, Rusmita, et al., 2013). In order to ensure the usability of DSL for IoT development, the modeling language must capture the concepts used in IoT. The author believes that IoT-A domain model (IoT-A, 2013) provides the necessary concepts for designing the DSL for IoT. It presents generic IoT concepts such as virtual entities, sensors, and actuators which are commonly used in IoT architectural patterns (Mattern, 2003; Kortuem et al., 2010; Bernini & Tisato, 2012).

In addition, the first part of the study shows that visual languages do not guarantee a better comprehension compared to textual programming languages. Visual diagrams can be understood faster when they have a clear structure which allows users to scan the diagram systematically in a consistent direction. This is confirmed by a study that could not find a significant improvement of Flowchart to textual programming language in terms of the comprehensibility (Ramsey et al., 1983). This might be caused by the structure of the flowchart, which requires developers to scan the diagram back and forth, e.g., when a decision logic is used. In contrast, IoTLink diagram offers a sequential structure which can be scanned faster and easier by the participants.

Another advantage of MDD is, it offers the possibility to automate programming tasks by generating the necessary code based on the abstract model. The developers are only required to define their solution in high-level abstract models which correspond to the domain. The

amount of time that could be saved by code generation depends on the amount of the abstraction level that the models expose and the amount of details that are generated. On the other hand, generating a more detailed code based on the abstract models reduces the ability of the developers to define detailed specifications, which make MDD less customizable compared to conventional general purpose programming languages. In contrast, limiting the freedom of the inexperienced developers is not necessarily bad for them. With a limited set of features, the developers are able to learn the tool faster since they are not overwhelmed by unnecessary information. Therefore, they tend to make less mistakes compared to using general purpose languages. This is confirmed by the result of the final study, where IoTLink was able to reduce up to 57% of mistakes of the less experienced developers compared to the Java development.

When designing the modeling language, it is quite essential that the use cases and development scenarios of the targeted domain being considered. The author identifies the following goals can typically be found in rapid IoT prototyping:

- Integration of different sensor and actuator technologies.
- Interoperability between different sensor & actuator technology.
- Understanding and modeling the application domain.
- Transforming sensor data into contextual information and linking the data stream to the processing modules.
- Designing persistence model.
- Correlating database model and domain model.

These processes must be done repetitively although they resemble similar activities. Thus, through MDD approach these activities could be automatically generated based on higher abstraction models.

9.2 Threats to validity

Construct validity: to measure the understandability of different notations the time to read and errors are measured. The notations did not show identical content to avoid the learning the effects. Although the author has designed the question with a similar level of complexity, the differences between the content and questions may still affect the time and errors made by the participant.

Internal validity: The study has been designed to examine the IoTLink's design and performance within a controlled experiment and case studies to ensure the validity of the result applies in the real world. However, in the real world IoT developments involves a broad range of use cases and scenarios that might not have been represented within the tasks and case studies.

External validity: The developers involved in the study work in the scientific communities such as research institutes, master and bachelor students, and university employees. There were only two participants that have worked for private initiatives. Therefore, the results might be biased towards the research environment.

Conclusion validity: The main threat to the conclusion whether MDD could accelerate IoT software development would be if the set of scenario and use cases are not foreseen by IoTLink. The developers would need to modify the generated code quite extensively which may take time longer than if they work without MDD tools.

9.3 Future works

An integrated development platform for IoT prototyping

IoT model driven tools could be extended by allowing developers to model the interaction between components on hardware boards (e.g., sensors and actuators on Arduino), and the interactions between distributed devices and applications on the local network as well as on the internet. Features required on the device level could include generating the necessary firmware that obtains data from the hardware components and transmit them through communication channels. The tools could also be integrated with user interface frameworks such as Sencha⁷¹, which is able to generate web applications for different devices. This holistic approach will allow developers to experiment rapidly with IoT prototyping hardware, integrate distributed devices and applications, and display the information to the users within a single development environment.

Model checking and verifications is an area that needs to be further developed in order to allow developers testing their models by simulating the systems rapidly and get live feedbacks before they generate the code. Being able to verify the correctness of the solutions could increase the developers' confidence when using model driven tools. Verifying the correctness of IoT models should allow the developers to trace the flow of the sensor stream as well as tracing how the data is being transformed in each component. In addition, debugging distributed systems has been always challenging since, most tools are not able to trace messages that are transmitted across distributed systems without using various tools with complicated configurations. Therefore, the future IoT debugging tools should allow remote debugging and provide an abstraction which encapsulate the complexity of distributed systems.

Long term study of model driven approach

Although the study conducted in this work has shown a promising results of applying MDD in IoT software development, until now there has been a lack of investigation on how MDD could affect a large development process in terms of developers' productivity and code maintainability. Therefore, there is a need of conducting a long-term study, particularly in large IoT projects.

Distributed Discovery Management

⁷¹ <http://www.sencha.com/> (retrieved on Oct 4, 2014)

As a continuation of this work, the Discovery Manager could be extended by considering a distributed peer-to-peer architecture in which several Discovery Managers exist in different local networks and can be connected through the Internet. They must be able to exchange information of the available devices so that each Discovery Manager has the overall view of all devices. Additionally to improve the usability the Discovery Manager could be improved by allowing developers and end users to use natural language to find devices based on their semantic properties. This requires the Discovery Manager to perform machine translation from human language to SPARQL (Kaufmann et al., 2007; C. Wang et al., 2007). Moreover, when sharing IoT resources for different applications, one should consider that several applications may have higher criticality than the others. Allowing all applications to access the available resources will affect the performance of the resources which in turn affect the performance of the applications. Therefore, it requires a system which is able to manage the access to the IoT resources and guarantee that the applications are able to meet their critical requirements (Takalo-Mattila et al., 2014).

Secured IoT as a Service

Recently there are several discussions to provide IoT as a service (IoTaaS) (Christophe et al., 2011; Mingozi et al., 2013). In this regards, the customers could lease IoT infrastructure to the service providers for obtaining the required data and information produced by the IoT. This business model allows the costs to build and maintain IoT infrastructure being shared between several users, which is able to reduce the initial investment costs from the user perspective. For enabling such as scenario, authentication, access, and accounting (AAA) must be supported by the IoT platforms. For instance, IoTLink and similar tools could allow developers to control the user access to specific data that are published through Web Services. Additionally, the produced IoT systems should provide an accounting ability to the data that are accessed by the customers. For instance, when it is used to expose data for traceability purposes, the data owners should be able to monitor how much data is being accessed by their partners. Moreover, they should be able to provide different access levels for the actors in the food production chain to protect any sensitive data being abused by unauthorized actors.

9.4 Outlook of IoT developments

The author anticipates that the future research and development in IoT will still aim at finding ways for making physical objects become smarter, more connected and able to cooperate autonomously to reach more demanding goals. In terms of connectivity, IoT lacks a strong market leadership, which is able to drive the standardization between the IoT manufacturers in diverse domains. Moreover, the broad extent of IoT application domains requires the technology to consider a broad range of requirements that may contradict each other. Therefore, the author believes that technology fragmentations will always exist, particularly between various application domains. The main question for the IoT designers is how to minimize the efforts to integrate these heterogeneous technologies, as well as finding compromise for the contradicting requirements. For instance, how to keep the messages exchanged for the low powered wireless devices small to maximize the battery life while ensuring that the message format does not decrease the developers' productivity. They also

have to consider a strategy that allows devices to interact without interfering their ability to fulfil different critical requirements.

Furthermore, the increase of processing power on embedded systems will allow us to delegate more intelligence and decision making on the device level, which distribute the computing load between the nodes and avoid a single point of failure. However, such a highly distributed architecture requires more engineering efforts to share and consolidate knowledge between devices as well as maintaining connectivity between them. Consequently, IoT middleware should not only be responsible for establishing communications, but also managing the cooperation between distributed software and hardware resources similar to what operating systems do to manage computing resources. Therefore, IoT platforms should act as distributed operating systems which provide an abstraction for accessing heterogeneous resources and manages the access to them. Having such a comprehensive platform enables the users to create IoT ecosystems which consolidate heterogeneous device and application developments.

The interaction between IoT technology and cloud services have been seen as an evolutionary step in the development of both areas. The IoT could take advantage of cloud technology to overcome the hardware limitations of small and inexpensive embedded systems. For instance, analyzing massive image data could be very challenging for small microcontrollers. By delegating the complex analytics tasks to the cloud services, small and inexpensive embedded devices only receive the necessary information from the cloud services which they could use to make a decision. For instance, image recognition services have been offered by Google (Google Goggles⁷²), Amazon (firefly⁷³), and Microsoft (Bing Vision⁷⁴) which allow mobile phones to recognize physical objects using image recognition processes executed on the cloud. Once the object is recognized, the server sends the information about the product to the smartphones. Weather forecast and warning is another application which process satellite and other sensor data as well as historical data in the cloud, and let third party applications use the information to inform the users or to take a decision upon it.

Cloud services could also take advantage of the data collected by IoT. There exist participatory sensing applications, in which communities contribute data to a cloud service in order to obtain a holistic view of the situation. For instance, traffic report applications often use navigational data of many users to report traffic jams. Waze⁷⁵ is an application that relies on users' manual input to collect situational awareness of various street conditions, including traffic jams and other hazards on the road. As IoT becomes more affordable for different user groups, the amount of data generated by them will grow exponentially, which requires a different approach for analyzing them. Thus, the author believes that big data processing will play a significant role in making IoT more intelligent and able to extract a more holistic information that can support the decision makers to assess the situation better.

IoT will also have negative impacts on privacy and security when system designers do not consider them carefully. Having more connected devices and sensors will increase the risks of

⁷² <https://support.google.com/websearch/answer/166331> (retrieved on Oct 4, 2014)

⁷³ <https://developer.amazon.com/public/solutions/devices/fire-phone/overview/firefly-sdk-for-fire-phone> (retrieved on Oct 5, 2014)

⁷⁴ <http://www.neowin.net/news/bing-features-for-windows-phone-8-explained> (retrieved on Oct 5, 2014)

⁷⁵ <https://www.waze.com/> (retrieved on Oct 5, 2014)

having unauthorized access to private information. Another problem is how the users could verify the authenticity of data and information that they received from IoT. For instance, sending false GPS location has been investigated (Humphreys et al., 2008) and could theoretically be used to hijack planes and ships⁷⁶. Therefore, there is a need for an in-depth research how to mitigate and eliminate these threats to security and privacy while still able to take advantage of seamless connectivity that IoT offers.

