HAL open science

AN ENTERPRISE ARCHITECTURE AND MODEL DRIVEN ENGINEERING BASED APPROACH FOR SENSOR NETWORKS

Charbel Aoun

To cite this version:

Charbel Aoun. AN ENTERPRISE ARCHITECTURE AND MODEL DRIVEN ENGINEERING BASED APPROACH FOR SENSOR NETWORKS. Computer Science [cs]. ENSTA Bretagne, 2018. English. ⟨NNT : ⟩. ⟨tel-03250233⟩

HAL Id: tel-03250233

https://theses.hal.science/tel-03250233v1

Submitted on 4 Jun 2021

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

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

Autorisation HAL

HAL Authorization

THESE DE DOCTORAT DE

L'ECOLE NATIONALE SUPERIEURE
DE TECHNIQUES AVANCEES BRETAGNE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies
de l'Information et de la Communication
Spécialité : *Télécommunications, Informatique,*

Par
Charbel AOUN

An Enterprise Architecture and Model Driven Engineering Based Approach for Sensor Networks

Thèse présentée et soutenue à Brest, le 29 Janvier 2018
Unité de recherche : Lab-STICC UMR CNRS 6285 Pôle : Mocs

Rapporteurs avant soutenance :

M. Luc FABRESSE
Prof, Ecole des Mines de Douai, France

Mme Marie-Pierre GERVAIS
Prof, Université Paris Nanterre, France

M. Hervé Leblanc
MdC, Institut de Recherche en Informatique de Toulouse, France

Composition du Jury :

M. Reinhardt EULER
Prof, Université de Bretagne Occidentale, France

M. Luc FABRESSE
Prof, Ecole des Mines de Douai, France

Mme Marie-Pierre GERVAIS
Prof, Université Paris Nanterre, France

M. Loic LAGADEC
Prof, ENSTA Bretagne, France

M. Joel CHAMPEAU
MdC, ENSTA Bretagne, France

Contents

Contents ii

Abstract vii

Acknowledgements ix

List of Figures xii

Introduction xiii
General Context . . . xiii
Sensor Networks . . . xiii
Complexity and Challenges of Sensor Networks . . . xv
Research Questions . . . xvi
Proposed Approach . . . xvii
Organization . . . xvii

I State of the Art 1

1 Sensor Networks Development Process 3
1.1 Sensor Networks . . . 3
1.1.1 Sensor Networks Life Cycles . . . 4
1.1.2 Design Phase of the Sensor Networks Life Cycles . . . 5
1.1.3 Roles in the Sensor Networks Life Cycles . . . 6
1.2 Sensor Networks Systems . . . 7
1.3 Fusion Algorithms . . . 9
1.4 Properties for Selecting a Data Fusion Architecture . . . 9
1.5 Data Fusion Architectures . . . 10
1.6 Limits and Comparison among the Different Data Fusion Architectures . . . 10
1.7 Requirements for Designing Sensor Networks Systems . . . 13
1.8 Limits and Comparison among the Different Approaches of Sensor Networks Design . . . 14
1.8.1 Approaches of Architectural Design Improvement . . . 15
1.8.2 Approaches of Providing Multiple Viewpoints . . . 16
1.8.3 Approaches of Offering Concepts Extensibility . . . 16
1.8.4 Approaches of Supporting Heterogeneity . . . 17
1.8.5 Approaches of Supporting Validation Tools . . . 17
1.9 Discussion . . . 18

2 Model Driven Engineering **21**
2.1 Model Driven Engineering Fundamentals . . . 21
2.2 Model Driven Engineering Aspects . . . 22
2.2.1 Modeling Languages . . . 23
2.2.2 Model Heterogeneity and Quality . . . 24
2.2.3 Models Transformation . . . 25
2.3 Separation of Concerns in Model Driven Engineering . . . 26
2.4 Model Driven Engineering Standards and Tools . . . 27
2.5 Model Driven Engineering for Sensor Networks . . . 28
2.6 Discussion . . . 29

3 System Architecture Modeling **31**
3.1 Modeling Context . . . 31
3.2 Enterprise Architecture Types . . . 32
3.3 Enterprise Architecture Frameworks . . . 32
3.4 Domain Specific Concepts in Enterprise Architecture Frameworks . . . 34
3.5 Enterprise Architecture Modeling Languages and MetaModels . . . 35
3.5.1 ArchiMate . . . 35
3.5.2 TOGAF 9 . . . 38
3.6 Requirements for Selecting the Enterprise Architecture MetaModel . . . 39
3.7 Comparison Among Enterprise Architecture MetaModels . . . 39
3.8 Enterprise Architecture Frameworks and Design Tools for Sensor Networks 40
3.9 DeVerTeS: A Design and Verification Framework for Telecommunication Services . . . 42
3.10 Discussion . . . 43

II Contributions **45**

4 Sensor Networks Design Process **49**
4.1 Context . . . 49
4.2 Software Development Processes . . . 50
4.3 Selected and Proposed tasks of the Sensor Networks Design Process . . . 51
4.3.1 Concept and Challenges of Sensor Networks Design Phase . . . 51
4.3.2 Requirements for Selecting or Proposing Tasks of the Sensor Networks Design Process . . . 52
4.3.3 Analyzing the Relation between the Tasks of the Software Development Processes and the Identified Requirements . . . 52
4.3.4 Proposed Tasks of the Sensor Networks Design Process . . . 53
4.4 The Proposed Sensor Networks Design Process and Model Driven Engineering 55
4.5 Content of the Proposed Tasks of the Sensor Networks Design Process . . . 55
4.5.1 Modeling . . . 57
4.5.2 Ensuring Consistency . . . 58
4.5.3 Validating . . . 59
4.6 Discussion . . . 59

5 Domain Specific Modeling Languages and Design Tools for Sensor Networks Design **61**
5.1 ArchiMO Definition . . . 61
5.1.1 Marine Observatory Context . . . 61
5.1.2 Selected ArchiMate Concepts and Relationships . . . 62
5.1.3 ArchiMO MetaModel . . . 64
5.1.4 ArchiMO MetaModel Layer Consistency . . . 71
5.1.5 Formalization of Layers Interoperability . . . 74
5.1.6 ArchiMO Design Tool . . . 75
5.2 Generation of Simulation Code . . . 80
5.3 ArchiMO and Iterative Approach . . . 82
5.4 Discussion . . . 84

6 Application of the Proposed Sensor Networks Design Process to a Case Study **89**
6.1 Underwater Object Localization Case Study . . . 89
6.2 Modeling a Marine Observatory Case Study using ArchiMO DSML and Design Tool . . . 91
6.2.1 The Business Model Design . . . 91
6.2.2 The Application Model Design . . . 92
6.2.3 The Technology Model Design . . . 93
6.3 Consistency between Model Layers . . . 93
6.4 Simulation Code . . . 95
6.5 Validation of Marine Observatory Model . . . 96
6.6 Iteration of the Proposed Sensor Networks Design Process . . . 97
6.7 Discussion . . . 99

Conclusion and Perspectives **101**
Answering the Research Questions . . . 101
Perspectives . . . 103
Bibliography . . . 103

Abstract

Marine observatories (MO) based on sensor networks provide a continuous ocean monitoring. These sensor networks contain several kinds of sensors including acoustic hydrophones to detect and localize moving objects or animals like dolphins. In the context of marine observatories, the sensor networks provide high level services and are included in an information system to process, store and present the sensor data. This kind of system is considered as complex system and is assimilated as enterprise system with business rules and services and with several hypothesis to map these services to the distributed enterprise infrastructure.

To specify, develop and deploy such systems remains a challenge to satisfy the needs, and the associated requirements, with the respect of the platform constraints. So, one of the questions is how to improve life-cycle of these systems to contribute the architecture design which is one of the sensible phase. Because this phase is the crucial one to obtain the best trade-of between the services and the infrastructure.

So in this work, we try to contribute a system life-cycle based on the use of a model driven approach with an early validation phase to support ease up the development and deployment phases. The use of the models provide the facility to apply an iterative approach at system level which remains a challenge compare to the software processes.

In this document, we present our approach based on an Enterprise Architecture Framework to take into account the complexity of the system. These frameworks provide the capacity to model the system on several viewpoints to express the different concerns of such systems. The choice to use an Enterprise Architecture Framework, and the associated tool ArchiMate, seems to be the most relevant due to our system features and the capacity to extend and specialize the associated tooling. The ArchiMate tooling is built on top of MDE technologies which provide facilities to extend the language definition with sensor network domain-specific concepts and constraints Thus, we propose a metamodel to define the domain concepts, and the metamodel is the support to generate a new design tool called ArchiMO. In addition, we specialize the mapping approach between the layers of the ArchiMO tool with the domain constraints to guarantee the model consistency regarding the domain. This resulting model is processed by a model compiler to generate a simulator code to achieve a simulation execution. The results of the simulation are used to analyze and validate the model of the system. After that, the iterative approach can be applied to improve the model regarding the requirements of the system, or to go forward in the development process.

Our approach and tooling are demonstrated with an example from the marine observatory domain on underwater moving object localization with several acoustics sensors. This use case is used to validate our tooling to model the system, ensure consistency of the model and finally simulate the model. Through this use case, we observe that our

tooling helps to reduce the complexity with the three viewpoints in the model, to improve the design activity via the domain constraints which ensure the model consistency of the Marine Observatory.

As conclusion, this work aims to demonstrate that we can improve the development process of complex system based on the use of MDE technologies and a domain specific modeling language with the associated tooling. The major improvement is to provide an early validation step via models and simulation approach to consolidate the system design.

Acknowledgements

List of Figures

1.1 Conceptual Model of Architectural Description, from [THE16] 6
1.2 Centralized Fusion Architecture 11
1.3 Hierarchical Fusion Architecture 11
1.4 Distributed Fusion Architecture 11
1.5 Comparison among SN Design Approaches according to the Requirements of SN Designer 15

2.1 Layered Architecture of MDE, from [Sof15] 23
2.2 Models Transformation 25
2.3 Classification of Enterprise Architecture Viewpoints, from [THE16] 27

3.1 Architecture Development Method (ADM), from [Gro09] 33
3.2 The ArchiMate Business Layer Meta Model, from [Gro09] 36
3.3 The ArchiMate Application Layer Meta Model, from [Gro09] 36
3.4 The ArchiMate Technology Layer Meta Model, from [Gro09] 36
3.5 The Business ArchiMate Concrete Syntax, from [Gro09] 37
3.6 The ArchiMate Relationships Concrete Syntax, from [Gro09] 37
3.7 The ArchiMate Business-Application alignment, from [Gro09] 37
3.8 The ArchiMate Application-Technology alignment, from [Gro09] 38
3.9 Layers of TOGAF 9 MetaModel, after [Gro09] 38
3.10 Comparison among ArchiMate and TOGAF 9 MetaModels 40
3.11 Compatibility between TOGAF ADM and ArchiMate, after [Gro09] 41
3.12 Architecture of NesC@PAT, from [oS07][ZSL+11] 42

4.1 Proposed Tasks and Approaches of Sensor Networks Design Process 54
4.2 Features and Aspects of Model Driven Engineering 55
4.3 Proposed Tasks and Approaches to be Performed by the Different Sensor Networks Designers using ArchiMate Layers 57

5.1 ArchiMate Business Layer 65
5.2 ArchiMate Application Layer 65
5.3 Communication Constraint between two Smart Sensors 65
5.4 Extended Relationship between Smart Sensor and Data Fusion Server . . . 66
5.5 Conceptual ArchiMate Relationships 70
5.6 Extended Relationship 71
5.7 Consistency between Business Layer and Application Layer 72
5.8 Generated MO Concepts and Relationships in the Application and Technology Layer 73
5.9 Generated ArchiMO Design Tool after the Extension of ArchiMate 76
5.10 Business and Application Layers (Palettes) 77

5.11 Extended SDR Relationship in Palette 77
5.12 Association and Assignment Relationships 79
5.13 Smart Sensor and Data Fusion Relationship is allowed 79
5.14 Smart Sensor and Data Fusion Relationship is not allowed 80
5.15 Generated MO concepts, Relationships and Constraints in the Application Layer after Entering a Proper Value 81
5.16 Mapping of Business and Application Viewpoints with Network Simulator . 82

6.1 Structure of MeDON - An Example: N=6, Y=3 90
6.2 Underwater Object Localization according the three ArchiMate Layers . . . 91
6.3 Model of a Dolphin Localization that is presented in the ArchiMate Business Layer 92
6.4 Model of a Dolphin Localization that is presented in the ArchiMate Application Layer 93
6.5 Model of a Dolphin Localization that is presented in the ArchiMate Technology Layer, from [All16] 94
6.6 Consistency between Business and Application Layers 95
6.7 The corresponding MO concepts in NS-3 96
6.8 A part of the animation through NetAnim tool after running NS-3, from [All16] 97
6.9 An example of an error detection in design model of Marine Observatory system, from [All16] 98

Introduction

General Context

A central concern in the area of computing has been the integration of digital artifacts with the physical world and vice-versa. However, recent developments in the field of embedded devices have led to smart things increasingly populating in our daily life: home automation (e.g. electrical objects control such light, gate alarm, heater, air conditioning, etc), smart cities (e.g. metro autopilot, real time error detections in the factory, etc), mobile apps (e.g. object localization) and earthquake detection. And not to forget the environmental monitoring context (e.g. ocean monitoring, atmosphere monitoring, etc).

All these systems are closed to the Web of Things (WoT) architecture and mostly relies on sensor networks (SN). SN are the base of infrastructure of the environmental monitoring systems. In [YZL+08], an environmental monitoring system is based on an integrated sensor concept that is structured to gather important data with signal processing hardware in one compact device. And these smart sensors are naturally integrated in a distributed system to manage the data processing, data storage and data presentation, mostly web oriented.

These systems present on the web clients various sensor data such as pressure, temperature, humidity, smoke, gas, and sound [LGS+05]. These presentation capabilities are coupled with data processing to provide high level services (e.g. localization of moving objects) based on the composition of basic functions. This means, multiple device types are configured on the SN, and we face set of different roles of each device, and different services according to each device.

Accordingly, the design of these systems has become increasingly complex according to the growing number of functionalities, processing tasks, sensors and the integration in an information system, including Internet. The design phases in the entire life cycle of such systems are concerned by the difficulties and challenges to map high level services and the set of functions on the SN architecture. These design difficulties may induce the designers to make architectural design errors during the design phase. More specifically, any error in the design phase can have serious consequences on the functioning and performance of the system.

In order to avoid the architectural design troubles, we tried to early validate the design of such created complex. For this purpose, we try to provide an approach that allows the SN designers to create consistent models to reduce the complexity of the design phase.

SENSOR NETWORKS

SN is applied and adopted in a wide range of applications in various areas (e.g. Smart Building, Transportation and Industrial Applications, Precision Agriculture and Animal Tracking, etc). The **sensor networks** term includes a wide scope and contexts, and it is difficult to define what exactly SN means [Cuz09][Mit04][Roo08].

Sensor Networks Definitions

[Cuz09] defines the Sensor Networks (SN) as "in which sensed data are periodically gathered at a single point, or sink, for external transmission and processing". This definition could be considered as an operation of two phases [SHL12]: (1) the observation/measuring, which means the accumulation of the gathered data at each sensor node; (2) transferring the collected data to some processing center (e.g. fusion servers) within the SN.

The scope of this thesis is the ocean survey which provides a continuous way of observing and monitoring underwater moving objects. This observation requires underwater environmental measurements and set of estimated states of the variable object [**?**][ZCKA09]. So in this context, Underwater Sensor Networks (UW-SNs) are required to be used. The UW-SN performs functions such as data acquisition, combining then transferring data from a device to another for all forms of underwater environmental monitoring [LKS08].

Sensor Networks Implementation

We distinguish several types of SN (e.g. non underwater wireless and wired, and the underwater wired). In order to implement these networks, series of connected monitoring equipments are required such as sensors, servers and communication infrastructure. These equipments have many requirements which vary according to the environmental constraints.

According to our thesis scope, we are involved in the underwater environmental constraints. For this purpose, during the implementation of the UW-SNs, several underwater environmental constraints should be taken into consideration. We cannot ignore the presence of underwater communication constraints that should be respected during the UW-SNs implementation, such as the type of the cable (marine cable) to connect a sensor to a server (physical constraint) or the required cable length between a sensor and a server (logical constraint). Otherwise, this may negatively affect the performance and tasks of the underwater communication, such as the delay while transferring data between sensors and servers [Ree15]. Thus, to deploy the UW-SNs with the necessary equipments, several requirements are needed [HLS$^+$05][HYW$^+$06]: acoustic communication such as marine cables between sensors (hydrophones) and workstation (fusion servers); network configuration (e.g. configuration of sensors and fusion servers); application (e.g. trilateration algorithm).

Adopted Definition of Underwater Sensor Networks

In order to fulfill the requirements of the connected monitoring equipments that are briefly presented in the previous section, we try to find the adequate definition. Relying on [EVG07a][MRI12][HSZ12], we can consider the following: (1) the set of the mobile/fixed nodes of UW-SNs reflect set of fixed software and hardware components that are connected together (e.g. sensors, hydrophones, algorithmic servers); (2) the data transfer between the different components reflects that the network is configured; (3) the monitoring of underwater moving objects reflects that a detection and localization algorithm should be deployed in order to process the location of this object. Based on these approaches, our definition relatively to our context:
"Underwater Sensor Networks is a set of anchored sensors with an underwater communication infrastructure which is designed to: get, exchange, control and combine data between different nodes. Then, processing and relaying information to provide high level of services such as the monitoring or the localization of an underwater moving object".

Complexity and Challenges of Sensor Networks

In order to achieve the observation and monitoring missions of a given area, the implementation of SN is required. SN is based on a set of specialized sensors with a communications infrastructure designed to monitor and record data at various locations. It consists of components (software/hardware) with varying levels of computational and communication capabilities with different communication protocols. It performs multiple functions that are assigned to each component [Mit04]. Therefore, SN is considered in [SCK11] as a complex distributed system.

Relying on [Cuz09][Roo08][BDF+05][LWW11], we distinguish three sources of complexity in the life cycle: the complexity of the system itself is related to the provided services and number of functions; the specification and design activities; and the system deployment. This complexity is related to:

1. Many different services could be defined with several quality of services on a large number of sensors and servers. Thus, SN designers face difficulties in satisfying the requirements of the SN design process.

2. Architecture of the system which contains different heterogeneous devices deployed on a distributed architecture to operate all the SN services.

3. Interactions and integrations between the software/hardware components and the core network (SN) that relies on communication protocols, such as exchanging and transmitting high number of messages between the different existing nodes/components in order to provide a high level service.

[Sri10] presents two challenges that face the SN implementation: (1) the system architecture, since there is no unified system and networking architecture to build different applications on top; (2) the hardware cost, since the current cost of each individual sensor unit is still very high. Therefore, and as we are involved in the underwater environment, the deployment of the set of underwater sensors (hydrophones) is a costly operation.

Also, this is due to the necessary equipments such as: specific boats, marine cables and experts in diving, etc. Additionally, we cannot ignore that the deployment operation is risky and the position of sensors and servers underwater should be in the right position where an error in meters may cause troubles in the algorithms.

Many stakeholders are involved in the SN implementation and they take part of sensor network's life cycle. A stakeholder is defined in [RW11] as: "a person, group, or entity with an interest in concerns about the realization of the architecture".

The design phase of the SN life cycle is divided into two main levels: behavioral (logical analysis) and networking architecture. Each level requires a designer that is different from the other one and this is for better network technologies deployment of a system. The design phase requires having different stakeholders according to their domains [Com]. In this document, among these stakeholders we focus on SN designers.

In order to face the complexity and the challenges of the design of such distributed system architecture, a support on design level should be provided to the SN designers to cover, elaborate and analyze all the aspects of a SN.

The goal of the design phase is to provide an architecture which includes all the aspects for our service architecture definition to the physical network infrastructure. To achieve this goal, one of the relevant approach is to provide abstractions of the final system to focus on the architectural purpose.

In this context, using a modeling language is adequate to focus on our purpose if this modeling language includes the appropriate abstractions. Several modeling languages are candidates to support the modeling phase from general purpose languages like UML, to more dedicated languages focused on network services like Enterprise Architecture Modeling Language (EAML) or a really focused one with a Domain Specific Modeling Language (DSML). DSML are really efficient if the focus of the model is clearly defined and accessible with a relatively small number of concepts.

We have seen that SN are dedicated context but many architecture purposes are related to the standard networks which support application services. Thus, an adequate trade-of for the modeling approach is to reuse existing or standard modeling language with a specialization on our SN context. In this case the models must be efficient to have early validation on the architectural hypothesis. In our context, the validation phase can be achieved by a network infrastructure simulation supporting the high level services of our application. This modeling and simulation phases could be iterative to improve the services mapping on the network infrastructure. For this purpose, we must have a powerful tooling to ease the modeling and simulation phases.

RESEARCH QUESTIONS

The research problem that drives this thesis deals with reducing the time and complexity of the design phase. To tackle this wide topic, we divide it into three research questions:

1. ***RQ 1 Design Process:*** To face the design complexity of SN applications, how can we provide an assistance to guide the design phase?

2. ***RQ 2 Modeling Language Definition Specialization:*** Is it relevant to use a domain specific language to enhance and increase the consistency and unity of the design?

3. ***RQ 3 Tool Building Process:*** How an efficient tooling support can be produced to achieve the main concerns of the design process?

Proposed Approach

We apply our proposed Sensor Networks development process to a complete SN case study. These types of applications can include underwater SN that consists in a variable number of sensors that are deployed to gather scientific data in collaborative monitoring missions [ZCKA09]. These applications aim to perform a continuous ocean monitoring.

To elaborate this process, we provide a modeling approach based on separation of concerns coupled with validation step based on a network architecture simulator. This model is an input to a model compiler to generate simulation code that runs directly in NS-3 network simulator [AKR13]. An early validation of the model is interesting in order to have during the development phase, a formal and validated model. The resulted simulation helps to obtain an early validation of the design model before any prototype or deployment.

Organization

We present in Chapter 1, the state of the art in SN development process and the current approaches to answer the research questions. We present the SN life cycle, we categorize and compare the existing approaches to answer the requirements of the methods and frameworks for SN design, and we indicate their limits.

One of the main features of these frameworks is the treatment of the complexity through viewpoints. We present two fields: Model Driven Engineering, in Chapter 2, and that of Enterprise Architecture Frameworks, in Chapter 3. One of the main requirements of methods for SN design, regarding the SN development environment, is to have an overall model of SN design process which integrates all business, and technical activities. This type of representation is one of the goals of Enterprise Architecture Frameworks and modeling languages (Chapter 3). Here, we present and discuss the main frameworks and Modeling Languages of Enterprise Architecture, from which we select TOGAF and ArchiMate as the most appropriate for our purpose. Both advantages and limitations of applying Enterprise Architecture to SN development process are discussed.

Another main concern of both methods and frameworks for SN design process is related to have specific languages for their tasks and domain in order to facilitate the task of the SN designer. One manner of producing these languages is through the

Model Driven Engineering approach, presented in Chapter 2. Moreover, Model Driven Engineering is based on models and generative techniques; hence it enables to rapidly obtain prototypes of SN models to validate it. The chapter ends with a general discussion on the advantages and limitations of using Model Driven Engineering for SN.

We continue by presenting our proposals to answer the identified research questions in Chapter 4 as an answer to ***RQ 1 Design Process***, we introduce a SN design process inspired from a software development life cycle. In order to design, domain specific modeling languages are proposed for the designers of SN, in Chapter 5; it's an answer to ***RQ 2 Modeling Language Specialization***. As an answer to ***RQ 3 Tool Building Process***, we build a dedicated tool for each role of the process stakeholders, we propose a model-driven tool building process, in Chapter 5. In fact, this process can be used to extend a standard modeling language by several DSMLs and associated tools. To validate the proposed tool process in the SN Environment and demonstrate their capabilities, we provide an application of the SN design process and tools to a complete marine observatory case study from a modeling phase to a validation phase based on a network simulator, in Chapter 6.

And finally, we conclude the document by an analysis of our results and by suggesting some future works.

Part I : State of the Art

To answer the research questions identified in Section Research Questions. First, we survey the current approaches for sensor networks development, in Chapter 1. Then, we identify the limits of these approaches and inquire two fields, that of Model Driven Engineering, in Chapter 2, and that of Enterprise Architecture, in Chapter 3, that can supply solutions to cross them.

TELECOM Bretagne

1 Sensor Networks Development Process

Reminders and Objectives

Based on the concept of SN (Sensor Networks) defined in the introduction, we state that SN are mainly software intensive systems with a network infrastructure having a central place. This kind of systems requires the definition of the life cycle, with a main focus on the software development life cycle. In this chapter, we present the entire SN life cycle and emphasize the importance of specification, design and development phases. Next, we motivate the focus on the design phase including early validation. Then, we present the different stakeholders involved in the phases of SN life cycle and especially for the design phase. Later, we elaborate on the roles of each stakeholder in the life cycle, to specify the context of our concerned stakeholders. Next, we detail the structure of SN systems to identify their software and hardware components. After, we present and discuss the algorithms and the approaches that are adopted while providing the services of SN by using the identified components. Then, to design the specified application, we specify the needs of the selected stakeholders such as software designer or network designer. Next, we select the requirements regarding to design tools, according to these needs. After, relying on the identified needs, we compare the existing approaches that satisfy our requirements of the concerned stakeholder and analyze the limits of these approaches. According to these comparisons, we suggest possible solutions and methodologies to fit the stakeholder's unsatisfied.

1.1 Sensor Networks

Sensor Networks (SN) are made up of heterogeneous sensors with communication capabilities and heterogeneous components dedicated to an or several application domains. These systems are positioning on a large spectrum of domains like the environment monitoring, home monitoring, city monitoring, transport monitoring and so on. Nowadays this kind of systems includes more and more intelligence and becomes smart cities, smart transports and so smart systems.

To provide a generic definition of these systems, we are relying on the Systems of Systems (SoS) definition proposed by Jam al [Jam08], SoS are:"large-scale integrated systems which are heterogeneous and interdependently operable on their own but are networked together for a common goal".

This goal is defined and achieved with the support of a system engineering process which is based on [RWO03]:

1. The requirements or needs that the solution should fulfill, must be explicitly defined.
2. We design and develop the system to cover the defined requirements.
3. We verify that the system meets the defined requirements.

Based on this generic approach, requirements are allocated to the system conceptualization which starts by a functional analysis with the associated modelling activity. Then, an allocation of the functions on a logical and physical architecture is performed to take into account functional and architectural aspects.

1.1.1 Sensor Networks Life Cycles

One of the crucial phases in the SoS life cycle, is the allocation of the functional architecture on the SoS architecture which is logical and physical one. This phase includes an architecture exploration to target the requirements of the SoS. Due to the problematic and the definition of SoS in [Jam08][CF01], SoS are: "large-scale concurrent and distributed systems that are comprised of complex systems", and the associated life cycle is considered as a distributed systems life cycle (DSLC) to take into account the geographic, operationnal and managerial independence and the different temporal evolutions. This latter can be considered as a distributed systems life cycle (DSLC).

To define a DSLC life cycle, we should take into consideration two main types of phases: (1) for development purposes such as the design and analysis phases; (2) for implementation purposes such as the deployment phase. Accordingly, we distinguish several definitions of the DSLC. These definitions are closed to the definition of [GBN10] which defines a typical development process with seven phases:

1. Set Up Development Environment: preparing the needed frameworks and the tools to apply the designed models.
2. Connect Hardware: preparing the hardware platforms, the connection between the concerned devices such as the SN.
3. Prepare Interfaces/Libraries: preparing the libraries and the operating systems that are compatible with sensor nodes hardware versions.
4. Compile Code: use the compilers of the programming languages in order to build the executable code.
5. Implement Code to Hardware: deploy the executable code on devices/nodes.
6. Evaluate Effects: testing the functionalities of each node and node networks.
7. Repeat from the Fourth Phase: continuing by iteration from the phase 4 till the end while errors are detected in order to fit the application requirements.

Several SN life cycles adopt similar phases to the ones listed above using different names and/or numbers. For example, [EG10] adopts eight phases which are similar to the adopted phases in [GBN10]. Also, it adopts the iteration action between the different phases.

More to identify the preferable life cycle that can be performed for our SN context, we want to highlight some suggestions that [GGB+10] insists on:

- The need for several distinct forms of testing during the design phase. So, a designer can apply within the same model different assumptions, architectures and parameters.
- The focus on the deployment evaluation, avoiding the mistakes and complexity in deployment phase and including an evaluation phase at the end of the life cycle to check how successful the deployment actually is.
- The importance of iteration: errors are detected during testing or deployment requires to re-implement modules of the system, an iteration can continue until application requirements are satisfied. There are different possibilities of iteration between the different phases. For example, in case the detected errors are related to the architecture defined in the design phase, they can be reconsidered rapidly by iterating.