⁷⁶ <http://www.foxnews.com/tech/2013/07/26/exclusive-gps-flaw-could-let-terrorists-hijack-ships-planes/>
(retrieved on Oct 6, 2014)

Bibliography

- Aalamifar, F., Hassanein, H. S., & Takahara, G. (2012). Viability of powerline communication for the smart grid. *Proceedings of the 26th Biennial Symposium on Communications (QBSC)*, (pp. 19-23), IEEE. doi:10.1109/QBSC.2012.6221343
- Acquaviva, A., Blaso, L., Dalmasso, D., Del Giudice, M., Fracastoro, G., Verso, V. L., et al. (2012). From historical buildings to smart buildings via middleware and interoperability. *Proceedings of the 14th International Conference on Computing in Civil and Building Engineering, ISCCBE*, Moscow, Russia., Retrieved from http://www.iccbe.ru/paper_long/0428paper_long.pdf
- Acquaviva, A., Blaso, L., Dalmasso, D., Giudice, M., Fracastoro, G., Verso, V. L., et al. (2012). From Historical Buildings to Smart Buildings via Middleware and Interoperability. *Proceedings of the 14th International Conference on Computing in Civil and Building Engineering, ISCCBE.*, (pp. 1-8), Moscow, Russia. ISCCBE, Retrieved from http://www.iccbe.ru/paper_long/0428paper_long.pdf
- Adi, A., Botzer, D., Nechushtai, G., & Sharon, G. (2006). Complex event processing for financial services. *Proceedings of the Services Computing Workshops (SCW)*, (pp. 7-12), IEEE
- Ahamed, S. I., Zulkernine, M., & Anamanamuri, S. (2006). A dependable device discovery approach for pervasive computing middleware. *Proceedings of the First International Conference on Availability, Reliability and Security (ARES)*, (pp. 8), IEEE. doi:10.1109/ARES.2006.5
- Akkiraju, R., Goodwin, R., Doshi, P., & Roeder, S. (2003). A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. *Proceedings of the IIWeb*, (pp. 87-92), Retrieved from <http://www.isi.edu/info-agents/workshops/ijcai03/papers/Akkiraju-SemanticUDDI-IJCA%202003.pdf>
- Alliance, O. (2007). OSGi service platform, core specification, release 4, version 4.1 *OSGi Specification*.
- Alliance, Z. (2009). Zigbee RF4CE Specification Version 1.00.
- Amador, L. (2012). Drools developer's cookbook. In (pp. 97-118): Packt Publishing Ltd.
- Anderson, D., Shanley, T., & Budruk, R. (2004). *PCI express system architecture*: Addison-Wesley Professional.
- Apache, C. (2009). *An Open Source Service Framework*. Retrieved from <http://cxf.apache.org/docs/index.html> (last accessed October 30, 2014).
- Ashton, K. (2009). *That 'internet of things' thing in the real world, things matter more than ideas*. Retrieved from <http://www.rfidjournal.com/article/print/4986> (last accessed June 20, 2014).
- Association, K. (2004). *KNX Specification Version 1.1*.
- Atemezing, G., Corcho, O., Garijo, D., Mora, J., Poveda-Villalón, M., Rozas, P., et al. (2013). Transforming meteorological data into linked data. *Semantic Web*, 4(3), 285-290. doi:10.3233/SW-120089
- Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15), 2787-2805.
- Avila, B. (2013). *Device Proxy Discovery and Ontology-Based Domain Model Middleware*. RWTH Aachen University, Aachen, Germany.
- Bali, M. (2009). Basic Rule. In *Drools JBoss Rules 5.0 Developer's Guide* (pp. 17-22): Packt Publishing Ltd.
- Bandara, A., Payne, T., De Roure, D., Gibbins, N., & Lewis, T. (2008). A pragmatic approach for the semantic description and matching of pervasive resources. In *Advances in Grid and Pervasive Computing* (pp. 434-446): Springer.
- Bandyopadhyay, D., & Sen, J. (2011). Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless Personal Communications*, 58(1), 49-69. doi:10.1007/s11277-011-0288-5

BIBLIOGRAPHY

- Bandyopadhyay, S., & Bhattacharyya, A. (2013). Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. *Proceedings of the Computing, Networking and Communications (ICNC), 2013 International Conference on*, (pp. 334-340), IEEE
- Bandyopadhyay, S., Sengupta, M., Maiti, S., & Dutta, S. (2011a). Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3), 94-105.
- Bandyopadhyay, S., Sengupta, M., Maiti, S., & Dutta, S. (2011b). A survey of middleware for internet of things. In *Recent Trends in Wireless and Mobile Networks* (pp. 288-296): Springer.
- Barchetti, U., Bucciero, A., De Blasi, M., Mainetti, L., & Patrono, L. (2010, 17-19 June 2010). RFID, EPC and B2B convergence towards an item-level traceability in the pharmaceutical supply chain. *Proceedings of the 2010 IEEE International Conference on RFID-Technology and Applications (RFID-TA)*, (pp. 194-199), IEEE doi:10.1109/RFID-TA.2010.5529939
- Barnaghi, P., Compton, M., Corcho, O., Castro, R. G., Graybeal, J., Herzog, A., et al. (2011, June, 29 2011). *Incubator Report* [MediaWiki]. Retrieved from http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report#Stimuli-Centered (last accessed June 13, 2014).
- Baronti, P., Pillai, P., Chook, V. W., Chessa, S., Gotta, A., & Hu, Y. F. (2007). Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards. *Computer communications*, 30(7), 1655-1695.
- Becker, M. (2012). Interoperability Case Study: The Bar Code/UPC. *Berkman Center Research Publication*, (2012-4).
- Bender, K. (1993). *Profibus: the fieldbus for industrial automation*: Prentice-Hall, Inc.
- Bernini, D., & Tisato, F. (2012). Spaces of Things: Application-Level Abstractions for the Internet of Things. *International Transactions on Systems Science and Applications*, 8, 99-112.
- Berrueta, D., Polo, L., & Álvarez, J. (2008). *Measurement Units Ontology*. Retrieved from <http://idi.fundacionctic.org/muo/muo-vocab.html> (last accessed December 3, 2014).
- Bishop, R. H. (2007). *Mechatronic systems, sensors, and actuators: fundamentals and modeling*: CRC press.
- Blackstock, M., & Lea, R. (2012). IoT mashups with the WoTKit. *Proceedings of the 3rd International Conference on the Internet of Things (IOT)*, (pp. 159-166), IEEE
- Bose, I., & Pal, R. (2005). Auto-ID: managing anything, anywhere, anytime in the supply chain. *Communications of the ACM*, 48(8), 100-106.
- Botts, M., & Robin, A. (2007). *OpenGIS sensor model language (SensorML) implementation specification*.
- Brandl, D. (2002). Business to manufacturing (B2M) collaboration between business and manufacturing using ISA-95. *Revue de l'Electricité et de l'Electronique*(8), 46-52.
- Brauer, J. R. (2006). *Magnetic Actuators and Sensors*: Wiley. Retrieved from <http://books.google.de/books?id=Wwk1EeZubdUC>
- Brizzi, P., Conzon, D., Pramudianto, F., Paralic, M., Jacobsen, M., Pastrone, C., et al. (2013, June, 23 – 27). The ebbits platform: leveraging on the Internet of Things to support meat traceability. *Proceedings of the EFITA 2013*, Torino, Italy. CIGR (to appear)
- Brock, D. L. (2001). The electronic product code (epc) *Auto-ID Center White Paper MIT-AUTOID-WH-002*.
- Brock, D. L., Milne, T. P., Kang, Y. Y., & Lewis, B. (2001). *The physical markup language*: Auto-ID Center
- Brogi, A., Popescu, R., Gutiérrez, F., López, P., & Pimentel, E. (2008). A service-oriented model for embedded peer-to-peer systems. *Electronic Notes in Theoretical Computer Science*, 194(4), 5-22.
- Casier, H., Casier, H., Steyaert, M., & van Roermund, A. H. (2008). *Analog circuit design: sensors, actuators and power drivers; integrated power amplifiers from wireline to RF; very high frequency front ends*: Springer Publishing Company, Incorporated.
- Ceri, S., Daniel, F., Matera, M., & Facca, F. M. (2007). Model-driven development of context-aware Web applications. *ACM Transaction on Internet Technology*, 7(1), 2. doi:10.1145/1189740.1189742
- Cetina, C., Serral, E., Munoz, J., & Pelechano, V. (2007). Tool support for model driven development of pervasive systems. *Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, (pp. 33-44), IEEE

BIBLIOGRAPHY

- Chapuis, C., Roustit, M., Bal, G., Schwebel, C., Pansu, P., David-Tchouda, S., et al. (2010). Automated drug dispensing system reduces medication errors in an intensive care setting. *Critical care medicine*, 38(12), 2275-2281.
- Charette, R. N. (2009). *This Car Runs on Code*. Retrieved from <http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code> (last accessed April 23, 2014).
- Chen, H., Cong, T. N., Yang, W., Tan, C., Li, Y., & Ding, Y. (2009). Progress in electrical energy storage system: A critical review. *Progress in Natural Science*, 19(3), 291-312.
doi:<http://dx.doi.org/10.1016/j.pnsc.2008.07.014>
- Chen, M., Wan, J., & Li, F. (2012). Machine-to-machine communications: Architectures, standards and applications. *KSII Transactions on Internet & Information Systems*, 6(2), 480.
- Chen, Y.-K. (2012). Challenges and opportunities of internet of things. *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, (pp. 383-388), IEEE
- Cheng, J., Cline, M., Martin, J., Finkelstein, D., Awad, T., Kulp, D., et al. (2004). A knowledge-based clustering algorithm driven by gene ontology. *Journal of biopharmaceutical statistics*, 14(3), 687-700.
- Cherbakov, L., Bravery, A., Goodman, B. D., Pandya, A., & Baggett, J. (2007). Changing the corporate IT development model: Tapping the power of grassroots computing. *IBM Systems Journal*, 46(4), 1-20.
- Choi, J., Shin, D., & Shin, D. (2005). Research and implementation of the context-aware middleware for controlling home appliances. *IEEE Transactions on Consumer Electronics*, 51(1), 301-306.
- Christophe, B., Boussard, M., Lu, M., Pastor, A., & Toubiana, V. (2011). The web of things vision: Things as a service and interaction patterns. *Bell Labs Technical Journal*, 16(1), 55-61.
- Cilibrasi, R. L., & Vitanyi, P. M. (2007). The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3), 370-383.
- Establishing a system for the identification and registration of ovine and caprine animals and amending Regulation (EC) No 1782/2003 and Directives 92/102/EEC and 64/432 (2003).
- Committee, I. S. (2009). *ISA100. 11a, "Wireless Systems for Industrial Automation: Process Control and Related Applications"*: Technical Report, Research Triangle Park, North Carolina.
- Compton, M., Barnaghi, P., Bermudez, L., Garcia-Castro, R., Corcho, O., Cox, S., et al. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17, 25-32.
- Condori-Fernandez, N., Panach, J. I., Baars, A. I., & Pastor, T. V. (2013). An empirical approach for evaluating the usability of model-driven tools. *Sci. Comput. Program.*, 78(11), 2245-2258.
doi:10.1016/j.scico.2012.07.017
- Consortium, U. (2005). UPC language specifications v1. 2.
- Couto, F. M., & Silva, M. J. (2011). Disjunctive shared information between ontology concepts: application to Gene Ontology. *J. Biomedical Semantics*, 2, 5.
- Cox, B. (1995). No silver bullet revisited. *American Programmer Journal*, 17, 85-92.
- Cox, B. J. (1990). Planning the software industrial revolution. *IEEE software*, 7(6), 25-33.
- Crockford, D. (2006). *JSON: The fat-free alternative to XML*. Retrieved from <http://www.json.org/fatfree.html> (last accessed December 2, 2014).
- Curtis, B., Sheppard, S. B., Kruesi-Bailey, E., Bailey, J., & Boehm-Davis, D. A. (1989). Experimental evaluation of software documentation formats. *Journal of Systems and Software*, 9(2), 167-207.
- Day, J. D., & Zimmermann, H. (1983). The OSI reference model. *Proceedings of the IEEE*, 71(12), 1334-1340.
- de Souza, L., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., & Savio, D. (2008). Socrades: A web service based shop floor integration infrastructure. *The Internet of Things*, 50-67.
- DeMichiel, L., & Keith, M. (2006). Java Persistence API JSR (Vol. 220).
- Directive, E. (2010). Directive 2010/75/EU of the European Parliament and of the Council on industrial emissions (Industrial Emissions Directive).
- Duquennoy, S., Grimaud, G., & Vandewalle, J.-J. (2009). The Web of Things: interconnecting devices with high usability and performance. *Proceedings of the International Conference on Embedded Software and Systems (ICCESS'09)*, (pp. 323-330), IEEE