The presented suggestions above elaborate the issues that can be occurred during the design and deployment phases of the SN life cycle. Also, they elaborate that, by adopting the iteration approach, the errors that may be happened in the deployment phase can be prevented by detecting them in the design phase. This reflects the need of having a efficient support for the tasks that should be performed during the design phase, in order to reduce the complexity of performing these tasks. This support can be provided by emphasizing on the allocation of software functional components on the physical architecture with minimum architectural errors. Then, by having a tool to validate the defined physical architecture. For this purpose, we focus on the design phase among the different phases of the SN life cycle, in the next section.

1.1.2 Design Phase of the Sensor Networks Life Cycles

The allocation of the software functional components on the physical architecture, is considered as a complex task to be performed. This due to the different concerns (viewpoints) that are involved to perform this task. Relying on IEEE Standard 1471-2011, a viewpoint "codifies a way of addressing some architectural concerns in terms of notations, kinds of models or other forms". And, it is defined as: "a collection of patterns, templates, and conventions for constructing one type of view". A view is specified by a viewpoint, which prescribes the concepts, models, analysis techniques, and visualizations that are provided by the view. A view is conformed to the definition of a viewpoint (see. Figure 1.1). In general, a system architecture description is defined in a view that addresses a set of related concerns related of stakeholders.

An architecture concern is defined in [RW11] as: "about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture". [KK07][KK03] define architectural concern as: "groups aspect designs and can be seen as a software architecture view-point". Thus, we need different experts in software architecture who have different domains of experience in order to address the different view-points. And also relying on [RW11], we consider that a software architecture is related to several stakeholders that have different roles in the life-cycle related to their domains of experience. If the separation of concern is efficient to model software architectures, one of the drawbacks is to ensure consistency between these viewpoints, for example to allow the mapping between the logical and physical components.

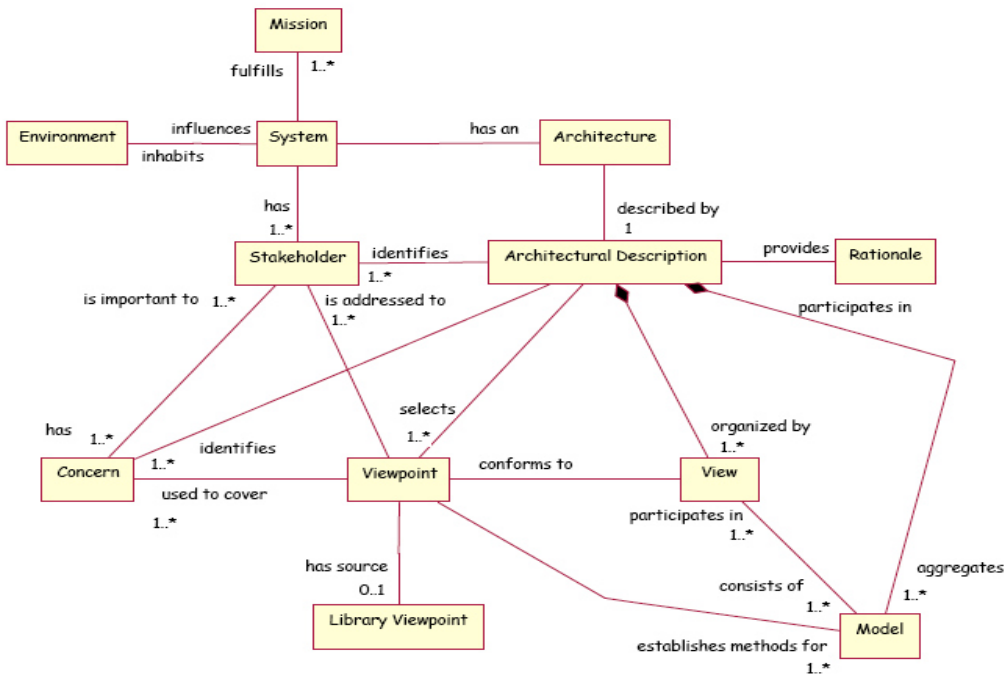

Figure 1.1: Conceptual Model of Architectural Description, from [THE16]

1.1.3 Roles in the Sensor Networks Life Cycles

As we have just described before, different stakeholders are involved in the life-cycles and their roles are tightly connected with the view-point definition that are related to. To refine our definitions related to the SN life-cycle we identify the stakeolders and their roles particularly involved during the design phase which remains a critical phase of these systems.

Zuniga and al, [ZD13], identified clearly the main contributors in this phase with the following stakeholders:

1. Domain Expert: provide the technical specific events, actions and services that are related to the domain, to be used by all applications in the domain.

2. Software Designer: define an architecture of the sensor network applications by specifying software components, distribution of actions, events, services such as a service includes functions and procedures, and relationships between components.

3. Network Designer: design the required network and deploy it on hardware components on the whole system. We cannot ignore the possibility of generating binary codes and configurations of the hardware components.

All these stakeholders with their own expertise, contribute to the design of the SN system. However, to improve the definition of their roles and tasks, we will look for the use of several SN systems in SoS.

1.2 Sensor Networks Systems

To consider the use of SN systems, we can try to know how the sensor datas are used to provide a higher level service to target customer needs. The systems or SoS which deliver continuous customer services are like NEPTUNE Canada, American GPS System and European GALILEO System and Russian GLONASS System. Such systems provide important information on the properties and behavior of complex systems [VG09]. High level services, such as continuous observations and processing of weather parameters, or continuous observations of moving objects, and many others [VG09], are based on a wide range of functions [Mil07] on data like acquisition, tracking, routing, fusion, distribution, storage and querying. The resulting information can be consider as unlimited sequence of complex data items obtained from the sensor nodes and ready to be analysed and stored [Mil07][VG09].

For example, NEPTUNE Canada (North East Pacific Time-series Undersea Networked Experiments) enables real-time study of: tectonic plates; the movement of fluids on the ocean floor and the effects of climate change on marine ecosystems. Also, it will help accurately locate earthquakes and observe the resulting seismic stress. In addition, it measure salinity, carbon dioxide and even movement of organisms in sediments. Then, The multitude of data collected will be transmitted via optical fiber to the Port Alberni station on Vancouver Island, which will connect them via a broadband internet connection at the University of Victoria. They will be available on the NEPTUNE website (Canadaneptunecanada.ca). The data archiving and management system used by NEPTUNE, also collects information from VENUS (Victoria Experimental Network Under the Sea), a coastal network of cable-linked underwater observatories. Thus, NEPTUNE Canada is the largest ocean observatory in the world.

In order to provide these services, NEPTUNE Canada deploys set of connected nodes and instruments that must be anchored to optical cables on a SN. These nodes and instruments can be: Hydrophones, seismometers, video cameras and high resolution cameras. Also, several other sub-systems are deployed and connected to the SN, such as: database, database replication and web server. A Database is deployed to fuse the

sequence data items that are coming from different sensors. A Database replication is deployed to have a copy about the entire collected and analyzed data. A web server is deployed to diffuse and display the data on web. Regarding the router, is deployed to can access the internet network.

Therefore, such systems are not only a simple SN, they are systems equivalent to enterprises with high level services delivered by a high number of interconnected heterogeneous components (software and hardware) [Mil07]. For example Neptune system is not only a SN, it can be considered as enterprise system since they require several heterogeneous components (software and hardware) with communication constraints to be deployed on the SN in order to provide services that are similar to the mentioned above.

These information systems are categorized according to two main types: wireless outdoor and underwater [Wan08][WBH+13][MMD15]. Such information systems can be divided into non-acoustic and acoustic types [Ini12]. In both types, a service of localization and tracking moving objects is available through a set of sensors and appropriate algorithms. These systems adopt two main approaches [Wan08][Ini12]:

1. Identifying the cell (circle or sphere) in which the mobile object is located. The position is calculated relying on one sensor.

2. Identifying the area of the moving object which is deduced by the intersection of a minimum of three cells. The position is calculated relying on two or more source (receivers) independently of the sensor technology [CGLP05].

Based on these approaches, high level services like series of moving object localization can be provided by elementary services gathering. Mainly these services include data fusion approaches that require a set of receivers (sensors) [CL10]. This approach is used in many applications (e.g. localization systems) where a large amount of data must be combined or fused to obtain relevant information [LCD+14].

Data fusion algorithms are differentiated and categorized according to the source number (two, three or many receivers) [Hot13][BONL08][Wan08][SHS01]. Relying on [KH05], the position accuracy of mobile object increases with the number of receivers. The efficiency of the localization service is highly correlated with the increasing of the number of sensors. For this purpose, we find out several types of fusion algorithms that are differentiated by the number of sensors in the network.

In addition, we focused our interest on the fusion algorithms and the associated services because they are quite representative services included in the SN and the extended systems. Indeed, they provide a high level service in many SN and they are based on a undetermined number of sensors and several deployments of elementary algorithm nodes exist. This means that, we exploit many aspects of SN, and many aspects of information system to display and store the different results (locations) that are provided by performing the different localization services.

1.3 Fusion Algorithms

To provide a pertinent example of a high level service for the SN, we study fusion algorithms as a representative service in a SN. Effectively by nature, this kind of algorithm is necessarly highly distributed based on heterogeneous components (sensor nodes and fusion nodes) and several architectures are possibly déployed on a selected network infrastructure. Due to all these features, it seems to us that fusion algorithms are good candidates as representative SN applications and so by extension how can we manage the design of this kind of application.

On this hypothesis, we distinguish several fusion algorithms such as: trilateration [HJS+12], triangulation [EVG07b][CGLP05][HJS+12], Bounding-Box [EVG07b][HJS+12], set-membership [CGLP05], and Dive'N'Rise(DNR) [EVG07b][HJS+12].

As the extension of trilateration to more than three sources is possible, we select it to be adopted while providing the localization service of moving objects [ASMT+12]. This selection provides us the possibility of increasing the accuracy of the determined position of a moving object. For this purpose, the trilateration algorithm is used by the most relevant positioning system in the world such as, the GPS [RZ00][Wan08]. This algorithm combine and integrate information from a number of different sources (sensors) using an architecture for data fusion approach [Cas13].

However, several Data Fusion Architectures (DFA) are possible to be used while performing the fusion algorithm. This diversity of DFA types generates the possibility of creating different models by SN designer according to each type, with a difficulty to choose the most convenient one. This designer should select the adequate architecture for the components that are required to perform this algorithm. So, the decision of this designer is totally oriented toward selecting the appropriate data fusion architecture among several. In order to select the appropriate one, the properties that can fit with these architectures should be identified. Then, the SN designer checks the availability of each property in the different data fusion architectures, in order to select the most satisfied architecture. For this purpose, we present these properties, in the next section.

1.4 Properties for Selecting a Data Fusion Architecture

Several challenges face the SN designer to adopt the data fusion approach and its logical architecture and the relevant mapping on the distributed network infrastructure. We distinguish the following difficulties and challenges:

1. Different heterogeneous components should be deployed on a SN by adopting a data fusion architecture. Multiple functions are assigned to each component on the SN [Mit04]. This requires a communication infrastructure with varying levels of computational and communication capabilities according to each communication protocol.

2. The number of sensors should increase upon the needed degree of object localization accuracy. This may cause the needs of increasing the number of other components such as the fusion servers. Thus, the number of the heterogeneous components on the SN is dynamic and can be increased at any time without component constraints.

3. The impacts of the sensor failures on the network should be minimized. This due to the long life duration that is required by the sensor networks systems [RGK+11]. This failure can affect the entire or a part of network. So, the lifetime of SN can be optimized.

Accordingly, we can detail the properties that are required by the SN designer, in order to select the appropriate data fusion architecture. This last should provide the capacity of adding a large number of sensors and fusion servers on SN. This capacity increases the accuracy of localizing a moving object, or enlarges the covered observed area. Also, it minimizes the network failure by replacing the failed components (sensors or fusion servers) by stable and functional ones. In addition relying on [KH05], more sensors and fusion nodes are required to be deployed on the SN to perform and optimize the fusion service. So, the SN designer requires an architecture which has enough flexibility to cover the needs of adding or removing components with the less impact on the localization application. These modifications on the application architecture must be performed without calling into question the system architecture and are inevitable on a long time period of the system life cycle.

1.5 Data Fusion Architectures

In the SN domain, the sources of information are the sensors and the fusion algorithm is often done in a fusion center that collects this information [EZ14][Mit07][LIHL17], but it can be also decentralized [EZ14] with many possibilities between these two kind of architectures. The main possibilities for the architecture categories are centralized, hierarchical and distributed [And13]. In a centralized architecture, there is a single fusion node [LLL09][LCD+14][LCK+97], see Figure 1.2. Sensors acquire data and transfer them directly to a central single fusion node. In a hierarchical architecture, figure 1.3, the fusion nodes are classified in a hierarchy with the higher level nodes processing results from the lower level nodes and possibly providing some feedback [LLL09][LCD+14][LCK+97]. In a fully distributed architecture, see. Figure 1.4, there is several fusion nodes. Each fusion node sends information to the other fusion nodes [KGC][CM05][Pao94]. There is no predetermined hierarchical relationship so each fusion node can communicate with any other node.

1.6 Limits and Comparison among the Different Data Fusion Architectures

In a fully distributed architecture, the number of sensors and fusion nodes that is deployed on the network can be changed as required [And13]. More we add sensors, more the accurate position of moving object increase, or more the observed area is enlarged [KH05][Vey16]. Consequently, the distributed fusion architecture allows to add sensors and fusion nodes on SN as much as it is required to enhance the results of the provided

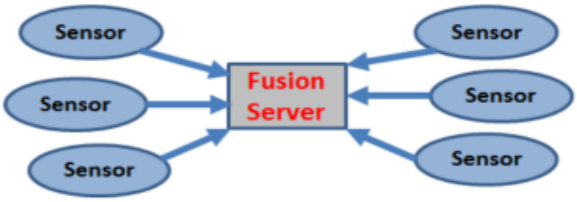

Figure 1.2: Centralized Fusion Architecture

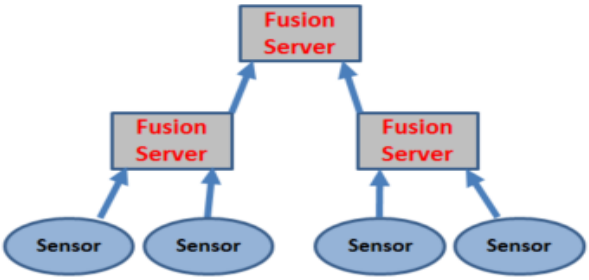

Figure 1.3: Hierarchical Fusion Architecture

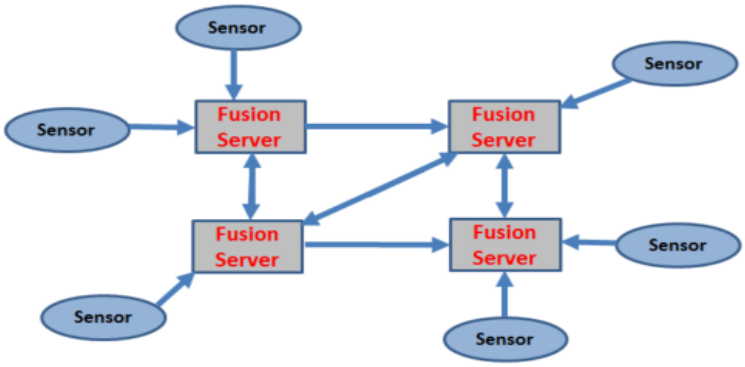

Figure 1.4: Distributed Fusion Architecture

services such as the localizations of moving objects.