BIBLIOGRAPHY

- E.1, U. (2014). *Internet of Things*. Retrieved from <http://ec.europa.eu/dgs/connect/en/content/network-technologies-smart-networks-and-novel-architectures> (last accessed August, 1 2014).
- Eisenhauer, M., Jahn, M., Pramudianto, F., Sabol, T., & Hreno, J. (2011). Towards a generic Middleware for developing Ambient Intelligence Applications. *Proceedings of the 2nd Workshop on eeBuildings Data Models at CIB W078–W102*, (pp. 26-28), Sophia Antipolis, France, European Communities
- Eisenhauer, M., Rosengren, P., & Antolin, P. (2009). A development platform for integrating wireless devices and sensors into ambient intelligence systems. *Proceedings of the Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON)*. (pp. 1-3), IEEE
- Emerson, D., Kawamura, H., & Matthews, W. (2007). *Plant-to-Business (P2B) Interoperability Using the ISA-95 Standard* (0911-8977).
- Emmerson, B. (2010). M2M: the Internet of 50 billion devices. *WinWin Magazine*, 19-22.
- Ericsson-Australia. (2010). *Towards 50 billion connected devices*. Retrieved from http://www.ericsson.com/au/res/region_RASO/docs/2010/ericsson_50_billion_paper.pdf
- ETSI, T. (2011). 102 690:" Machine-to-Machine communications (M2M). *Functional architecture*.
- Evans, D. (2011a). *The internet of things: How the next evolution of the internet is changing everything*: Cisco Internet Business Solutions Group (IBSG).
- Evans, D. (2011b). *The Internet of Things: How the Next Evolution of the Internet is Changing Everything*.: Cisco Internet Business Solutions Group (IBSG). Retrieved from http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
- Fan, Z., Kalogridis, G., Efthymiou, C., Sooriyabandara, M., Serizawa, M., & McGeehan, J. (2010). The new frontier of communications research: smart grid and smart metering. *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, (pp. 115-118), ACM
- Fan, Z., Kulkarni, P., Gormus, S., Efthymiou, C., Kalogridis, G., Sooriyabandara, M., et al. (2012). Smart grid communications: Overview of research challenges, solutions, and standardization activities. *IEEE Communications Surveys & Tutorials*, 15(1), 21 - 38.
- Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 379-383.
- Fellbaum, C. (1999). *WordNet*: Wiley Online Library.
- Ficco, M., & Romano, L. (2011). A generic intrusion detection and diagnoser system based on complex event processing. *Proceedings of the First International Conference on Data Compression, Communications and Processing (CCP)*, (pp. 275-284), IEEE
- Fielding, R. T. (2000a). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.
- Fielding, R. T. (2000b). *Chapter 5: Representational State Transfer (REST) in Architectural Styles and the Design of Network-based Software Architectures* University of California, Irvine, Irvine, California.
- Fink, J. K. (2012). *Polymeric Sensors and Actuators*: John Wiley & Sons.
- Foundation, E. (2007). *Eclipse Graphical Modeling Framework (GMF)*. Retrieved from <http://eclipse.org/modeling/gmp/> (last accessed December 2, 2014).
- France, R., & Rumpe, B. (2005). Domain specific modeling. *Software and Systems Modeling*, 4(1), 1-3.
- France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. *Proceedings of the 2007 Future of Software Engineering*, (pp. 37-54), IEEE Computer Society
- Frey, G., & Litz, L. (2000). Formal methods in PLC programming. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, (pp. 2431-2436), Nashville, TN. IEEE. doi:10.1109/ICSMC.2000.884356
- Galli, S., Scaglione, A., & Wang, Z. (2011). For the grid and through the grid: The role of power line communications in the smart grid. *Proceedings of the IEEE*, 99(6), 998-1027.
- Gao, G., Zhang, H., & Li, L. (2013). A reliable multipath routing strategy for wireless mesh networks using subgraph routing? *Journal of Computational Information Systems*, 9(5), 2001-2008.
- Gardner, J. W., Varadan, V. K., & Awadelkarim, O. O. (2001). *Microsensors, MEMS, and smart devices* (Vol. 1): Wiley Online Library.

- Ghatikar, G. (2010). Open automated demand response technologies for dynamic pricing and smart grid. *Lawrence Berkeley National Laboratory*.
- Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., & Razafindralambo, T. (2011). A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*, 49(11), 58-67. doi:10.1109/MCOM.2011.6069710
- Goessner, S. (2007). *JSON Path–XPath for JSON*. Retrieved from <http://goessner.net/articles/JsonPath/> (last accessed April 6 2014).
- Gottschalk, K., Graham, S., Kreger, H., & Snell, J. (2002). Introduction to web services architecture. *IBM Systems Journal*, 41(2), 170-177.
- Grammel, L., & Storey, M.-A. (2008). An end user perspective on mashup makers. *University of Victoria Technical Report DCS-324-IR*.
- Grammel, L., & Storey, M.-A. (2010). A Survey of Mashup Development Environments. In M. Chignell, J. Cordy, J. Ng, & Y. Yesha (Eds.), *The Smart Internet* (Vol. 6400, pp. 137-151): Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-16599-3_10. doi:10.1007/978-3-642-16599-3_10
- Green, T. R. (1989). Cognitive dimensions of notations. *Proceedings of the Fifth Conference of the British Computer Society Human-Computer Interaction Specialist Group (People and computers V)*, (pp. 443-460), Cambridge University Press
- Green, T. R., Petre, M., & Bellamy, R. (1991). Comprehensibility of visual and textual programs: A test of superlativism against the 'match-mismatch' conjecture. *ESP*, 91(743), 121-146.
- Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2), 131-174.
- Greenfield, J., & Short, K. (2003). Software factories: assembling applications with patterns, models, frameworks and tools. *Proceedings of the Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, (pp. 16-27), ACM
- Gretlein, S. (2013). Software Modeling for Embedded Systems. In *Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications* (pp. 59-90): Elsevier.
- Gronback, R. C. (2009). *Eclipse modeling project: a domain-specific language (DSL) toolkit*: Pearson Education.
- Gruber, T. R., & Olsen, G. R. (1994). An Ontology for Engineering Mathematics. *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, (pp. 258-269), Gustav Stresemann Institut, Bonn, Germany. Morgan Kaufmann
- Gu, T., Pung, H. K., & Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, 28(1), 1-18.
- Gu, T., Wang, X. H., Pung, H. K., & Zhang, D. Q. (2004). An ontology-based context model in intelligent environments. *Proceedings of the Communication networks and distributed systems modeling and simulation conference*, (pp. 270-275), Proceedings Central (Thompson Reuter)
- Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645-1660.
- Guinard, D., Fischer, M., & Trifa, V. (2010, March 29 2010-April 2 2010). Sharing using social networks in a composable Web of Things. *Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, (pp. 702-707), IEEE doi:10.1109/PERCOMW.2010.5470524
- Guinard, D., Floerkemeier, C., & Sarma, S. (2011). Cloud computing, REST and mashups to simplify RFID application development and deployment. *Proceedings of the 2nd International Workshop on Web of Things*, (pp. 9), ACM
- Guinard, D., & Trifa, V. (2009). Towards the web of things: Web mashups for embedded devices. *Proceedings of the Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, in *proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, (pp. 15)
- Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., & Savio, D. (2010). Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *Services Computing, IEEE Transactions on*, 3(3), 223-235. doi:10.1109/TSC.2010.3

BIBLIOGRAPHY

- Guinard, D., Trifa, V., Pham, T., & Liechti, O. (2009, 17-19 June). Towards physical mashups in the Web of Things. *Proceedings of the Sixth International Conference on Networked Sensing Systems (INSS)*, (pp. 1-4), Pittsburgh, USA. IEEE. doi:10.1109/INSS.2009.5409925
- Guinard, D., Trifa, V., & Wilde, E. (2010a, Nov. 29 2010-Dec. 1 2010). A resource oriented architecture for the Web of Things. *Proceedings of the Internet of Things (IOT), 2010*, (pp. 1-8), IEEE. doi:10.1109/IOT.2010.5678452
- Guinard, D., Trifa, V. M., & Wilde, E. (2010b). *Architecting a mashable open world wide web of things*: Institute for Pervasive Computing, ETH Zurich.
- Gülcü, C. (2003). *The complete log4j manual*. Montreux, Switzerland: QOS. ch.
- Gutierrez, J. A. (2004, 5-8 Sept. 2004). On the use of IEEE 802.15.4 to enable wireless sensor networks in building automation. *Proceedings of the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications. PIMRC.*, (pp. 1865-1869), IEEE. doi:10.1109/PIMRC.2004.1368322
- Gutierrez, J. A., Callaway, E. H., & Barrett, R. L. (2004). *Low-rate wireless personal area networks: enabling wireless sensors with IEEE 802.15. 4*: IEEE Standards Association.
- Gutierrez, J. A., Naeve, M., Callaway, E., Bourgeois, M., Mitter, V., & Heile, B. (2001). IEEE 802.15. 4: a developing standard for low-power low-cost wireless personal area networks. *network, IEEE, 15*(5), 12-19.
- Haag, A., Goronzy, S., Schaich, P., & Williams, J. (2004). Emotion recognition using bio-sensors: First steps towards an automatic system. In *Affective dialogue systems* (pp. 36-48): Springer.
- Haakenstad, L. K. (1999). The open protocol standard for computerized building systems: BACnet. *Proceedings of the IEEE International Conference on Control Applications*, (pp. 1585-1590), IEEE
- Hachem, S., Teixeira, T., & Issarny, V. (2011). Ontologies for the internet of things. *Proceedings of the 8th Middleware Doctoral Symposium*, (pp. 1-6), Lisbon, Portugal. 2093193, ACM. doi:10.1145/2093190.2093193
- Hadim, S., & Mohamed, N. (2006). Middleware: Middleware challenges and approaches for wireless sensor networks. *Distributed Systems Online, IEEE, 7*(3), 1-1.
- Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal, 45*(3), 451-461.
- Hanshen, G., & Dong, W. (2009). A Content-aware Fridge based on RFID in smart home for home-healthcare. *Proceedings of the 11th International Conference on Advanced Communication Technology (ICACT)* (pp. 987-990), IEEE
- Hay, C. (2014). Scan a med, save a life. *Journal of AHIMA/American Health Information Management Association, 85*(1), 36-39; quiz 40.
- He, D., Lobov, A., & Lastra, J. L. M. (2012). An Application of B2MML in Computer Integrated Manufacturing Systems. *Proceedings of the 16th International Conference on Mechatronics Technology*, (pp. 16-19)
- Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. *Proceedings of the 14th conference on Computational linguistics-Volume 2*, (pp. 539-545), Association for Computational Linguistics
- Henricksen, K., Indulska, J., McFadden, T., & Balasubramaniam, S. (2005). Middleware for distributed context-aware systems. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE* (pp. 846-863): Springer.
- Hodgson, R., & Keller, P. J. (2011). QUDT-quantities, units, dimensions and data types in OWL and XML. *Online (September 2011)* <http://www.qudt.org>.
- Holger, N., & Compton, M. (2009). The Semantic Sensor Network Ontology: A Generic Language to Describe Sensor Assets. *Proceedings of the 12th AGILE International Conference on Geographic Information Science, Workshop on Challenges in Geospatial Data Harmonisation*, Hannover, Germany. Retrieved from <http://www.w3.org/2005/Incubator/ssn/wiki/images/2/2e/SemanticSensorNetworkOntology.pdf>
- Hollingsed, T., & Novick, D. G. (2007). Usability inspection methods after 15 years of research and practice. *Proceedings of the 25th annual ACM international conference on Design of communication*, (pp. 249-255), ACM
- Horstmann, M., & Kirtland, M. (1997). DCOM architecture *Microsoft Corporation*.

BIBLIOGRAPHY

- Hui, J., & Thubert, P. (2010). Compression format for IPv6 datagrams in 6LoWPAN networks. *draft-ietf-6lowpan-hc-13 (work in progress)*.
- Humphreys, T. E., Ledvina, B. M., Psiaki, M. L., O'Hanlon, B. W., & Kintner Jr, P. M. (2008). Assessing the spoofing threat: Development of a portable GPS civilian spoofer. *Proceedings of the ION GNSS international technical meeting of the satellite division*, (pp. 56)
- Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE)*, (pp. 791-798), IEEE
- Hur, S.-H., Kim, D., & Park, G.-T. (2006). Building automation system via LonWorks and Linux based personal computer. *Automation in construction*, 15(4), 522-530.
- IEEE. (2010). IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems–Local and Metropolitan Area Networks–Specific Requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments *IEEE Standard* (Vol. 802, pp. 10-12).
- Institute, P. M. (2013). Project Time Management. In *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)* (pp. 129-164): Project Management Institute, Incorporated.
- International, M. (1997). *MES explained: a high level vision: MESA*
- IoT-A. (2013). *DI.5 – Final architectural reference model for the IoT*. Retrieved from http://www.iot-a.eu/public/public-documents/di.5/at_download/file
- ISO. (1998). ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability.
- ISO. (2009). 9241-210: 2010. Ergonomics of human system interaction-Part 210: Human-centred design for interactive systems. *International Standardization Organization (ISO). Switzerland*.
- ITU-T. (2012). Overview of Internet of Things. *Series Y: Global Information Infrastructure, Internet Protocol Aspects and Next-Generation Networks* (Vol. Y.2060). Geneva: Telecommunication Standardization Sector of ITU.
- Jaeger, R. G., & Halliday, T. R. (1998). On confirmatory versus exploratory research. *Herpetologica*, S64-S66.
- Jahn, M., Jentsch, M., Prause, C. R., Pramudianto, F., Al-Akkad, A., & Reiners, R. (2010, 21-23 May 2010). The Energy Aware Smart Home. *Proceedings of the 5th International Conference on Future Information Technology (FutureTech)*, (pp. 1-8), Busan, Korea. doi:10.1109/FUTURETECH.2010.5482712
- Jammes, F., Mensch, A., & Smit, H. (2005). Service-oriented device communications using the devices profile for web services. *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, (pp. 1-8), ACM
- Jammes, F., & Smit, H. (2005). Service-oriented paradigms in industrial automation. *Industrial Informatics, IEEE Transactions on*, 1(1), 62-70.
- Janowicz, K., & Compton, M. (2010). The Stimulus-Sensor-Observation Ontology Design Pattern and its Integration into the Semantic Sensor Network Ontology. *Proceedings of the 3rd International workshop on Semantic Sensor Networks*, Retrieved from <http://ceur-ws.org/Vol-668/paper12.pdf>
- Jara, A. J., Zamora, M. A., & Skarmeta, A. (2012). Glowbal IP: An adaptive and transparent IPv6 integration in the Internet of Things. *Mobile Information Systems*, 8(3), 177-197.
- Jasperneite, J., & Feld, J. (2005, Sept. 19-22). PROFINET: an integration platform for heterogeneous industrial communication systems. *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, (pp. 822), IEEE. doi:10.1109/ETFA.2005.1612610
- Jentsch, M., Ramirez, L., Wood, L., & Elmasllari, E. (2013). the reconfiguration of triage by introduction of technology. *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*, (pp. 55-64), Munich, Germany. 2493212, ACM. doi:10.1145/2493190.2493212
- Jeronimo, M., & Weast, J. (2003). *UPnP design by example* (Vol. 158): Intel Press.
- Jianhua, S., Jiafu, W., Hehua, Y., & Hui, S. (2011, 9-11 Nov. 2011). A survey of Cyber-Physical Systems. *Proceedings of the International Conference on Wireless Communications and Signal Processing (WCSP)*, (pp. 1-6). doi:10.1109/WCSP.2011.6096958

- Jianping, S., Song, H., Mok, A. K., Deji, C., Lucas, M., & Nixon, M. (2008, 22-24 April 2008). WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE*, (pp. 377-386). doi:10.1109/RTAS.2008.15
- Jin, Y., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding Mashup Development. *Internet Computing, IEEE, 12*(5), 44-52. doi:10.1109/MIC.2008.114
- Jingjing, X., Lee, Y. H., Tsai, W. T., Wu, L., Young-Sung, S., Jun-Hee, P., et al. (2009, 25-27 May 2009). Ontology-Based Smart Home Solution and Service Composition. *Proceedings of the International Conference on Embedded Software and Systems (ICCESS '09)*, (pp. 297-304). doi:10.1109/ICCESS.2009.60
- John, K.-H., & Tiegelkamp, M. (2010). *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*: Springer.
- Johnson, F. A. J., Harrison, M., US, B. H. G., Mitsugi, J., Preishuber, J., CVS, O. R., et al. (2005). The EPCglobal Architecture Framework.
- Jung, M., Reinisch, C., & Kastner, W. (2012). Integrating building automation systems and ipv6 in the internet of things. *Proceedings of the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, (pp. 683-688), IEEE
- Kampers, F., Rossing, W., & Eradus, W. (1999). The ISO standard for radiofrequency identification of animals *Computers and electronics in agriculture* (Vol. 24, pp. 27-43).
- Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., & Terziyan, V. Y. (2008). Smart Semantic Middleware for the Internet of Things. *ICINCO-ICSO, 8*, 169-178.
- Kaufmann, E., Bernstein, A., & Fischer, L. (2007). *NLP-Reduce: A "naive" but Domain-independent Natural Language Interface for Querying Ontologies*: ESWC Zurich. Retrieved from <https://www.merlin.uzh.ch/contributionDocument/download/2304>
- Kazman, R., Bass, L., Webb, M., & Abowd, G. (1994). SAAM: a method for analyzing the properties of software architectures. *Proceedings of the 16th international conference on Software engineering*, (pp. 81-90), Sorrento, Italy. 257746, IEEE Computer Society Press
- Kefalakis, N., Leontiadis, N., Soldatos, J., & Donsez, D. (2009). Middleware building blocks for architecting RFID systems. In *Mobile Lightweight Wireless Systems* (pp. 325-336): Springer.
- Kiliccote, S., Piette, M. A., Koch, E., & Hennage, D. (2011). Utilizing automated demand response in commercial buildings as non-spinning reserve product for ancillary services markets. *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, (pp. 4354-4360), IEEE
- Kim, J.-T., Oh, Y.-J., Lee, H.-K., Paik, E.-H., & Park, K.-R. (2007). Implementation of the DLNA proxy system for sharing home media contents. *Consumer Electronics, IEEE Transactions on*, 53(1), 139-144.
- Kim, J., Tang, K., Kumara, S., Yee, S.-T., & Tew, J. (2008). Value analysis of location-enabled radio-frequency identification information on delivery chain performance. *International Journal of Production Economics, 112*(1), 403-415. doi:<http://dx.doi.org/10.1016/j.ijpe.2007.04.006>
- Kitchenham, B., Pickard, L., & Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE software, 12*(4), 52-62.
- Klusck, M., Fries, B., & Sycara, K. (2006). Automated semantic web service discovery with OWLS-MX. *Proceedings of the Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, (pp. 915-922), ACM
- Kortuem, G., Kawsar, F., Fitton, D., & Sundramoorthy, V. (2010). Smart objects as building blocks for the internet of things. *Internet Computing, IEEE, 14*(1), 44-51.
- Kostelnik, P., Sarnovsky, M., & Hreno, J. (2009). Ontologies in HYDRA-Middleware for Ambient Intelligent Devices. *Proceedings of the Ambient Intelligent Perspectives II*, (pp. 43-46), IOS Press
- Kostelnik, P., Sarnovsky, M., Hreno, J., Ahlsen, M., Rosengren, P., Kool, P., et al. (2008). Semantic devices for ambient environment middleware. *Proceedings of EURO TrustAmi*, 18-19.
- Koubâa, A., & Andersson, B. (2009). A vision of cyber-physical internet. *Proceedings of the Workshop of Real-Time Networks (RTN), Satellite Workshop to (ECRTS)*, (pp. 1)