In a centralized architecture, the number of sensors and fusion nodes that is deployed on the network cannot be changed as required [LLL09][LCK+97]. This is due to the structure of the centralized architecture as one single fusion node is allowed to be deployed on the SN. So, this architecture cannot contribute by increasing the accuracy of object localization result. Consequently, with the centralized fusion architecture is not possible to deploy more sensors and fusion nodes on SN.

In a hierarchical architecture, we can add more sensors and fusion nodes but it is a complex task. This is due to the required feedback scenario (communication constraint) that should be performed between the sender fusion node and receiver fusion node [LLL09][LCD+14][LCK+97]. So, the communication constraints increases in parallel with the number of added sensors and fusion nodes. Also, due to the required hierarchical order of the location of fusion nodes presented above. So, in case we add more nodes, we face a problem in the location of these added nodes as we should respect their order of communication, from the higher to the lower nodes. It is not an easy task as any error in the location of the added nodes may affect negatively the performance and the result of the object localization.

Consequently adding sensors and fusion nodes in a hierarchical fusion architecture is allowed but faced up to several difficulties in locating the newly added nodes in the right positions on the SN.

Accordingly, the property of adding several components on the SN is available in the hierarchical and distributed fusion architectures. In the distributed ones, the fusion nodes can be added at any location on the SN without any hierarchical level of execution. This means all the fusion nodes can communicate together without any execution cauisality. So the distributed architecture provides the more flexible approach when we take into accounts the long time period of the system life cycle.

Relying on the analysis and the comparison discussed above, the SN designer should take into consideration the distributed fusion architecture to perform the trilateration fusion algorithm. Thus, the distributed fusion architecture is the most generic architecture related to our requirements.

In addition, the selected architecture should be compatible with the selected trilateration fusion algorithm [LCK+97]. According to [AR07], this algorithm adopt and use the distributed fusion architecture.

Making a SN durable in the case of individual sensor failures is another main advantage of selecting the distributed architecture. In a distributed environment, the loss of one sensor does not affect the entire SN [KGC][CM05][Pao94]. So, there is no component that constitutes a weak point whose stopping paralyzes the system [And13].

For the purposes presented above, the distributed fusion architecture is the prefer-

able one among the others. This due also, to the provided advantages by this architecture such as reliability and energy efficiency [LCK+97][Vey16][KGC][CM05]. Therefore, it seems to us that consider a system example based on a trilateration algorithm (with minimum three sources) performed by a distributed fusion architecture is a relevant and generic example for illustrating the design problematics.

1.7 Requirements for Designing Sensor Networks Systems

The distributed fusion architecture is adopted by the most SN systems. In order to perform the design phase of such systems, several challenges face the concerned SN designers. A model should be defined to describe and analyze how the provided service (e.g. localization of moving objects) can be mapped on a SN. This requires the intervention of different SN designers that are experts in different domains of experience. The provided service is required to be described in a model by a SN designer which is an expert in the business process of such service. To perform such service, a model that contains the corresponding software components and relationships is required to be defined by another SN designer which is an expert in the application process. To implement these software components and relationships, a model that contains the corresponding technology platform is required to be defined by an expert in network infrastructures. In addition, as the SN designers define models in a complex context, they may make architectural design errors. So, the created models must support the analysis of architectural constraints which must help the designers and avoid errors that would be detected during the next phases of process development.

For this purpose, while designing such systems, the SN designer should take into consideration two parts: (1) services that are related to business domain such as localization of moving objects; (2) information systems to support the deployment of these services and to provide enough flexibility to take into consideration the long life cycles of these systems.

In order to design such information systems, the SN designer must be supported by an appropriate tooling [HML02]. Due to the analysis of [RBR10][DNR03], we can consider that the design approaches could be supported by graphical modeling languages. These approaches and their related tools aid in the design and deployment of SN applications [RBR10][MFHH02]. These tools provide the ability for the SN designer to analyze and model complex systems such as Neptune Canada [ZSL+11]. Thus, the requirements of SN designer can be coupled with the requirements of this tool. These requirements are identified as:

1. ***Requirement 1 - Improving Architectural Design:*** Possibility of preventing the architectural errors that may be made by the SN designer. These errors can occurred while defining the provided services that are performed by the adopted SN architecture regarding the communication constraints. For example, we prevent architectural errors by avoiding connection of two sensors, or a connection between

a sensor to a database server without any processing node.

2. ***Requirement 2 - Multiple Viewpoints:*** Providing each designer the ability to work independently in a viewpoint (cf. section 1.1.3), in order to have his proper model according to his domain of experience. The different concerned designers cooperate together in order to share the design of the same system. And these different viewpoints must correlated in order to have a consistent model. For example, the domain expert, software designer and network designer create their models in their independent viewpoints (cf. section 1.1.3), so they cooperate together by interrelating their models to obtain this consistent model. This interrelation can be created by using specific relationships to ensure model consistency of this unique model.

3. ***Requirement 3 - Extensibility:*** Ability of adding new SN specific elements, constraints and relationships in the design tool. The absence of such components while defining the scenario of the provided services affects negatively the design phase, by increasing the number of required components, communication constraints and relationships that should be performed manually by the SN designer. For example, by using an added specific SN component or relationship to the design tool in a viewpoint, the SN designer can get automatically the corresponding related components and relationships in another viewpoint. These generated components and relationships can be built-in or added in the design tool to help the designer related to the complexity of the resulting information system.

4. ***Requirement 4 - Heterogeneity Supported:*** Ability of having different types of components and communication types in the same defined SN model. As the SN is an information system which is set of heterogeneous devices and communication protocols, the presence of this ability while defining the scenario of the provided services is necessary. The defined SN model contains different communication types between different heterogeneous components. For example, the communication protocol between a sensor and a database server is different than the one between a server and a web server.

5. ***Requirement 5 - Validation Tools Supported:*** Ability to verify the created SN models by simulation. This ability is required in order to detect the architectural errors during the design phase, and to validate the created models as early as possible in the development life cycle. For example, using a network simulation tooling could efficient to evaluate the mapping of a service on a SN infrastructure.

1.8 Limits and Comparison among the Different Approaches of Sensor Networks Design

In order to select the proper approaches to design then implement a SN system, we discuss a comparison between the different approaches regarding our requirements, identified previously. Our comparison is based on the most relevant approaches regarding our context: SimStudio [TTH11], CA-PSCF (Context-Aware Pervasive Service Creation Framework) [AYG10], DSM (Domain-Specific Model) [VMP14], ITSML (Intelligent Transportation Systems Modeling Language) [FIFF15] and SysWeaver [RBR10]. The results of our comparison is summarized in the Figure 1.5 and detailed in the next sections.

Requirements / Approaches	Improving Architectural Design	Multiple Viewpoints	Extensibility	Heterogeneity Supported	Validation Tools Supported
SimStudio	✓	X	✓	✓	✓
CA-PSCF	✓	X	✓	✓	X
SysWeaver	✓	X ✓	✓	✓	✓
DSM	✓	X	✓	✓	X
ITSML	✓	X	✓	✓	✓

Figure 1.5: Comparison among SN Design Approaches according to the Requirements of SN Designer

1.8.1 Approaches of Architectural Design Improvement

This section presents the five approaches regarding the improvement of the architectural design of SN. The five approaches aim at integrating in a single environment, tools for modeling and validating high level of pervasive services. These services can be related to complex contexts such as SN. These approaches proposed new specific SN concepts that are implemented in different frameworks and tools. According to each approach, these concepts are dedicated to be used while designing SN systems by the SN designer.

The SimStudio concepts are built by developing the generic DEVS (Discrete Event System Specification) concepts [TTH11]. The CA-PSCF concepts are based on the EMF-based concepts [AYG10]. The DSM concepts rely on Jersey JAX-RS [VMP14]. The ITSML concepts have been developed using the infrastructure of INGENME [PGSP11][FIFF15]. Therefore, during the design phase, the SN designer can use these new specific defined concepts. These concepts are created for specific domains and not for general purposes.

However, they did not propose constraints on these concepts or on the communication between them. The implementation of such constraints in the concerned design tool can help the SN designer to improve the architectural design by preventing architectural design errors that maybe made by him. For example, the SN designer is not able to connect two sensors together, but he is able to connect a sensor to a fusion server by respecting the required criteria to establish this connection.

Consequently, these five approaches help the designer by providing the ability to

have SN concepts and not only general purposes concepts. However, the lack of the implementation of domain specific constraints applied on the domain concepts and relationships is not powerful support for the designer. With this kind of tooling, the designer hasn't enough support to prevent the architectural design mistakes.

1.8.2 Approaches of Providing Multiple Viewpoints

This section presents one main approach that enables the SN designer to define a SN model that contains different viewpoints. These viewpoints are defined by the different concerned SN designers according to their different domains of experience. This approach is SysWeaver [RBR10].

SysWearver is a very effective approach, it aims at controlling the sensor networks by controlling its programming. It seeks to go up in abstraction, it works from top to bottom. It focuses on the application level that is based on programming, and the technology level that is based on simple network components. However, it misses advanced and complex networks components. Accordingly, a main question could be asked here which is: is SysWeaver can manipulate information systems (e.g. SN systems)?

SysWeaver cannot be the proper approach to be adopted by the SN designer. This is due to the two main issues that are required to implement such information systems (e.g. NEPTUNE localization systems): (1) advanced and complex networks components (e.g. web servers, routers, database replication server) should be implemented in the technology level of SysWeaver design tool, which are required to be used when the designers describe the business process of a complex service such as localization of moving objects; (2) the descriptions of all the possible provided services are required to be defined using one or the different data fusion architectures. The second issue requires a business level design that contains specific SN components (e.g. fusion algorithm, data acquisition) to be used while designing such services. Therefore, a business level design and advanced IT components are required to be used while designing localization systems.

Consequently, as a solution, we can exploit SysWeaver by extending with a business level that meets the needs (e.g. specific SN components and relationships) to describe a complex service. And, by adding to the technology level, IT components that can be the correspondences of specific SN components in the business level. Another approach can be proposed, which is to reuse a design tool that contains business, application and technology levels. Then, each level can be extended and specialized by adding new SN concepts and relationships that can be implemented in the concerned design tool. Therefore as we present later, it seems to us that this last solution is less complex if the tooling respects the extensibility requirements.

1.8.3 Approaches of Offering Concepts Extensibility

This section presents the five selected approaches regarding the ability to extend new concepts while designing a SN. These approaches enable the extensibility by adding new concepts (e.g. Sensors, Servers, Relationships) to the concerned frameworks or

design tool. This is similar to [CAKR11][CKR12]. Due to the created or generated editors/design tools that are used to create models according to each approach, these new concepts can be new SN specific elements, constraints and relationships in these design tools. This gives the SN designers an easy way to use the new specific added components and constraints.

Consequently, these five approaches help the SN designer by providing the ability to use built-in SN components and relationships from the SN generated design tools. However, modifying a tooling remains difficult when we want to respect the semantics of the added concepts and relationships.

1.8.4 Approaches of Supporting Heterogeneity

This section presents the five selected approaches regarding the support of the concepts heterogeneity while designing a SN. The approaches provide the SN designer the possibility to use heterogeneous components (software and hardware)and relationships while designing SN systems. According to each approach, this possibility is provided by using the generated SN editors/design tool that contains different types of components, such as sensor, database server, fusion algorithm. These components perform different functions and provide different services.

Consequently, by supporting the heterogeneity of concepts and relationships that can be implemented in the SN design tools, these five approaches help the SN designer by providing the ability to define SN models that contains different types of components and relationships.

1.8.5 Approaches of Supporting Validation Tools

This section presents only the three approaches that provide tools to validate the defined SN models. These approaches are: SimStudio [TTH11], ITSML (Intelligent Transportation Systems Modeling Language) [FIFF15] and SysWeaver [RBR10].

These approaches enable the SN designer to validate the defined models by detecting the architectural design troubles prior to the implementation phase. This early validation is performed by simulating the defined SN models.

Consequently, these three approaches help the SN designer by validating and verifying the result of the design phase (defined SN models). This can be performed by using a specific simulator that has the ability to provide results of the simulation task. Therefore, these approaches integrate the possibility to improve architectural design models after the analysis of the simulation result.

1.9 Discussion

According to the presentation of the previous section on the existing SN approaches and the associated tooling, we conclude the following:

1. ***Requirement 1:*** Improvement of architectural design of SN is available in all the presented approaches. Mainly, the frameworks are based on the defintion of domain concepts which is efficient support for the designer. However, we have noticed that these approaches lack the domain constraints relative to the domaine concepts and relationships. To improve the lack in the actual tooling and to keep the domain concept definition, we will looking for the relevance of Model Driven Engineering approach relative to our context, in the next section.

2. ***Requirement 2:*** Multiple Viewpoints is addressed by only one approach of all the presented approaches, the SysWeaver. It provides separate viewpoints according to each domain of experience, with the ability of inter-relating these viewpoints, in order to have one unique model. However, this approach is mainly focused on the design of limited scale SN and does not satisfy the requirement to design complex information systems such as Neptune. This is the reason why we study and analyze Enterprise Architecture frameworks in chapter 3 to try to identify a system approach including SN and the necessarily associate IT infrastructure.

3. ***Requirement 3:*** Extensibility is possible in all the presented approaches even if the associated tooling remains difficult to extend. Moreover, we notice that some tools are based metamodeling approach which guaranty a clear language definition to ease the language extensibility. However, this possible extension is not a guaranty to have a generating approach for the extended design tool with the added concepts and constraints. Once again to improve the extensibility and face this requirement, we will try to apply a Model Driven Engineering approach, introduced in chapter 2.

4. ***Requirement 4:*** Heterogeneity of components and communications in the design tool is well supported in the presented approaches. But multiple heterogeneous components is really available when the associated constraints are also supported to prevent design mistakes, as we noticed in the text relative to the requirement 1. So this requirement should be really satisfied in association with the requirement 1.

5. ***Requirement 5:*** Validation step is available in three approaches by using network simulators and indubitably these simulators improve the quality of the models and the resulting system. So we keep this kind of simulator to satisfy the requirement 5 and will try to include the simulation in our future approach and tooling.

Consequently, regarding the presented comparison and analysis; proposing a design process and tooling for the SN life cycle must take the best of each approach and propose a contribution which satisfy all the identified requirements and firstly the ***RQ 1 Design Process***. Our contribution will be elaborated and presented in chapter 4.

Synthesis

In this chapter, we presented the concept of SN life cycles with their advantages, features and options. Then, we pointed out our interest in the design phase of SN life cycle. Next, we defined the roles of the involved stakeholders in the SN life cycle, and we extracted that the SN designer is the proper stakeholder to be involved in the design phase. Then, we presented the structure of complex information system, and the challenges that may face the development of such systems. Thus, we identified that the SN is an information system and no only a simple SN, it is a complex distributed system. After, we selected that the distributed fusion architecture is the appropriate architecture that can be adopted for our SN context. Also, we specified that the trilateration algorithm can be implemented on the selected architecture. Next, we extracted the requirements of SN design. Then, the comparison between the existing approaches based on the specified requirements, shows that there is one approach which fulfills completely the identified requirements, but it is not enough according to our context, which is the SysWeaver approach. The **Requirement 2 Multiple Viewpoints** *is satisfied by the SysWeaver approach where the system to be designed is only a simple SN. However, this requirement is not satisfied where the system to be designed is a complex information system. To fit the satisfied and non satisfied requirements, in the next two chapters, we study and adopt the fields of Model Driven Engineering and Enterprise Architecture. We propose a process for SN Design, in Chapter 4. We provide a domain specific modeling language for SN, which is needed by the designer in order to create his specific model, in chapter 5. Also, we propose for the designer, a tool building process for SN design in order to have a design tool where he invokes the proposed metamodel, in chapter 5. Finally, to point out the applicability of our SN Design Process, we use it in Chapter 6 to model a marine observatory case study.*

2 Model Driven Engineering

Reminders and Objectives

We detailed the requirements of SN Design in the previous chapter, and we identified that **Req 1 Improving Architectural Design**, **Req 3 Extensibility** *and* **Req 4 Heterogeneity Supported** *are not covered by all the presented approaches. In this chapter, we focus on existing approaches that handle* **Req 1**, **Req 3** *and* **Req 4**. *In order to satisfy these requirements, we are looking for the Model Driven Engineering (MDE) benefits and try to identify the obtained benefits from using such approach. First, we give an overview of the MDE approach, to illustrate its capacity and to obtain models with an high level of abstraction. Then, we present the remaining MDE challenges related to our needs, in order to have an earlier validation of SN models. Next, we present the separation of concerns which is a key support in MDE and for SN development process. This support and our analysis lead us to think that MDE is a relevant approach for SN development. Next, we present and analyze the standards and tools of MDE. This analysis enables us to identify the relevant features of the tools required by SN design. Finally, we identify the advantages of using MDE related to our requirements of SN design.*

2.1 Model Driven Engineering Fundamentals

Regarding our context and the achieved tooling analysis described previously, we present in this chapter how Model Driven Engineering can contribute to improve the design of the SN and the associated information system. The fundamental concepts of MDE are: model, metamodel and model transformation [Par12][vdB09].

A Model is a simplified view of a system for a targeted concern. The goal of a model is to describe and to improve the understanding of the system often on several abstraction levels. A model selects interesting concepts on view points for a given context, and provides a representation of the reality for a dedicated purpose [Par12][vdB09].

As we previously presented in chapter 1 (cf. section 1.1.2), to create models, the SN designers need to use modeling languages. Therefore, we will elaborate on the main existing modeling languages in the section 2.2.1. Modeling languages are defined by

semantics, an abstract syntax and a concrete syntax. There are several approaches to formally define an abstract syntax of languages [KRV07]. In the modeling context, metamodels are used to define the abstract syntax of modeling languages.

A metamodel is defined in [GKH07][OMG15] as: "a model that defines the language for expressing a model". Metamodeling is a popular method that define the abstract syntax of domain specific modeling language (DSML), because the language designer can directly map the classes of a domain analysis to classes in the metamodel [EIS]. Associations and inheritance of domain classes are also mapped to the language definition. For this purpose, in order to define Modeling Languages such as UML or DSML, we can adopt the metamodeling method [GKH07][OMG15][CEKS01][KBJV06].

Based on this metamodel definition for a specific domain of interest [PMDC+07][Par12][vdB09], we can instantiate a large number of models that conform to this metamodel [Béz04][Par12].

Precisely, a metamodel is not a model of model and is not strictly a language. A metamodel is a model that defines a language to define models, an explicit and concrete definition of a language. In the four layers approach (Model0, Model1, Model2, Model3 in figure 2.1) promoted by OMG standard organization, each layer is conformed to the upper layer, from the real system to reflexive layer (Model3 layer of the figure 2.1).

Model Transformations (MT) is a set of rules applied to parts of a metamodel elements [vdB09]. The transformation engine reads a source model, which must conform to a source metamodel, and applies the rules defined in the transformation model to create the target model that will conform to the target metamodel.

According to [FR07], we distinguish two main kinds of transformations: Endogenous, where the source and models conform to the same metamodel such as UML Model to another UML model [vdB09]; Exogenous, where the source and target are defined in different metamodels such as UML Model to java program [vdB09].

2.2 Model Driven Engineering Aspects

Van Der Straeten and al have identified major aspects in MDE which are [VDSMVB09]:

1. Requirements Modeling: transferring the specified business requirements to functional requirements that describe the functionality of the system (each role/function), using modeling languages. The created models may contain different types of elements and relationships, such as functions, data, actors, association relations and triggers.

2. Modeling Languages: necessary needed languages, methods and principles to design specified metamodels in order to build domain specific modeling language (DSML), and to provide specific concepts for designing complex systems [FR07].

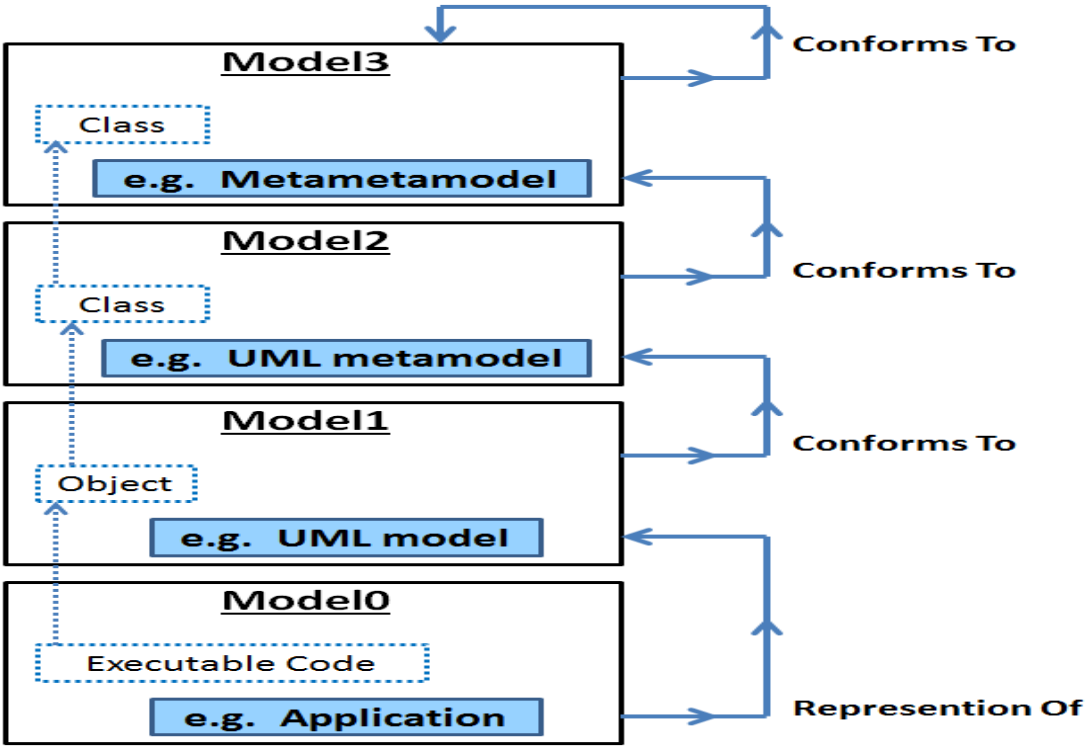

Figure 2.1: Layered Architecture of MDE, from [Sof15]

3. Model Heterogeneity and Quality: developing models by different stakeholders in a distributed architecture, using multiple viewpoints that utilize possibly heterogeneous modeling languages. In other words, models could be built using a variety of domain specific modeling languages [FR07]. Also, ensuring a correspondence between inconsistent quality aspects in and between viewpoints [Rec08][FR07].

4. Models Validation: verifying and testing the models and the code generated from those models. This topic will be elaborated on and discussed in chapter 6.

5. Models Transformations: converting models from one type to another, from one extension to another.

6. Run-time Models: executing models during analysis, design, implementation, and deployment phases of development life cycle [FR07].

Related to SN design, we will focus on three aspects in the coming sections: Modeling Language, Model Heterogeneity and Quality, and Model Transformation.

2.2.1 Modeling Languages

We distinguish two types of modeling languages: the ones that are used for general purpose such as UML, which could be adopted in any domain; and Domain Specific

Modeling Language, which is used in specific domains [MT11]. To create an understandable model for a system in a specific domain, using a general purpose modeling language, is difficult due to the complexity of describing the precise meaning of domain concepts and inter-related concepts in the entire model. Consequently, general purpose modeling languages such as UML, are not well-suited to cover some of the SN designer requirements. However, Domain Specific Modeling Language (DSML) is specifically designed for a technical domain or business domain, generally comprise a small number of concepts, and they are used by a small number of specialist and expert users [VDKV00a]. These languages provide a targeted and effective solution to a restricted and specific set of modeling problems [VDKV00b].

A large number of DSMLs with very different levels of abstraction exist. Various studies, including the one documented in [KMB+96][GK03], ensure that specific languages allow specialists and experts to increase productivity and efficiency in dealing with problems over the use of General Purpose Languages. In addition, DSML allows to have specific components of a domain in the abstract syntax, concrete syntax, and semantics of the modeling languages. These components could be ready to be used during the design phase. Thereby, DSML facilitates the job of the domain experts such as the designers [VDKV00b].

The definition of a language involves various kinds of activities that are complementary to each other: (1) defining the abstract syntax of a language, and the corresponding graphical representation of this abstract syntax which is the concrete syntax; (2) defining the meaning of the language, the semantics [KRV07][HR04]. Defining the abstract syntax consists in defining the concepts used in the modeling language. Defining the concrete syntax, consists in defining the use of the concepts of the abstract syntax. Usually an abstract and a concrete syntax are developed first, and then a semantics to define the meaning of the language [KRV07].

To formalize the definition of the modeling language, the abstract syntax is defined as a metamodel, as we have explained before. And also we add constraints relative to the metamodel concepts and relationships between these concepts. The OCL (Object Constraint Language) [CT07][Mar08] is used in this case to express the constraints in declarative formulas. These OCL constraints are the relevant support to encode specific domain constraints associated with the concepts of the DSML. In our case, we use the OCL language to provide specification of our domain constraints before providing implementation conforms to these specifications, see the section 5.1.3.2. The same activities provide the definition of a DSML which is used to describe a domain system with high level of abstraction [KBJV06]. These activities reflects the metamodeling approach to define a DSML [KBJV06][CEKS01].

2.2.2 Model Heterogeneity and Quality

The use of viewpoint models in the process of building a complex software design phase, become a standard fact [GR04]. The problem is to deal with heterogeneous models and the need for integration at the model level, in order to get an integrated and coherent

model. Thereby, at the system level, it is well accepted and understood that during the development of a complex software system such as SN, an integration in and between the created models is required. The components of complex (software and physical components) systems interact together once the integration is applied. Some components are bought, some are taken over from older systems, and some are newly developed. The components (physical and logical) are configured and implemented with different languages.

A complex system could be represented by sets of heterogeneous viewpoint models and eventually different modeling languages are used [GR04]. The decomposition of the design process into heterogeneous viewpoints models is similar to the decomposition of the complex system into several modules. Thus, to have a unique and consistent model for a complex system, a viewpoint model may also be a supplement to another one by interrelate these two viewpoints. In this sense, we can have one consistent model by having a coherency between the different created viewpoints models [BBD+00][BBDS99].

2.2.3 Models Transformation

A model transformation (MT) is a set of rules that describes and controls the transformation process of one model, in the source language into a target language [Par12][vdB09], see. Figure 2.2. For example, we can give a model as input (source language) for a code generator, in order to get a C++ code (desired language) as output. In this case, the model transformation is named as model to text transformation. And it is possible to have a transformation between two models that is named model to model transformation. Atlas Transformation Language (ATL) is a model transformation tool, a part of the Eclipse Modeling project. It supports XMI Ecore models (format from the Eclipse Modeling Framework) as input format and metamodels and generates the model resulting from the processing to XMI [BJKV06].

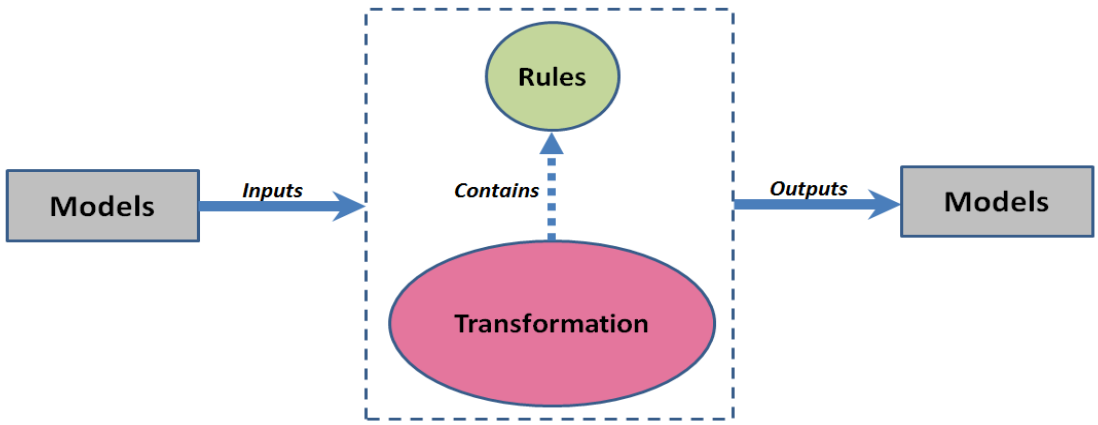

Figure 2.2: Models Transformation

2.3 Separation of Concerns in Model Driven Engineering

As we see in the previous section, the system architecture description is based on several concerns and viewpoints. One of the key feature of the MDE approaches is to provide languages and tooling which defines viewpoints and views on the system. The modeling framework must take into account this definition to provide a modeling approach based on separation of concerns.