- Kranz, M., Holleis, P., & Schmidt, A. (2010). Embedded Interaction: Interacting with the Internet of Things. *Internet Computing, IEEE, 14*(2), 46-53. doi:10.1109/MIC.2009.141
- Kumar N.G., C., Vyas, S., Cytron, R. K., Gill, C. D., Zambreno, J., & Jones, P. H. (2013). Scheduling Challenges in Mixed Critical Real-Time Heterogeneous Computing Platforms. *Proceedings of the International Conference on Computational Science*, (pp. 1891-1898), Elsevier, Retrieved from <http://www.sciencedirect.com/science/article/pii/S1877050913005012>. doi:<http://dx.doi.org/10.1016/j.procs.2013.05.358>
- Lampkin, V., Leong, W. T., Olivera, L., Rawat, S., Subrahmanyam, N., & Xiang, R. (2012). *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*: IBM.
- Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes, 25*(2-3), 259-284.
- Leach, P. J., Mealling, M., & Salz, R. (2005). A universally unique identifier (uuid) urn namespace.
- Lee, G. M., Crespi, N., Choi, J. K., & Boussard, M. (2013). Internet of Things. In *Evolution of Telecommunication Services* (pp. 257-282): Springer.
- Lefort, L., Henson, C., Taylor, K., Barnaghi, P., Compton, M., Corcho, O., et al. (2011). Semantic sensor network xg final report. *W3C Incubator Group Report, 28*.
- Lewis, J. R. (1991). Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the ASQ. *ACM SIGCHI Bulletin, 23*(1), 78-81.
- Lewis, J. R. (1992). Psychometric evaluation of the post-study system usability questionnaire: The PSSUQ. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, (pp. 1259-1260), SAGE Publications
- Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction, 7*(1), 57-78.
- Liu, Q., Bai, Q., Zednik, S., Taylor, P., Fox, P., Taylor, K., et al. (2010). A Provenance Model for Real-Time Water Information Systems. *Proceedings of the American Geophysical Union Fall Meeting Abstracts*, (pp. 1361)
- Locke, D. (2010). MQ Telemetry Transport (MQTT) V3. 1 Protocol Specification. *IBM developerWorks Technical Library*, available at <http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html>.
- Lu, G., Krishnamachari, B., & Raghavendra, C. S. (2004). Performance evaluation of the IEEE 802.15. 4 MAC for low-rate low-power wireless networks. *Proceedings of the IEEE International Conference on Performance, Computing, and Communications*, (pp. 701-706), IEEE
- Lu, G., Seed, D., Starsinic, M., Wang, C., & Russell, P. (2012). Enabling smart grid with ETSI M2M standards. *Proceedings of the Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, (pp. 148-153), IEEE
- Lu, W., Chiu, K., & Gannon, D. (2006). Building a generic SOAP framework over binary XML. *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, (pp. 195-204), IEEE
- Macii, E., & Osello, A. (2011). Smart Energy Efficiency Control in Existing Buildings by a BIM Interoperable Process.
- Madsen, T., Martin, Y., Brizzi, P., & Rosengren, P. (2013). *D10.5 Second prototype application for traceability*: ebbits consortium.
- Mattern, F. (2003). From smart devices to smart everyday objects. *Proceedings of the Smart Objects Conference*, Retrieved from http://www.vs.inf.ethz.ch/publ/papers/Generic_106.pdf
- Maximilien, E. M., Ranabahu, A., & Gomadam, K. (2008). An online platform for web apis and service mashups. *Internet Computing, IEEE, 12*(5), 32-43.
- Mellor, S. J. (2004). *MDA distilled: principles of model-driven architecture*: Addison-Wesley Professional.
- Milagro, F., Antolin, P., Fernandes, J., Zhang, W., Hansen, K. M., & Kool, P. (2009). Deploying pervasive web services over a p2p overlay. *Proceedings of the 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, (pp. 240-245), IEEE

BIBLIOGRAPHY

- Miller, B. A., Nixon, T., Tai, C., & Wood, M. D. (2001). Home networking with universal plug and play. *Communications Magazine, IEEE*, 39(12), 104-109.
- Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, 3(4), 235-244.
- Mingozzi, E., Tanganelli, G., Vallati, C., & Di Gregorio, V. (2013, 24-27 June 2013). An open framework for accessing Things as a service. *Proceedings of the 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, (pp. 1-5)
- Modbus, I. (2004a). Modbus application protocol specification v1. 1a. *North Grafton, Massachusetts* (www.modbus.org/specs.php).
- Modbus, I. (2004b). Modbus application protocol specification v1. 1a. *North Grafton, Massachusetts*.
- Modi, V., & Kemp, D. (2009). Web Services Dynamic Discovery (WS-Discovery) Version 1.1. *Organization for the Advancement of Structured Information Standards (OASIS), OASIS Standard*.
- Mokhtar, S. B., Preuveneers, D., Georgantas, N., Issarny, V., & Berbers, Y. (2008). EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *Journal of Systems and Software*, 81(5), 785-808.
- Morrison, J. P. (2010). *Flow-Based Programming: A new approach to application development*: CreateSpace.
- Mulligan, G. (2007). The 6LoWPAN architecture. *Proceedings of the 4th Workshop on Embedded Networked Sensors*, (pp. 78-82), ACM
- Musset, J., Juliot, É., Lacrampe, S., Piers, W., Brun, C., Goubet, L., et al. (2006). Aceleo user guide.
- Navarro-Prieto, R., & Cañas, J. J. (2001). Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies*, 54(6), 799-829.
- Nielsen, J. (2000). *Why You Only Need to Test with 5 Users*. Retrieved from <http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/> (last accessed June 23, 2014).
- Nielsen, J., & Landauer, T. K. (1993). A mathematical model of the finding of usability problems. *Proceedings of the INTERACT and CHI conference on Human factors in computing systems*, (pp. 206-213), ACM
- Ning, H., Liu, H., Wu, J., Du, W., Yang, L., Wang, Z., et al. (2013). Human Attention Inspired Resource Allocation for Heterogeneous Sensors in the Web of Things. *IEEE Intelligent Systems*, 28(6), 20 - 28. doi:10.1109/MIS.2013.103
- Niyato, D., Xiao, L., & Wang, P. (2011). Machine-to-machine communications for home energy management system in smart grid. *Communications Magazine, IEEE*, 49(4), 53-59.
- Northover, S., & Wilson, M. (2004). *Swt: the standard widget toolkit, volume 1*: Addison-Wesley Professional.
- Nottingham, M., & Sayre, R. (2005). The atom syndication format.
- Nunes, D. A., & Schwabe, D. (2006). Rapid prototyping of web applications combining domain specific languages and model driven design. *Proceedings of the 6th international conference on Web engineering*, (pp. 153-160), Palo Alto, California, USA. 1145616, ACM. doi:10.1145/1145581.1145616
- OMG. (2009). *Quantities, Units, Dimensions, Values (QUDV)*. Retrieved from http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-qudv:quantities_units_dimensions_values_qudv (last accessed January 24, 2014).
- OMG, M. (2003). Guide Version 1.0. 1. *Object Management Group*, 62.
- Ong, E. H., Kneckt, J., Alanen, O., Chang, Z., Huovinen, T., & Nihtila, T. (2011). IEEE 802.11 ac: Enhancements for very high throughput WLANs. *Proceedings of the 22nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, (pp. 849-853), IEEE
- OPCFoundation. (2014). *OPC overview*. Retrieved from <https://opcfoundation.org/about/what-is-opc/> (last accessed December 3, 2014).
- Osello, A., Acquaviva, A., Aghemo, C., Blaso, L., Dalmasso, D., Erba, D., et al. (2013). Energy saving in existing buildings by an intelligent use of interoperable ICTs. *Energy Efficiency*, 6(4), 707-723.
- Outay, F., Vèque, V., & Bouallègue, R. (2007). Survey of Service Discovery Protocols and Benefits of Combining Service and Route Discovery. *IJCSNS*, 7(11), 85.

BIBLIOGRAPHY

- Parikh, P. P., Kanabar, M. G., & Sidhu, T. S. (2010). Opportunities and challenges of wireless communication technologies for smart grid applications. *Proceedings of the Power and Energy Society General Meeting, 2010 IEEE*, (pp. 1-7), IEEE
- Pastor, A., Ho, E., Magerkurth, C., Martín, G., Sáinz, I., Segura, A. S., et al. (2011). *IoT-A D6.2 – Updated Requirements List*
- Pawlak, A. M. (2006). *Sensors and actuators in mechatronics: design and applications*: CRC Press.
- Pedersen, T., Patwardhan, S., & Michelizzi, J. (2004). WordNet:: Similarity: measuring the relatedness of concepts. *Proceedings of the Demonstration Papers at HLT-NAACL 2004*, (pp. 38-41), Association for Computational Linguistics
- Pekar, V., & Staab, S. (2002). Taxonomy learning: factoring the structure of a taxonomy into a semantic classification decision. *Proceedings of the 19th international conference on Computational linguistics*, (pp. 1-7), Taipei, Taiwan. 1072318, Association for Computational Linguistics. doi:10.3115/1072228.1072318
- Peña-López, I. (2005). *ITU Internet Report 2005: The Internet of Things*: International Telecommunication Union.
- Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey. *Communications Surveys & Tutorials, IEEE*, 16(1), 414-454. doi:10.1109/SURV.2013.042313.00197
- Perkins, C., Belding-Royer, E., & Das, S. (2003). RFC 3561-ad hoc on-demand distance vector (AODV) routing. *Internet RFCs*, 1-38.
- Pettey, C. (2013). Gartner Says It's the Beginning of a New Era: The Digital Industrial Economy. Orlando, Florida.
- Plugge, E., Membrey, P., & Hawkins, T. (2010). Introduction to MongoDB. In *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing* (pp. 2-4): Springer.
- Plummer, D. C. (1982). RFC 826: An ethernet address resolution protocol. *InterNet Network Working Group*.
- Ponzetto, S. P., & Strube, M. (2007). Deriving a large scale taxonomy from Wikipedia. *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, (pp. 1440-1445), AAAI Press
- Pramudianto, F., Avila, B., Jarke, M., Pullman, J., & Jahn, M. (2014, August 26-28). Extending Semantic Device Discovery with Synonym of Terms. *Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, (pp. 81-88), Milan, Italy. IEEE
- Pramudianto, F., Khaleel, H., Pastrone, C., Simon, J., Eisenhauer, M., & Spirito, M. (2013). Prototyping the Internet of Things Technology for the Future Factory Using Service Oriented Architecture Middleware *Proceedings of the 18th IEEE International Conference on Emerging Technologies & Factory Automation*, (pp. 1-4), IEEE
- Pramudianto, F., Rusmita, I., & Jarke, M. (2013). Model Driven Development for Internet of Things Application Prototyping. *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, (pp. 703-708), Knowledge Systems Institute Graduate School, Retrieved from http://www.ksi.edu/seke/Proceedings/seke/SEKE2013_Proceedings.pdf
- Presser, A., Farrell, L., Kemp, D., Lupton, W., Tsuruyama, S., Albright, S., et al. (2008). Upnp device architecture 1.1 *UPnP Forum*.
- Qin, W., Shi, Y., & Suo, Y. (2007). Ontology-based context-aware middleware for smart spaces. *Tsinghua Science & Technology*, 12(6), 707-713.
- Ramsey, H. R., Atwood, M. E., & Van Doren, J. R. (1983). Flowcharts versus program design languages: an experimental comparison. *Communications of the ACM*, 26(6), 445-449.
- Rathnayaka, A. D., Potdar, V. M., & Kuruppu, S. J. (2011). Evaluation of wireless home automation technologies. *Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies Conference (DEST)*, (pp. 76-81), IEEE
- Recordon, D., & Reed, D. (2006). OpenID 2.0: a platform for user-centric identity management. *Proceedings of the Second ACM Workshop on Digital identity management*, (pp. 11-16), ACM

- Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *Proceedings of the 14th international joint conference on Artificial intelligence. IJCAI.*, (pp. 448-453), San Francisco, CA, USA, Morgan Kaufmann Publishers Inc.
- Roberti, M. (2004). EPCglobal ratifies Gen 2 standard. *RFID Journal*, 16.
- Rose, J., & Reimann, J. (2014). *Eclipse SCADA: The definite guide*: Jens Reimann.
- Rukzio, E., Schmidt, A., & Hussmann, H. (2004, 2-3 Dec. 2004). Physical posters as gateways to context-aware services for mobile devices. *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications. WMCSA.* , (pp. 10-19). doi:10.1109/MCSA.2004.20
- Rusmita, I. (2012). *Prototyping Tool for Mapping Physical Sensors to Domain Model*. RWTH Aachen.
- Sauter, T. (2005, 19-22 Sept. 2005). Integration aspects in automation - a technology survey. *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 9 pp.-263). doi:10.1109/ETFA.2005.1612688
- Sauter, T. (2007). The continuing evolution of integration in manufacturing automation. *Industrial Electronics Magazine, IEEE*, 1(1), 10-19. doi:10.1109/MIE.2007.357183
- Schleipen, M. (2008). OPC UA supporting the automated engineering of production monitoring and control systems. *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, (pp. 640-647), IEEE
- Semiconductors, P. (1996). CANbus specification.
- Serbanati, A., Medaglia, C. M., & Ceipidor, U. B. (2011). Building blocks of the internet of things: State of the art and beyond. *Deploying RFID-Challenges, Solutions, and Open Issues*, C. Turcu, Ed., ed: InTech.
- Serral, E., Valderas, P., & Pelechano, V. (2008). A model driven development method for developing context-aware pervasive systems. In *Ubiquitous Intelligence and Computing* (pp. 662-676): Springer.
- Shelby, Z., Hartke, K., & Bormann, C. (2013). Constrained application protocol (coap).
- Shelby, Z., Hartke, K., Bormann, C., & Frank, B. (2012). *Constrained Application Protocol (CoAP), draft-ietf-core-coap-13*: IETF.
- Shih, D.-H., Sun, P.-L., & Lin, B. (2005). Securing industry-wide EPCglobal network with WS-security. *Industrial Management & Data Systems*, 105(7), 972-996.
- Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, 20(6), 373-381.
- Silva, P. G. D., Karnouskos, S., & Ilic, D. (2012). A survey towards understanding residential prosumers in smart grid neighbourhoods. *Proceedings of the 3rd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe)*, (pp. 1-8), IEEE
- Solution, T. (2011). *Comparison of Passive RFID technologies*. Retrieved from http://handheld-rfid.com/RFID_Comparison.html (last accessed February 19, 2014).
- Song, J., Han, S., Mok, A. K., Chen, D., Lucas, M., & Nixon, M. (2008). WirelessHART: Applying wireless technology in real-time industrial process control. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, (pp. 377-386), IEEE
- Spencer, R. (2000). The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. *Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 353-359), ACM
- Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L., et al. (2009). SOA-based integration of the internet of things in enterprise services. *Proceedings of the IEEE International Conference on Web Services (ICWS)*. (pp. 968-975), IEEE
- Stanford-Clark, A., & Truong, H. L. (2013). *MQTT For Sensor Networks (MQTT-SN) - Protocol Specification*: International Business Machines Corporation (IBM).
- Stasch, C., Janowicz, K., Bröring, A., Reis, I., & Kuhn, W. (2009). A stimulus-centric algebraic approach to sensors and observations. In *GeoSensor Networks* (pp. 169-179): Springer.
- Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *EMF: eclipse modeling framework*: Pearson Education.
- Stirbu, V. (2008). Towards a restful plug and play experience in the web of things. *Proceedings of the IEEE international conference on Semantic computing*, (pp. 512-517), IEEE

- Streitz, N. A., Rocker, C., Prante, T., van Alphen, D., Stenzel, R., & Magerkurth, C. (2005). Designing smart artifacts for smart environments. *Computer*, 38(3), 41-49.
- Sullivan, F. (2013 9 September, 2013). *Over-the-Air Updates to Slash Automobiles' Recall Rates, Finds Frost & Sullivan*. Retrieved from <http://www.frost.com/prod/servlet/press-release.pag?docid=284456381> (last accessed April 28, 2014).
- Swetina, J., Lu, G., Jacobs, P., Ennesser, F., & Song, J. (2014). Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wireless Communications*, 21(3), 20-26.
- Takalo-Mattila, J., Kiljander, J., Pramudianto, F., & Ferrera, E. (2014). Architecture for mixed criticality resource management in Internet of Things. *Proceedings of the 2014 TRON Symposium* Tokyo, Japan. IEEE (to appear)
- Taylor, K., Lamb, D., Griffith, C., Falzon, G., Lefort, L., Gaire, R., et al. (2013). Farming the web of things. *IEEE Intelligent Systems*, 28(6), 12 - 19. doi:10.1109/MIS.2013.102
- Thomesse, J. P. (2005). Fieldbus Technology in Industrial Automation. *Proceedings of the IEEE*, 93(6), 1073-1101. doi:10.1109/JPROC.2005.849724
- Togias, K., Goumopoulos, C., & Kameas, A. (2010, 13-19 June 2010). Ontology-Based Representation of UPnP Devices and Services for Dynamic Context-Aware Ubiquitous Computing Applications. *Proceedings of the Third International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ)*, (pp. 220-225). doi:10.1109/CTRQ.2010.44
- Toninelli, A., Corradi, A., & Montanari, R. (2005). Semantic Discovery for Context-Aware Service Provisioning in Mobile Environments. *Proceedings of the 1st International Workshop on Managing Context Information in Mobile and Pervasive Environments.*, CEUR WS
- Tovar, E., & Vasques, F. (1999). Real-time fieldbus communications using Profibus networks. *IEEE Transactions on Industrial Electronics*, 46(6), 1241-1251.
- Tsai, D., Jing, Y., Liu, Y., Rowley, H. A., Ioffe, S., & Rehg, J. M. (2011). Large-scale image annotation using visual synset. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, (pp. 611-618), IEEE
- Tyhach, J., Wang, B., Sung, C., Huang, J., Nguyen, K., Wang, X., et al. (2005). A 90-nm FPGA I/O buffer design with 1.6-Gb/s data rate for source-synchronous system and 300-MHz clock rate for external memory interface. *IEEE Journal of Solid-State Circuits*, 40(9), 1829-1838.
- Uckelmann, D., Harrison, M., & Michahelles, F. (2011). *Architecting the internet of things*: Springer.
- Udsen, H., CheccoZZo, R., Madsen, T. N., Thestrup, J., Glova, J., & Vajda, V. (2010). *D2.1 Scenarios for Usage of the ebbits Platform*: ebbits consortium. Retrieved from http://www.ebbits-project.eu/downloads/deliverables/D2.1_Scenarios_for_Usage_of_the_ebbits_Platform.pdf
- Vermesan, O., Friess, P., Guillemin, P., Sundmaeker, H., Eisenhauer, M., Moessner, K., et al. (2013). Internet of Things Strategic Research and Innovation Agenda. In O. Vermesan & P. Friess (Eds.), *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. Aalborg, Denmark: River Publishers.
- Vermesan, O., Harrison, M., Vogt, H., Kalaboukas, K., Tomasella, M., Wouters, K., et al. (2010). Internet of Things Vision. In H. Sundmaeker, P. Guillemin, P. Friess, & S. Woelfflé (Eds.), *Vision and Challenges for Realising the Internet of Things* (pp. 43). Luxembourg: Publications Office of the European Union. doi:10.2759/26127
- Vernadat, F. B. (2003). *Enterprise modelling and integration*: Springer.
- Vinoski, S. (2006). Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), 87-89.
- Vullers, R. J., Schaijk, R., Visser, H. J., Penders, J., & Hoof, C. V. (2010). Energy harvesting for autonomous wireless sensor networks. *Solid-State Circuits Magazine, IEEE*, 2(2), 29-38.
- W3C. (2011). *Report work on the ssn ontology*: W3C.
- Wang, C., Xiong, M., Zhou, Q., & Yu, Y. (2007). Panto: A portable natural language interface to ontologies. In *The Semantic Web: Research and Applications* (pp. 473-487): Springer.
- Wang, N., Zhang, N., & Wang, M. (2006). Wireless sensors in agriculture and food industry—Recent development and future perspective. *Computers and electronics in agriculture*, 50(1), 1-14.

BIBLIOGRAPHY

- Wang, R.-C., Chang, Y.-C., & Chang, R.-S. (2009). A semantic service discovery approach for ubiquitous computing. *Journal of Intelligent Manufacturing*, 20(3), 327-335. doi:10.1007/s10845-008-0213-2
- Wasserman, E. (2009). *Riding herd: RFID tracks livestock*. Retrieved from <https://www.rfidjournal.com/purchase-access?type=Article&id=5272&r=%2Farticles%2Fview%3F5272> (last accessed December 2, 2014).
- Wei, W. W.-S. (1994). *Time series analysis*: Addison-Wesley publ.
- Weiser, M. (1999). The computer for the 21st century. *Scientific american*, 265(3), 94-104. doi:10.1038/scientificamerican0991-94
- Wenpeng, L., Sharp, D., & LaRoy, S. (2013, 21-25 July 2013). Data traffic analysis of utility smart metering network. *Proceedings of the IEEE Power and Energy Society General Meeting (PES)*, (pp. 1-4), IEEE. doi:10.1109/PESMG.2013.6672750
- White, S. A. (2004). *Introduction to BPMN: IBM Cooperation*.
- Widdows, D. (2003). Unsupervised methods for developing taxonomies by combining syntactic and statistical information. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, (pp. 197-204), Association for Computational Linguistics
- Williams, A. (2003). *Requirements for automatic configuration of ip hosts*. Retrieved from <https://tools.ietf.org/html/draft-ietf-zeroconf-reqts-12>
- Willis, M. (1999). Proportional-Integral-Derivative Control. *Dept. of Chemical and Process Engineering University of Newcastle*.
- Woolrych, A., & Cockton, G. (2001). Why and when five test users aren't enough. *Proceedings of the IHM-HCI Conference*, (pp. 105-108), Cépadéus Toulouse, France
- WordSense.eu. (2014). *troponym*. Retrieved from <http://www.wordsense.eu/troponym/> (last accessed June 16, 2014).
- Yang, D., & Powers, D. M. (2005). Measuring semantic similarity in the taxonomy of WordNet. *Proceedings of the Twenty-eighth Australasian conference on Computer Science*, (pp. 315-322), Australian Computer Society, Inc.
- Yue, K.-B. (2010). Experience on mashup development with end user programming environment. *Journal of Information Systems Education*, 21(1), 111.
- Zeeb, E., Bobek, A., Bohn, H., & Golatowski, F. (2007). Service-oriented architectures for embedded systems using devices profile for web services. *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. , (pp. 956-963), IEEE
- Zheng, L., & Nakagawa, H. (2002). OPC (OLE for process control) specification and its developments. *Proceedings of the 41st Society of Instrument and Control Engineers (SICE) Annual Conference*, (pp. 917-920), IEEE
- Zimmermann, A. (2007). *Context Management and Personalisation*. RWTH Aachen University, Aachen, Germany.
- Zimmermann, H. (1980). OSI reference model--The ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4), 425-432.
- Zorzi, M., Gluhak, A., Lange, S., & Bassi, A. (2010). From today's INTRAnet of things to a future INTERNet of things: a wireless- and mobility-related view. *Wireless Communications, IEEE*, 17(6), 44-51. doi:10.1109/MWC.2010.5675777

Appendix 1.