The separation of concerns appears in the different stages of the system life cycle, and thus takes many forms. Separation of concerns may deal with the time separation of each step of the development process from design to realization. Also, for each step of the process, several viewpoints are needed to describe several concerns of the design, such as for example the logical and physical architecture of the system.

In order to accomplish the separation of concerns process, several stakeholders and viewpoints must be identified. As an example, Figure 2.3 shows the viewpoints according to the purpose and the abstraction level. The top half of this figure shows the designing, deciding and informing viewpoints which are relative to the enterprise architecture design, analysis and dissemination. The stakeholders are identified relative to the view point definition or exactly the stakeholders impose the defintiion of the view points. In our context, we previously analyzed that our main concerned stakeholder is the designer, (cf. section 1.1.3), and also several designer skills are necessary in the SN system context.

The bottom half of the figure 2.3 shows a simple view of the level of abstraction from details to an overview. This part of the figure highlights also the coherence between the levels of abstraction. This aspect is unavoidable to guarantee the consistency of the system model on all the levels of abstraction. Several modeling languages are useful and available to define the models of each view-points. We divide them into two main categories: (1) for general purposes such as UML; (2) for specific domains such as DSML. However, the use of generic languages to describe specific domains such as UML, does not ensure the application of specific constraints during the process of the design phase. Except if like UML, the language provides the capacity to be specialized or extended.

In this case, the semantics of the concepts is respected but in many cases the associated tooling remains with the look-and-feel of the host language. Due to the design complexity of SN, many specific constraints should be respected by the SN designers during the design phase, otherwise too many architectural errors may occur. To improve the architectural design quality during the design phase, these constraints could be introduced into DSML for different levels of abstraction. This improvement is performed by preventing the architectural errors that may occur using DSML. In addition, by adopting the use of DSMLs during the design phase, the created models at different levels of abstraction can be effective [VDKV00a] [CM98]. For all these purposes, the use of Domain Specific Modeling Language (DSML) for our context involves advantages to apply separation of concerns and to ensure model consistency through domain constraints.

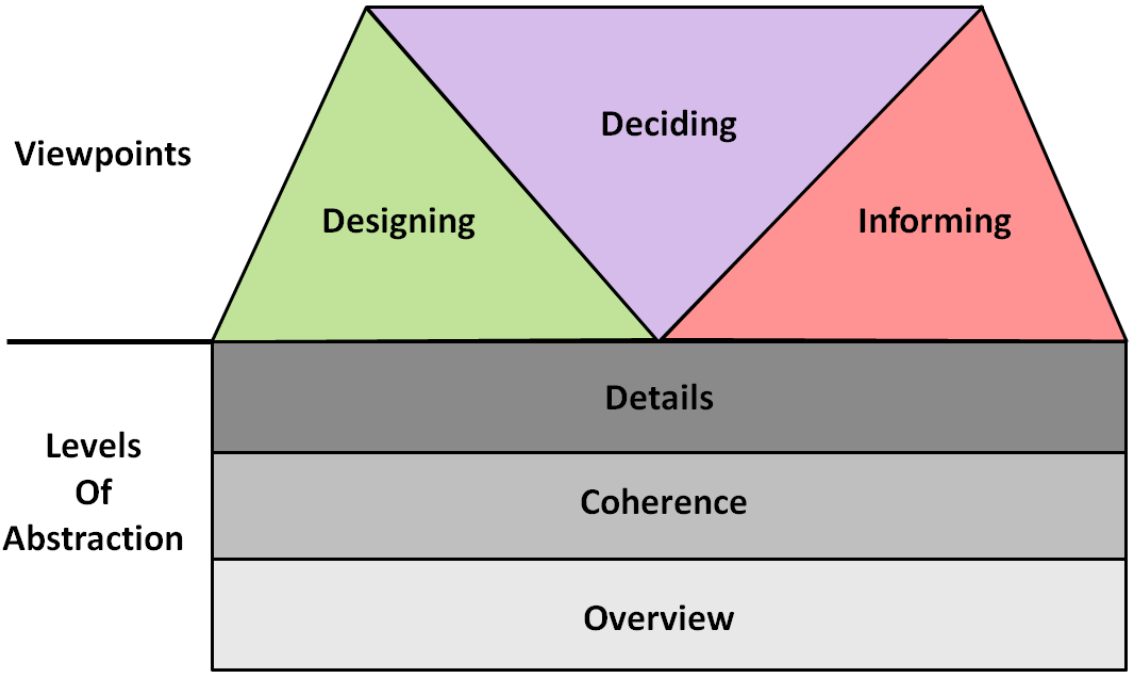

Figure 2.3: Classification of Enterprise Architecture Viewpoints, from [THE16]

2.4 Model Driven Engineering Standards and Tools

Model Driven Architecture (MDA) is an approach proposed and supported by the Object Management Group (OMG). This approach can be viewed as a specialization of the MDE [OMG03]. Its goal is to develop the software design practices based on a model-centric approach and not on code. The OMG has defined a typology of models and a set of transformation relations that allow to go from the application requirements to the implementation. In order to describe the defined models, OMG proposes to use Unified Modeling Language (UML). MDA languages are based on a set of OMG standards [OMG10], including UML (Unified Modeling Language), MOF (metaObject Facility) and CWM (Common Warehouse Metamodel).

The MOF is an extensible model driven integration framework for describing, defining and manipulating metamodels. We distinguish several tools that could be adopted for MDE with different users' concerns targets. Relying on the surveys for MDE tools that are done in [HSM10][PMDC+07], we can consider that Eclipse IDE provides a powerful environment which facilitates the modeling/metamodeling activities. The Eclipse Modeling Framework (EMF), included in the IDE, is a framework developed by the Eclipse Foundation as the foundation of the Modeling project [Ecl15]. The objectives of this framework is to enable the automatic generation of a set of tools or to provide facilities for model transformations and code generation.

2.5 Model Driven Engineering for Sensor Networks

Mainly, the approaches of SN system development focus on implementation issues, and they rarely rely on software engineering methodology which supports their entire development life cycle. However, various recent research approaches for developing SN in [RKM02][MM07][BS08][SGV+06][BT10] deal with this problem, and most of these approaches concentrate on the modeling of applications at different levels of abstraction with subsequent code generation as in MDE. MDE can contribute to the SN context by reducing the complexity of the design, by allowing designers to model their systems at different abstraction levels. Also, it provides the designers an automatic model transformations in order to convert the abstract models such as XMI files, into more concrete ones such as C++ or Java program [LVCÁ+07][Sch06].

For this purpose, [LVCÁ+07] and [Sch06] showed the use of MDE to model SN life cycle. In addition, a model driven performance engineering framework is proposed in [BS10]. Hence, to model SN, it is useful for the SN designers to use and adopt the Model Driven Engineering (MDE).

Regarding all the presented MDE features, the model transformation ability, the definition of DSML using the meta modeling approach and the model heterogeneity; we can extend existing design tool and then generate a new one that includes the new SN elements and relationships of a new defined SN DSML like in [TTH11][AYG10]. This means, it is realized, by extending the concepts of the initial metamodels (cf. section 5.1) by new components and constraints that are required by the model heterogeneity aspect. Then, generating a new design tool that contains the concrete syntax. This concrete syntax contains graphically the new added elements and relations in order to be used by the SN designers during the design time. This new generated design tool will be elaborated, in chapter 5.

By adopting MDE for developing SN, many facilities for the SN development process could be provided. As MDE shifts the focus of software development from coding to modeling, [LVCÁ+07] proposes a methodology for SN application development which is to build a model of the target system using the SN DSL. The metamodel of this high level of abstraction modeling language provides all the concepts and relationships commonly used for specifying SN applications. This is a main advantage since specific SN concepts and relationships are available to be used for the SN designer during the design phase of the SN life cycle. Another provided advantage, is a new graphical modeling editor which allows SN domain experts to graphically describe the structure and the behavior of their systems that has been developed on the basis of this SN metamodel [LVCÁ+07].

The advantages of MDE that can be provided for SN [AGF05][Chi12] are:

1. Ability to have DSML for SN, SN elements and relationships by adopting the meta modeling approach. Thus, we can introduce these elements and relationships inside the concerned framework, in order to facilitate the modeling task of the SN designer. And have design tools with concrete syntax that take into consideration the extended and customized SN DSML.

2. Diversity of SN elements and relationships within the same model from different viewpoints and DSML by adopting the model heterogeneity aspect. This advantage allows to have one consistent model that contains inter-relations between the different viewpoints according to the concerned stakeholders.

3. Having generated code as output by adopting the model transformation aspect and entering as input the created SN models. Code usable by a simulator could be generated automatically using a code generator in order to verify the created models. This advantage will be presented, in chapter 6.

In conclusion, MDE helps to facilitate the modeling task for the SN designers. This helps appear while building separate models according to each viewpoint, and also while building a consistent model from the different separately built models. In addition, MDE can provide early validation support of the created models, thanks to the static model checking and simulation code generation via model transformations.

2.6 Discussion

Relying on the presented MDE benefits, MDE is expected to contribute toward satisfying the following requirements:

1. ***Req 1 Improving Architectural Design***: in order to model a SN model, specific concepts in the IT domain are required. This domain is too wide and includes a large number of complex concepts. It contains different types of devices with different operating systems and different communication protocols, to exchange data between these devices. For this purpose, we can avoid creating such complex concepts by adopting a certain approach that allows the SN designer to reuse some existing specific IT concepts. Through the meta modeling approach for language definition (cf. section 2.2.1), existing metamodels can be extended with new SN concepts, relationships and constraints, to define the syntax and grammar of a SN specific DSML. The architectural design errors that may be made by the SN designer, are avoided by invoking the implementation of the constraints. These constraints are specified in the OCL language and associated with metamodel defintion. For example, the constraint which should be satisfied by the connection between a smart sensor and a fusion server, can be added to the abstract syntax of the SN DSML.

2. ***Req 3 Extensibility***: in order to model a SN model, the SN designer requires a specific design tool that contains the existing specific IT concepts and the new extended SN concepts, relationships, and constraints. It is a complex and difficult task to implement these new extended SN concepts in a design tool. For this purpose, we can adopt a certain approach to facilitate the implementation of new SN concepts, relationships, and constraints. Through the model transformation approach, the extensibility of the concepts appears by generating automatically a new design tool. It allows the generation of a new design tool that contains the new added components, relationships and constraints. For example, new elements or relationships with new SN constraints, are available to be used by the SN designer during the design phase.

3. ***Req 4 Heterogeneity Supported***: through the meta modeling approach for language definition, we can have different components and relationship types that are related to different contexts and activities. These extended components could be software and hardware, and they could be related together such as Voice Detection function, which is related to a Smart Sensor.

In conclusion, adding new extended concepts and constraints improve the architectural design, enforcing the respect of the constraints, and the satisfaction of certain criteria during the design phase. This prevents the SN designer from making errors that may happen. Therefore, the meta modeling approach for language definition deals with the ***Req 1 Improving Architectural Design***. Using the model transformation approach, the new specific extended concepts and relationships are generated automatically in a new design tool. It allows the SN designer to use these specific SN concepts while designing SN models. It also allows the creation of models from heterogeneous concepts and relationships, and also the creation of model from several heterogeneous models (different viewpoints). This is to ensure the ability for the SN designer to model complex systems, and satisfy the ***Req 3 Extensibility*** and the ***Req 4 Heterogeneity Supported***.

One of the key choice to apply an MDE approach, is to select the relevant Modeling Language related to our SN domain. Regarding the application domain, general purpose languages are too far from our dedicated concepts. Thus, the most relevant language could be a DSML to have the appropriate concepts, and its associated tooling. However, DSML creation and its tooling is a huge task to cover all the purposes of SN systems.

Therefore, to optimize our development life cycle, we try to find a meet in a middle approach using a powerful DSML for systems as close as possible to SN. We can apply a metamodel specialization to fit with accuracy to our SN domain. To target this constraint, the next chapter presents modeling languages and their tooling related to information systems based on a network infrastructure.

Synthesis

In this chapter, we presented aspects of Model Driven Engineering. Thus, between several aspects we identified three: Modeling Languages, Model Heterogeneity and Quality, and Model Transformations. Then, we justified our selection of adopting MDE. Next, we identified the appropriate frameworks and tools that should be used during the MDE approaches, EMF and ATL. Then, we extracted the advantages of adopting MDE by supporting the viewpoint models. Thus, according to these advantages, we identified that MDE satisfies **Req 1 Improving Architectural Design**, **Req 3 Extensibility** *and* **Req 4 Heterogeneity Supported**. *In order to optimize the* **Req 2 Multiple Viewpoints** *related to our domain, we will present in the next chapter a Domain Specific Modeling Language for information systems that relies on an MDE approach.*

3 System Architecture Modeling

Reminders and Objectives

In the previous chapter, we illustrated how MDE satisfies the requirements, **Req 1**, **Req 3** *and* **Req 4**. *In order to improve the satisfaction of the* **Req 2 Multiple Viewpoints**, *we try to identify the relevant and existing DSML. However, the* **Req 2 Multiple Viewpoints** *remain unsatisfied. In this chapter, we point out existing approaches that handle* **Req 2** *in the context of information systems. We consider Enterprise Architecture (EA) frameworks and their capacities regarding the* **Req 2**. *Hence, in relation to the needs of having practical framework for the developers of SN, we seek to find a framework that relies on EA. Next, we compare common modeling languages and EA modeling languages in order to verify how they can fit our needs. Then, we study the advantages of EA for SN in order to check if we need to adopt EA Frameworks for developing SN. At the end, we present how EA may contribute toward satisfying* **Req 2 Multiple Viewpoints** *to argue the selection of the ArchiMate framework as a baseline of our DSML.*

3.1 Modeling Context

To design a SN system, different experts are required to be involved in the modeling phase to cover the sensor integration in an IT system. Because as we defined previously our SN systems the sensors are connected to dedicated algorithms and also IT infrastructure to deliver high level services to the SN's users.