Own Publications

Publications related to this dissertation

- Conzon, D., Brizzi, P., Kasinathan, P., Pastrone, C., Pramudianto, F., & Cultrona, P. (2015). Industrial application development exploiting IoT Vision and Model Driven programming *Proceedings of the 18th International Innovations in Services, Network and Clouds*, Paris, France. IEEE (to appear)
- Takalo-Mattila, J., Kiljander, J., Pramudianto, F., & Ferrera, E. (2014). Architecture for mixed criticality resource management in Internet of Things. *Proceedings of the 2014 TRON Symposium* Tokyo, Japan. IEEE (to appear)
- Pramudianto, F., Kamienski, C. A., Souto, E., Borelli, F., Gomes, L. L., Sadok, D., et al. (2014, December 9-12, 2014). IoTLink: An Internet of Things Prototyping Toolkit *Proceedings of the 11th Ubiquitous Intelligence and Computing*, IEEE (to appear)
- Pramudianto, F., Avila, B., Jarke, M., Pullman, J., & Jahn, M. (2014, August 26-28). Extending Semantic Device Discovery with Synonym of Terms. *Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, (pp. 81-88), Milan, Italy. IEEE
- Patti, E., Acquaviva, A., Jahn, M., Pramudianto, F., Tomasi, R., Rabourdin, D., et al. (2014). Event-Driven User-Centric Middleware for Energy-Efficient Buildings and Public Spaces. *Systems Journal, IEEE(99)*, 1-10. doi:10.1109/JSYST.2014.2302750
- Pramudianto, F., Rusmita, I., & Jarke, M. (2013). Model Driven Development for Internet of Things Application Prototyping. *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, (pp. 703-708), Knowledge Systems Institute Graduate School, Retrieved from http://www.ksi.edu/seke/Proceedings/seke/SEKE2013_Proceedings.pdf
- Pramudianto, F., Khaleel, H., Pastrone, C., Simon, J., Eisenhauer, M., & Spirito, M. (2013). Prototyping the Internet of Things Technology for the Future Factory Using Service Oriented Architecture Middleware *Proceedings of the 18th IEEE International Conference on Emerging Technologies & Factory Automation*, (pp. 1-4), IEEE
- Brizzi, P., Conzon, D., Pramudianto, F., Paralic, M., Jacobsen, M., Pastrone, C., et al. (2013, June, 23 – 27). The ebbits platform: leveraging on the Internet of Things to support meat traceability. *Proceedings of the EFITA 2013*, Torino, Italy. CIGR (to appear)
- Brizzi, P., Conzon, D., Khaleel, H., Cultrona, P., Pramudianto, F., Tomasi, R., et al. (2013, September). Bringing the Internet of Things along the Manufacturing Line: A Case Study in Controlling Industrial Robot and Monitoring Energy Consumption Remotely. *Proceedings of the 18th IEEE International Conference on Emerging Technologies & Factory Automation*, (pp. 1-8), Cagliari, Italy.
- Brizzi, P., Conzon, D., Jacobsen, M., Pramudianto, F., & Tomasi, R. (2013). D5.6.3 Context awareness prototypes 3: ebbits consortium.

Pramudianto, F., Zimmermann, A., Prause, C., Eisenhauer, M., & Al-Akkad, A. (2010). *D5.2.1 Architecture for intelligence integration 1: ebbits consortium*. Retrieved from http://www.ebbits-project.eu/downloads/deliverables/D5.2.1_Architecture_for_intelligence_integration.pdf

Jahn, M., Jentsch, M., Prause, C. R., Pramudianto, F., Al-Akkad, A., & Reiners, R. (2010, 21-23 May 2010). The Energy Aware Smart Home. *Proceedings of the 5th International Conference on Future Information Technology (FutureTech)*, (pp. 1-8), Busan, Korea. doi:10.1109/FUTURETECH.2010.5482712

Other Publications

Eisenhauer, M., Jahn, M., Pramudianto, F., Sabol, T., & Hreno, J. (2011). Towards a generic Middleware for developing Ambient Intelligence Applications. *Proceedings of the 2nd Workshop on eeBuildings Data Models at CIB W078–W102*, (pp. 26-28), Sophia Antipolis, France, European Communities

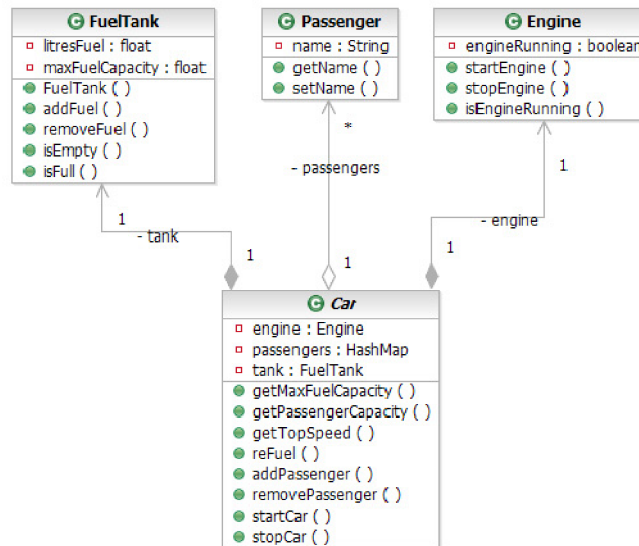
Pramudianto, F., Zimmermann, A., & Rukzio, E. (2009). Magnification for Distance Pointing. *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, (pp. 1-4), Bonn, Germany. ACM

Lorenz, A., Pramudianto, F., & Zimmermann, A. (2009). Lessons learned from an interaction-kiosk for open selection of input devices of a gaming application. *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, (pp. 62), Bonn, Germany. ACM. doi:10.1145/1613858.1613934

Appendix 2.

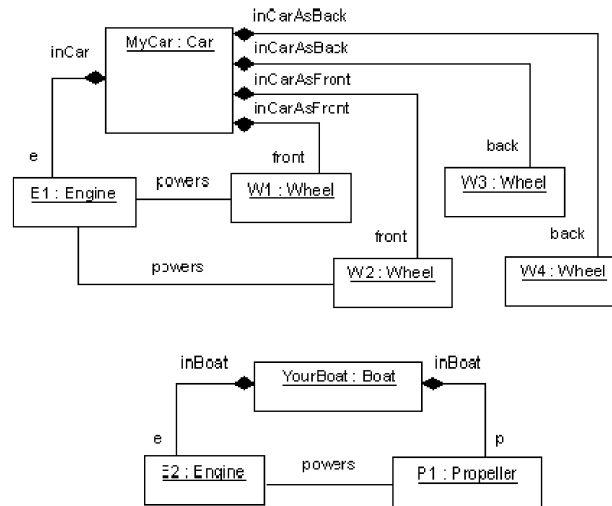
Comprehension Study Tasks

UML Questions



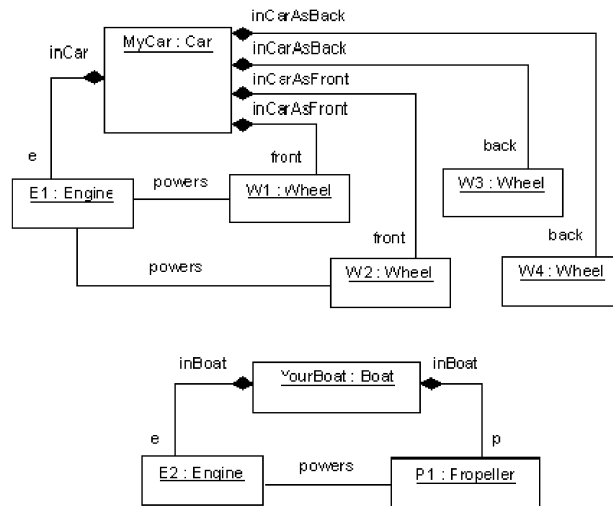
1. Choose the correct meaning of the diagram/code snippet above:
 - A Car may have one or more passengers.
 - A Car may have one or more engines
 - Each car has a fuel tank
 - Each passenger may have one or more cars
 - Each car contains an engine

COMPREHENSION STUDY TASKS



2. Fill in the number of objects you can find in the diagram/code snippet above:

- How many wheels are there?
- How many engines are there?
- How many cars are there?



3. Choose 1 or more correct statements for the diagram/code snippet above:

- The number of engines, front wheels, and back wheels are currently unknown.
- The engine will power the front wheels in the same car as the engine.
- The engine will power the back wheels in the same car as the engine.
- The engine will power the all wheels in the same car as the engine.
- w1 and w2 are the rear (back) wheels.

Java Code Questions

```
class Room
{
    List<Window> windows;

    public List<Window> getWindow()...
    public void setWindows(List<Window> _windows)...

    List<Door> doors;

    public List<Door> getDoors()...
    public void setWindows(List<Door> _doors)...

    List<Light> lights;

    public List<Light> getLights()...
    public void setLights(List<Light> _lights)...
}

class Window...

class Door...

class Light...

class SmartPlug
{
    public void switches(Light light)...
    public void measures(Light light)...
}
```

1. Choose the correct meaning of the diagram/code snippet above:

- A Room may have zero or more windows.
- A Room may have zero or more lights.
- Each smart-plug could switch a light.
- Each light could measure a plug.
- A Room may have arbitrary numbers of doors.

```
1
public static void Main(string[] args)
{
    Room myOffice = new Room();

    Window leftWindow = new Window();
    myOffice.getWindow().Add(leftWindow);

    Window backWindow = new Window();
    myOffice.getWindow().Add(backWindow);

    Light spotLight = new Light();
    myOffice.getLights().Add(spotLight);

    SmartPlug plugX0000 = new SmartPlug();
    plugX0000.switches(spotLight);
    plugX0000.measures(spotLight);

    Light floorLamp = new Light();
    myOffice.getLights().Add(floorLamp);

    SmartPlug plugX0001 = new SmartPlug();
    plugX0001.switches(floorLamp);
    plugX0001.measures(floorLamp);

    Light ceilingLamp = new Light();
    myOffice.getLights().Add(ceilingLamp);

    Light emergencyLight = new Light();
    myOffice.getLights().Add(emergencyLight);
}
}
```

2. Fill in the number of objects you can find in the diagram/code snippet above:

- How many rooms are there?
- How many smart plugs are there?
- How many lights are there?

COMPREHENSION STUDY TASKS

```
1
public static void Main(string[] args)
{
    Room myOffice = new Room();

    Window leftWindow = new Window();
    myOffice.getWindows().Add(leftWindow);

    Window backWindow = new Window();
    myOffice.getWindows().Add(backWindow);

    Light spotLight = new Light();
    myOffice.getLights().Add(spotLight);

    SmartPlug plugX0000 = new SmartPlug();
    plugX0000.switches(spotLight);
    plugX0000.measures(spotLight);

    Light floorLamp = new Light();
    myOffice.getLights().Add(floorLamp);

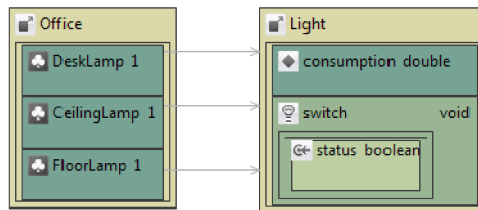
    SmartPlug plugX0001 = new SmartPlug();
    plugX0001.switches(floorLamp);
    plugX0001.measures(floorLamp);

    Light ceilingLamp = new Light();
    myOffice.getLights().Add(ceilingLamp);

    Light emergencyLight = new Light();
    myOffice.getLights().Add(emergencyLight);
}
1
```

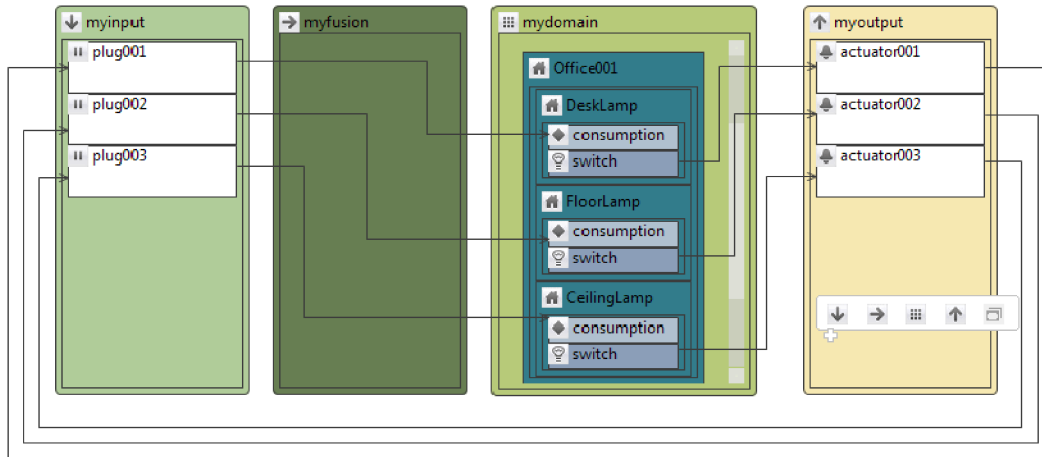
3. Choose 1 or more correct statements for the diagram/code snippet above:
- The number of rooms, smartplugs, and lights are currently unknown.
 - The smartplugs plugX0000 is connected to the ceilinglamp.
 - The smartplugs plugX0001 is connected to the floorlamp.
 - The smartplugs plugX0000 is connected to the spotlight.
 - The smartplugs plugX0001 is connected to the spotlight.

IoTLink Questions



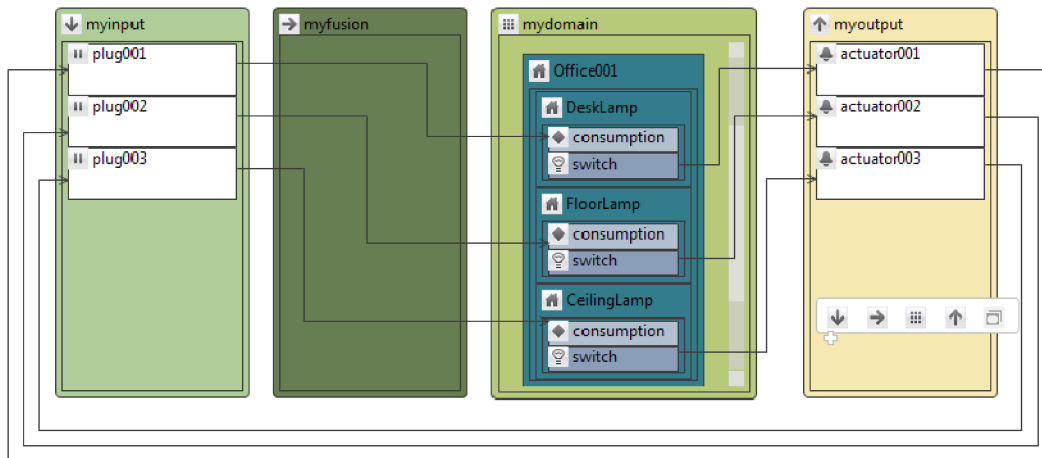
1. Choose the correct meaning of the diagram/code snippet above:
- Each office has a Floor lamp.
 - CeilingLamp is a Light.
 - DeskLamp and FloorLamp have the same type.
 - Each office has three different lamps.
 - The number of lamps in the office are currently unknown.

COMPREHENSION STUDY TASKS



2. Fill in the number of objects you can find in the diagram/code snippet above:

- How many offices are there?
- How many plugs are there?
- How many lamps are there?



3. Choose 1 or more correct statements for the diagram/code snippet above:

- The number of offices, plugs, and lamps are currently unknown.
- The plug001 is connected to the ceilinglamp.
- The plug003 is connected to the ceilinglamp.
- The plug002 is connected to the floorlamp.
- The actuator002 is connected to the plug002.

Appendix 3.

Programming Tasks

Task 1:

You want to create an App that monitor the temperature and light intensity of two rooms, C5-123 and C4-124. Therefore, in the task you want to define a class and objects representing these two rooms.

Task 2:

The temperature and light intensity of the rooms are published through an MQTT Event Broker. Therefore in this task you want to subscribe to four sensor events with the following topics:

- Temperature of Room C5-123 : “FHG/IZB/C5/123/Temperature”
- Temperature of Room C5-124 : “FHG/IZB/C5/124/Temperature”
- Light intensity of Room C5-123 : “FHG/IZB/C5/123/Brightness”
- Light intensity of Room C5-124 : “FHG/IZB/C5/124/Brightness”

That are published to an MQTT server at “tcp://localhost:1883”.

Use these input connections to update the relevant properties of the room objects defined in the previous tasks.

Task 3:

Since the brightness sensors are too sensitive to ambient light changes, you want to perform `\b` an average filter to the two brightness events every five values.

Task 4:

Since latter on you need to access the virtual objects through a mobile application, you want to expose the virtual objects using a RESTful service at `HTTP://localhost:9123/rest`.

Task 5:

Latter on you want to push the states of the objects if they have changed therefore you want to publish them through an MQTT broker at `tcp://localhost:1883` with a base topic of `'UFPE/Room/'`.

Appendix 4.

Post-Study Questionnaire.

IBM Computer System Usability Questionnaire based on: Lewis, J. R. (1995) IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use.

1. Overall, I am satisfied with how easy it is to use this system

1 2 3 4 5 6 7
strongly agree strongly disagree

2. It was simple to use this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

3. I could effectively complete the tasks and scenarios using this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

4. I was able to complete the tasks and scenarios quickly using this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

5. I was able to efficiently complete the tasks and scenarios using this system

1 2 3 4 5 6 7
strongly agree strongly disagree

6. I felt comfortable using this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

7. It was easy to learn to use this system

1 2 3 4 5 6 7
strongly agree strongly disagree

POST STUDY QUESTIONNAIRE

8. I believe I could become productive quickly using this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

9. The system gave error messages that clearly told me how to fix problems.

1 2 3 4 5 6 7
strongly agree strongly disagree

10. Whenever I made a mistake using the system, I could recover easily and quickly.

1 2 3 4 5 6 7
strongly agree strongly disagree

11. The information (such as online help, on-screen messages and other documentation) was effective in helping me completing the tasks.

1 2 3 4 5 6 7
strongly agree strongly disagree

12. It was easy to find the information I needed

1 2 3 4 5 6 7
strongly agree strongly disagree

13. The information provided for the system was easy to understand.

1 2 3 4 5 6 7
strongly agree strongly disagree

14. The information was effective in helping me complete the tasks and scenarios

1 2 3 4 5 6 7
strongly agree strongly disagree

15. The organization of information on the system screens was clear

1 2 3 4 5 6 7
strongly agree strongly disagree

16. The interface of this system was pleasant.

1 2 3 4 5 6 7
strongly agree strongly disagree

17. I liked using the interface of this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

POST STUDY QUESTIONNAIRE

18. This system has all the functions and capabilities I expect it to have.

1 2 3 4 5 6 7
strongly agree strongly disagree

19. Overall, I am satisfied with this system.

1 2 3 4 5 6 7
strongly agree strongly disagree

List the most positive aspect(s):

List the most negative aspect(s):

Other comment(s):

Appendix 5.

After-Scenario Questionnaire.

IBM Computer System Usability Questionnaire based on: Lewis, J. R. (1995) IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use.

1. Overall, I am satisfied with the ease of completing this task.

	1	2	3	4	5	6	7	
strongly agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	strongly disagree

2. Overall, I am satisfied with the amount of time it took to complete this task.

	1	2	3	4	5	6	7	
strongly agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	strongly disagree

3. Overall, I am satisfied with the support information (online help, messages, documentation) when completing this task.

	1	2	3	4	5	6	7	
strongly agree	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	strongly disagree

Appendix 6.

Curriculum Vitae

Name	Ferry Pramudianto M.Sc. PMP
Address	Rheinbacher Str 51, Bonn 53115, Germany
Nationality	Indonesian
Place & date of birth	Jakarta, February 10, 1982
Email	ferryxo@yahoo.com
Phone	+49 176 914 10493
EDUCATION	
2006 – 2009	Master of Science in Media Informatics GPA 1.9/1 (equivalent to grade “A” in the U.S. grading system)
2000 – 2004	Universitas Bina Nusantara (Binus), Jakarta GPA 3.19/4 (equivalent to grade “B” in the U.S. grading system)
PROFESSIONAL EXPERIENCE	
2009 – Present	Research Associate in Fraunhofer FIT, Sankt Augustin – Germany <ul style="list-style-type: none"> • Managing two EU-Brazil research projects, worth more than 1.25M€ and a work package with 137 person months. • Involve in the design and development of an open source middleware using .NET and Java SOAP and RESTFUL Web Services. • Applied human-centered design for interactive systems (ISO 9241-210, 2010) in the projects, Performed User centered design and usability evaluation such as software walkthrough, cognitive dimension. • Involved in system design and development of several distributed-system prototypes and cloud services for the European research projects in building and factory automation. • Conducting a PhD research on model driven development for the Internet of Things mashup applications using eclipse M2T and M2M (MQTT, Web Services, CoAP), Database (Derby, MySQL) • Supervised three master thesis projects in the IoT area.
Mar 2007 – Mar 2009	Research Assistant in Fraunhofer FIT, Sankt Augustin – Germany <ul style="list-style-type: none"> • Performed research on interaction techniques using gesture

CURRICULUM VITAE

	<p>recognition based on Hidden Markov Model statistical analysis.</p> <ul style="list-style-type: none">• Development of home automation prototype using OSGi and Web Services.
Apr 2005 – Jan 2006	<p>Co-founder of Weefer, Batam- Indonesia</p> <ul style="list-style-type: none">• Handled RFP & RFI, Performed requirement engineering, analyzing customer needs and translate them into requirements.• Involved in the design and development of a finance software.
Skills	Jira, Confluence, MS. Project, C#, Windows Form, ASPX, WPF, Silverlight, Java, OSGi, Eclipse, Hibernate, SQL, XML, RDF, PHP, JavaScript, Silverlight, SQL Server, MySQL, Derby, MongoDB
Certification	Project Management Professional (PMP) Certified Professional for Requirements Engineering (CPRE)