The experts have different domains of experience in order to address the different required viewpoints. For example, an expert in the business process, an expert in the logical process and another one in the technical process, are required to design a SN distributed architecture. Thus, the problematic of the modeling phase is to produce relevant models according to the required viewpoints for distributed applications on network infrastructure. Therefore, several viewpoints must be modeled to define and choose mapping of software application on a given network architecture.

To build viewpoints, designers require a set of structured and domain specific concepts. These concepts are provided mainly by frameworks. In order to handle the ***Req 2 Multiple Viewpoints***, the designers need design tools that provide the ability to

define a model using different viewpoints. This capacity can be provided by Enterprise Architecture (EA) frameworks which are based on several viewpoints on domain specific concerns. Each viewpoint is defined by its own concepts and a framework aggregates the viewpoints in a modeling language. For this purpose, we are interested to present and study the existing EA frameworks, in order to select the most appropriate one.

3.2 Enterprise Architecture Types

[OLPW+08] defines EA as:"A coherent set of descriptions, covering a regulations-oriented, design-oriented, and patterns-oriented perspective on an enterprise, which provides indicators and controls that enable the informed governance of the enterprise's evolution and success". [JE14] proposes another definition of EA as: "The organizing logic for business and IT infrastructure reflecting the integration and standardization requirements of the firm's operating model" . Hence, EA model is about dividing a model into several inter-related models such as a model for business, and a model for IT. Each model is a set of related elements and describes the activities and actions of a specific domain of experience such as Business and IT.

According to [For99] and ISO 15704, there are two types of EA: (1) EA dealing with the design of a system, called System Architecture; (2) EA dealing with the organization of the development and implementation of a project, called Enterprise-Reference Projects. System Architecture describes the structure and the behavior of a system, such as the information system of an enterprise. Enterprise-Reference Projects are frameworks which target at structuring the required concepts and tasks to design and build complex systems such SN. According to survey of EA in [CDV08], Enterprise-Reference Projects are the most adopted and used to build such systems. For this purpose, we present some of Enterprise Architecture Frameworks, in the next section.

3.3 Enterprise Architecture Frameworks

An Enterprise Architecture (EA) Framework is a set of models, principles, and methods that are used for the implementation and use of EA [CM13]. The framework is built to support the communication between the different stakeholders and different domains of experience, within the same model, by providing specific relations [CM13]. In addition, this framework allows describing a wide range of domains, it fits the problematic of our SN modeling phase by [Chi12]: (1) producing relevant models according to the different required domains of experience which is divided into different viewpoints; (2) providing the ability to relate these models by specific relationships. Five major EA Frameworks can provide the features listed above, which are: the Zachman Framework, the Open Group Architecture Framework (TOGAF), Federal Enterprise Architecture Framework (FEAF) and the Department of Defense Architecture Framework, and Gartner Framework [CM13].

Therefore, relying on previous studies in [Chi12][CM13][Sha06], the most useful framework is TOGAF to: (1) build a model from different viewpoints; (2) interrelate the business viewpoint to the technical one; (3) detail the technical viewpoint as it is required to build complex systems such as SN.

The Open Group Architecture Framework (TOGAF) defines, as part of its core, a detailed method called Architecture Development Method (ADM) to design an enterprise architecture [TOG]. ADM defines a full life cycle process, and consists of the following phases: planing, analysis, design, development, testing, deployment, and a preliminary one. These phases should be performed in sequential order [BBL12], see. Figure 3.1. In addition, TOGAF follows the strategy of simplifying the attachment of the last phase (H) of an ADM cycle to the first phase (A) of the next cycle. This attachment reflects that, ADM becomes an iterative and repeatable process [BBL12][Gro09].

This iteration feature satisfies the first requirement of the SN Design, ***Req 2 Improving Architectural Design*** for detecting and minimizing architectural design errors during the design phase (cf. section 1.3).

Figure 3.1: Architecture Development Method (ADM), from [Gro09]

3.4 Domain Specific Concepts in Enterprise Architecture Frameworks

In order to define models, the designers require modeling languages while their use of frameworks. We distinguish some existent modeling languages such as UML and SysML that are efficient to design general purpose applications using several viewpoints. General purposes modeling languages contain concepts and relationships for general purposes [MT11]. Thus, in case we need to define a model that is specific for a domain, specific concepts and relationships are required to be introduced in the adopted modeling language.

According to our context, we define models for a specific domain which is the SN. Therefore, in order to define such models, the designers require dedicated SN modeling languages to be used while their use of the adopted frameworks such as TOGAF. This due to the difficulties of having specific SN concepts in the general purpose modeling languages. These concepts are the initial phase of building Domain Specific Modeling Language (DSML) [KT08]. Furthermore, relying on [Chi12], to satisfy the requirements of the modeling task in a specific domain, the adopted framework recommends the use of DSML.

Accordingly, and relying on our SN context, a SN DSML should be defined. This definition should take into consideration the needs of SN designer. These needs are concepts, relationships and constraints in the IT and SN domains such as servers, clients and communication protocols to connect servers to clients, and exchange data between them. Also, the designer needs to define a SN model from different viewpoints according to different layers. Thus, to build such complex concepts and models is a hard and difficult task to be performed. For this purpose, we can extend existing metamodels that may contain the required complex concepts, and may allow to define a model from different viewpoints according to different domains of experience. This extension is useful as it enables the SN designer to reuse such concepts and features easily in the modeling task and avoid their complex creation from scratch.

Hence, a main question can be asked here: What are the existing metamodels that can be extended in order to define a SN DSML that satisfies the needs presented above? the ideal answer on this question is to find an existing metamodel that contains IT, SN concepts and enables the designer to build a SN model from different viewpoints.

We distinguish several SN metamodels such as SensorML [CCAN14], ThingML [FMSB11], Deep Sea Observatory metamodel [ZCKA09], Heterogeneous Sensor Web Node MetaModel [CWXG14], Wearable Markup Language (WML) [FDFL+14], SUM MetaModel [Bur14] and GINPEX Sensor MetaModel [Hau14]. These SN metamodels are already defined in previous researches and experiences. None of them contain structural, behavioral and informational SN concepts. Also, they do not contain predefined IT concepts, and they are not useful to define a model from different viewpoints. However, the EA metamodels such as TOGAF 9 and ArchiMate rely on EA, so they fit the gap presented of previous researches. Therefore, we elaborate TOGAF 9 and ArchiMate, in the next sections.

3.5 Enterprise Architecture Modeling Languages and MetaModels

EA metamodels are the abstract syntax of EA modeling languages. Thus, to model a complex system from several viewpoints by adopting EA Framework, the use of EA Modeling Language is required [CKRS14][JLVB+04]. EA Modeling Language is a conceptual or logical representation of EA with high abstraction level [Kim07].

ArchiMate and TOGAF 9 are EA Modeling Languages relying on the concepts defined by EA Frameworks such as TOGAF [Nor03][QEJVS09][Gro09][CKR12]. These EA Modeling Languages are defined by EA metamodels which precise definitions of the concepts, relationships and constraints needed for creating models. We present and discuss these two EA metamodels in the next sub sections: ArchiMate and TOGAF 9.

3.5.1 ArchiMate

ArchiMate decomposes the system design into three layers: Business, Application, and Technology. It ensures an interoperability between these layers [Nor03][QEJVS09][Gro09][Chi12]. We present these layers below:

1. Business layer: describes the actions, functions and the exchange between them that are specified by the end-user.
2. Application layer: describes the way of performing actions, functions that are defined in the business layer.
3. Technology layer: specifies the hardware components and communication protocols that are required to perform the defined actions and functions in the application layer.

Each layer of ArchiMate is defined by a metamodel. A metamodel defines by itself a language for describing a Specific Domain of interest [PMDC+07]. Three metamodels are defined according to each layer (Business, Application and Technology layers), meanwhile they are interrelated by specific relationships [Gro09]. The ArchiMate Business Layer metamodel is presented in figure 3.2. The ArchiMate Application Layer metamodel is presented in figure 3.3. The ArchiMate Technology Layer metamodel is presented in figure 3.4. Also, ArchiMate metamodels defines relationships in order to be used in the three layers listed above. This metamodel is the abstract syntax of ArchiMate modeling language.

ArchiMate has a concrete syntax, is an interface between the concepts and users [Fon07]. It may be textual or graphical, but is often a mix of both. The graphical interface of the ArchiMate concepts in business layer and relations (cf. the abstract syntax, figure 3.2) is shown in the following figures: figure 3.5, figure 3.6. For example, the Business Object (cf. figure 3.2) is displayed as an entity or class shape (cf. fourth image in the first line of figure 3.5). Another example is the graphical displays of the relations (e.g. association relationship is the third image in the first line of figure 3.6). Each concept of ArchiMate's concrete syntax metamodel has a meaning. This meaning is provided by a

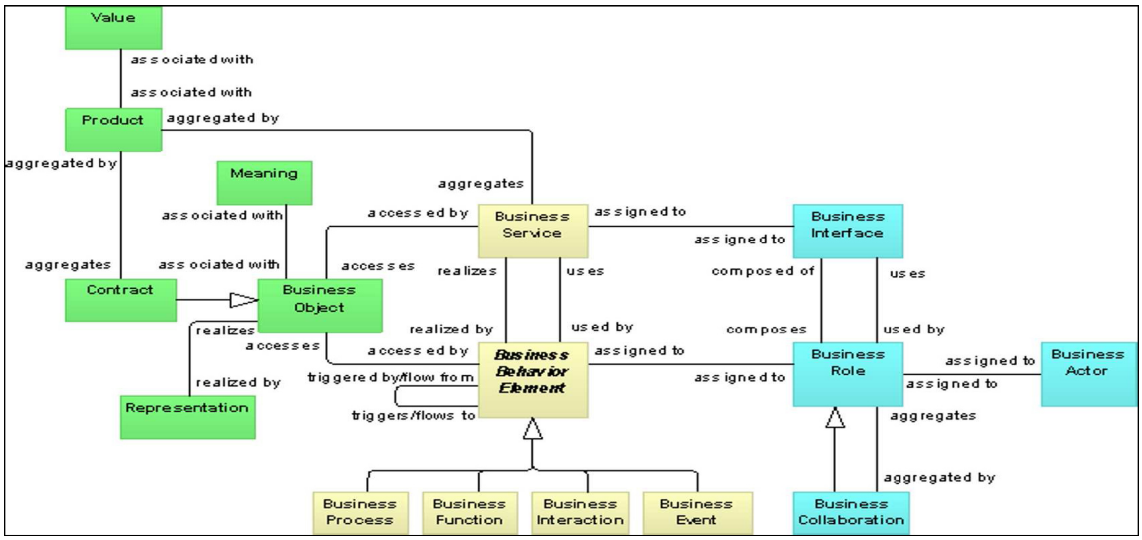

Figure 3.2: The ArchiMate Business Layer Meta Model, from [Gro09]

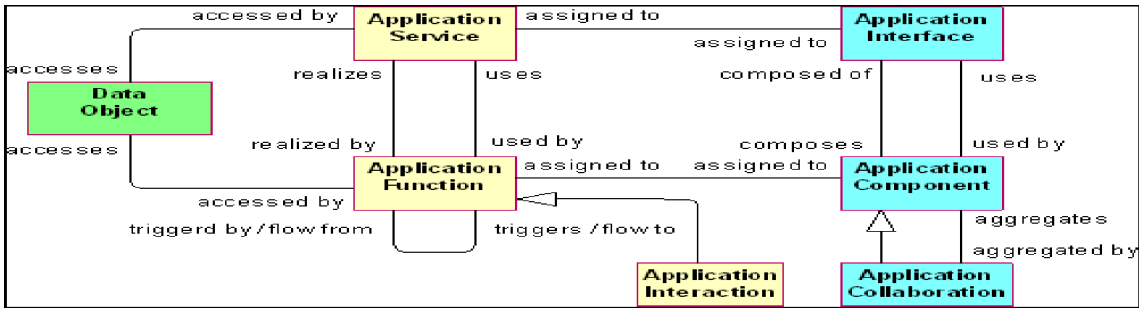

Figure 3.3: The ArchiMate Application Layer Meta Model, from [Gro09]

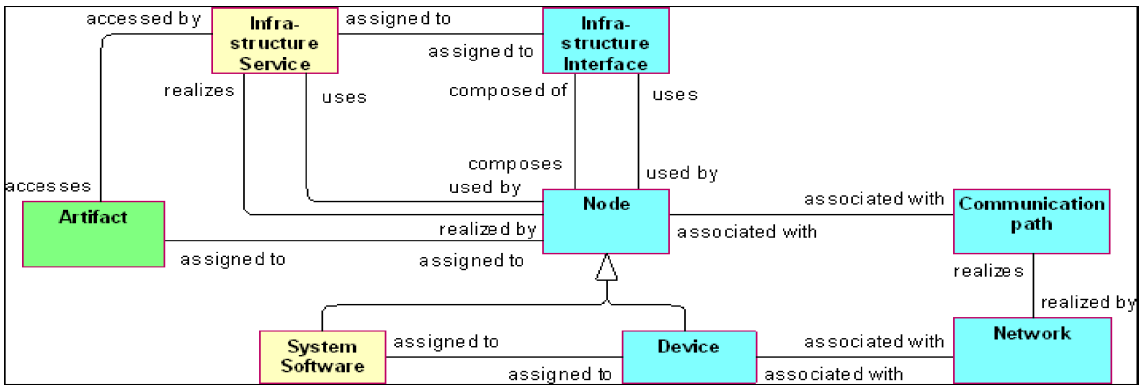

Figure 3.4: The ArchiMate Technology Layer Meta Model, from [Gro09]

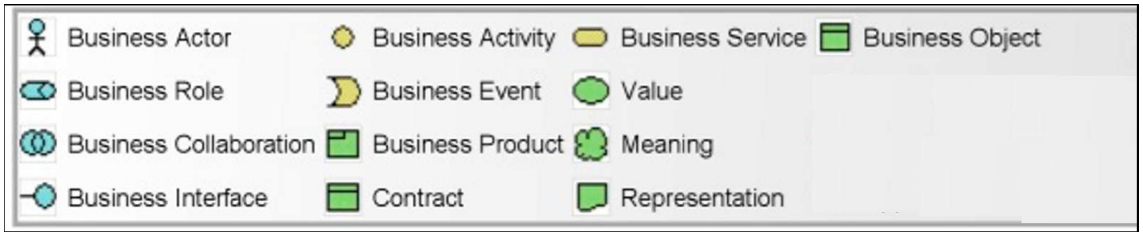

Figure 3.5: The Business ArchiMate Concrete Syntax, from [Gro09]

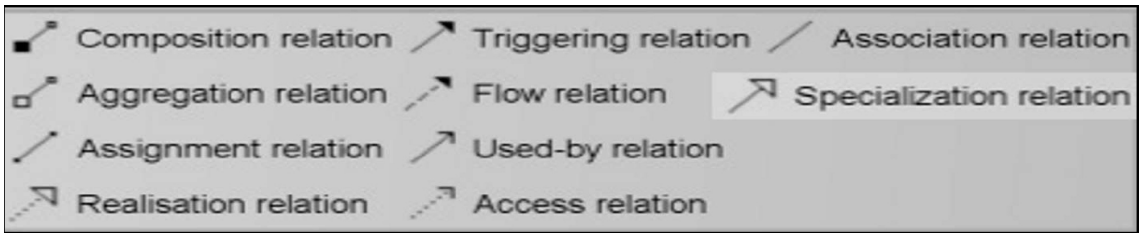

Figure 3.6: The ArchiMate Relationships Concrete Syntax, from [Gro09]

short description that is called semantics [Fon07][CGS12].

In addition, ArchiMate provides interoperability between the different views [Gro09][Chi12]. ArchiMate determines the functioning relationships between concepts of two contiguous layers [Nor03][QEJVS09][Gro09]. The relationships between business layer and application layer are shown in figure 5.7. The relationships between application layer and technology layer are shown in figure 3.8. According to our

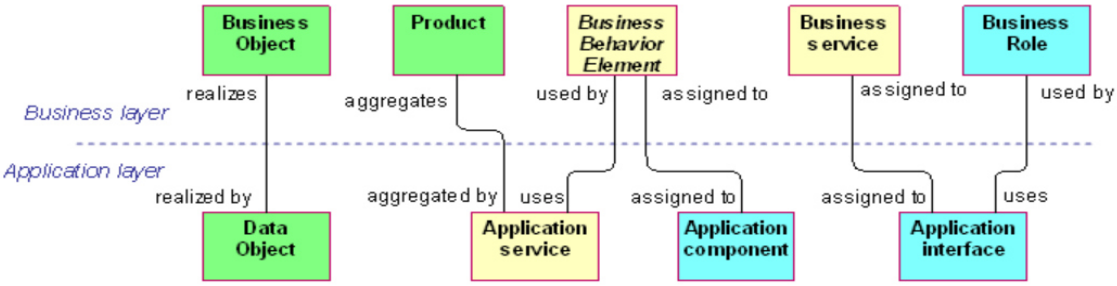

Figure 3.7: The ArchiMate Business-Application alignment, from [Gro09]

modeling context presented previously (cf. section 3.1), the relevant models according to the required viewpoints can be provided by relying on ArchiMate metamodel which is divided into three layers: business, application, and technology. The interoperability between ArchiMate layers presented previously in this section, is dedicated to manage the relation between the different layers of ArchiMate such as Used by and Realization relationships. These latter can allow the communication between the different created models by exchanging information between them (see. Figures 5.7 and 3.8).

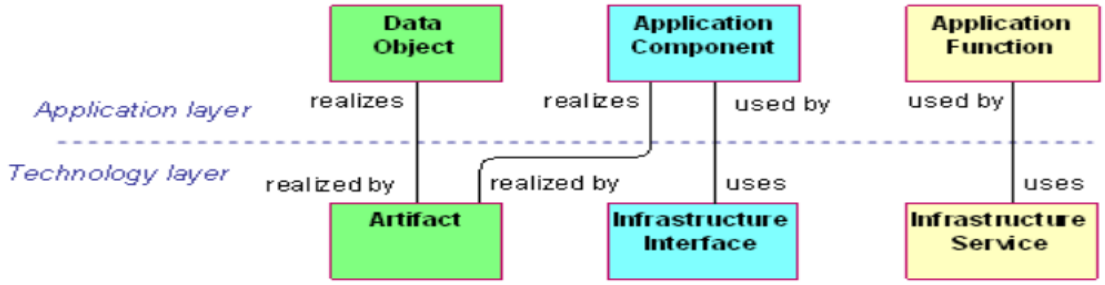

Figure 3.8: The ArchiMate Application-Technology alignment, from [Gro09]

3.5.2 TOGAF 9

TOGAF 9 decomposes the system design into three layers, see. Figure 3.9: Business Architecture, Information Systems Architecture (Information and Data), and Technology Architecture. It ensures an interoperability between these layers [Gro09]. The layers of TOGAF 9 are defined by a metamodel, where the latter defines by itself, a language for describing a Specific Domain of interest [PMDC+07]. And the metamodel is defined according to the three layers, meanwhile they are interrelated by specific relationships [Gro09]. This metamodel and its layers are presented in Figure 3.9.

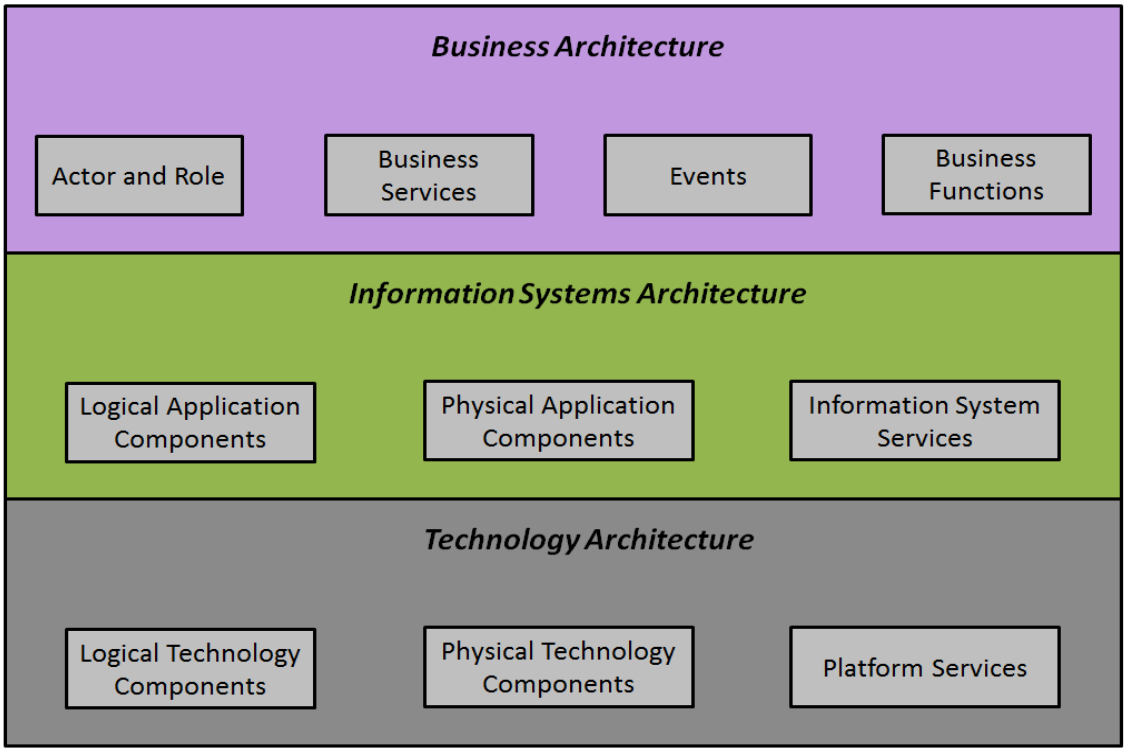

Figure 3.9: Layers of TOGAF 9 MetaModel, after [Gro09]

3.6 Requirements for Selecting the Enterprise Architecture MetaModel

As we discussed previously, in order to define a SN DSML, the ArchiMate or TOGAF 9 metamodel must be extended. For this purpose, we are interested in identifying the requirements in order to select the appropriate metamodel:

1. ***Req 1 Several Viewpoints:*** Providing each designer the ability to work independently in a viewpoint, in order to have his proper model according to his domain of experience. This is to address the different viewpoints independently. For example, a network designer and a software designer, each one designs in his independent viewpoint where he is expert.

2. ***Req 2 Separate Logical and Physical View:*** Providing the software designer the ability to define the logical models in a separate view that is specific to describe only logical components. And, providing the network designer the ability to define the physical models in another separate view that is specific to describe only physical components. This is required since we are not interested to mix logical and physical components and share several designers with different domains of experience within the same viewpoint, so we need a separation between components.

3. ***Req 3 Consistency Supported:*** Providing each created model the ability to interoperate with other created models, in order to share the design of the same model. This is, by relying on the interoperability between ArchiMate layers that is provided by ArchiMate, the different created models can be inter-related together in order to provide one consistent model from different viewpoints. For example, a function in a model calls another function in another model.

4. ***Req 4 Specific IT Components:*** Providing the network designer, facilities in the modeling phase, by using built-in IT components easily through the generated design tool. For example, the designer can use devices such as clients or servers, components, and relationships to connect devices together using specific relationships such as communication path.

3.7 Comparison Among Enterprise Architecture MetaModels

We discuss a comparison between two EA metamodels, ArchiMate and TOGAF 9, where one of them satisfies all the identified requirements, see. Figure 3.10:

1. ***Req 1:*** Several viewpoints are available in both metamodels TOGAF 9 and ArchiMate. They provide the ability to have several separated models according to different domains of experience.

2. ***Req 2:*** ArchiMate deals with the separation of logical and physical view, and it never mixes them together within the same viewpoint. This due to the three separated layers that are provided by ArchiMate: business, application and technology. The application layer contains only the logical components, and the technology one

Requirements / MetaModels	Several Viewpoints	Separate Logical and Physical View	Consistency Supported	Specific IT Components
TOGAF 9	✓	X	✓	✓
ArchiMate	✓	✓	✓	✓

Figure 3.10: Comparison among ArchiMate and TOGAF 9 MetaModels

contains only the physical components. However, the application architecture in TOGAG 9 contains physical application components and the technology architecture in TOGAG 9 contains logical technology components.

3. ***Req 3:*** ArchiMate and TOGAF 9 allow the inter-operation between different models. They provide the ability of having one consistent model that contains components and relationships from different viewpoints at different layers.

4. ***Req 4:*** Specific IT Components are available in both metamodels TOGAF 9 and ArchiMate. For example, ArchiMate has a node and a device, while TOGAF 9 has physical technology component.

Relying on Figure 3.10, we notice that the TOGAF metamodel 9 and its layers are similar to the ones of ArchiMate with some little differences. Thus, we ensure that ArchiMate shares with TOGAF ADM interesting concepts, and both of them are compatible, so they can be used together [Lan09] (see. Figure 3.11). Furthermore, ArchiMate provides for the designers a concrete syntax, which is an interface between the concepts and these designers [Fon07]. For this purpose, we can select ArchiMate to be the metamodel that could be extended by new SN concepts, relationships and constraints.

Accordingly, in order to build SN, the designers can adopt TOGAF as a framework and ArchiMate as a modeling language which relies on EA. However, we distinguish several frameworks and design tools that are dedicated for building SN. Therefore, in order to take the final decision concerning the framework and the modeling language that should be used, we must discuss the existing SN frameworks and design tools. For this purpose, these latter are elaborated, in the next section.

3.8 Enterprise Architecture Frameworks and Design Tools for Sensor Networks

A framework is a set of functions and libraries to model applications from different domains, like in [oS07]. Several frameworks provide design tools [oS07]. Design tools enable the designers to create analysis and design models of the system to be built, and

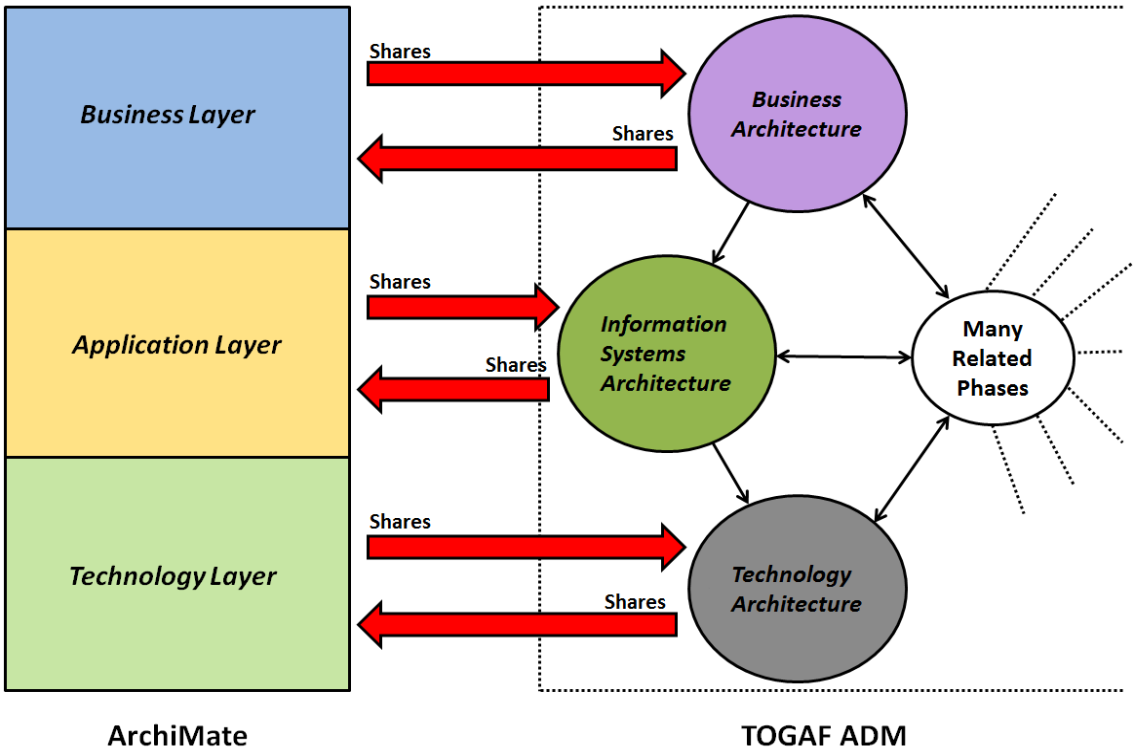

Figure 3.11: Compatibility between TOGAF ADM and ArchiMate, after [Gro09]

ensure consistency between models [KC90]. We differentiate design tools by their provided features. For example, [TTH11] offers the following features: modeling application within specific domains, preventing architectural errors by invoking the grammar of the modeling language during the design time, verifying the created models, including concepts diversity, and allowing to add new specific concepts. While, [AYG10] provides the features of preventing errors, components diversity and addition of new specific concepts. However, the features of preventing errors and components diversity are offered in [BJ02]. More features and options are offered in [oS07], such as including graphical user interfaces, and a built-in animated simulators, see. Figure 3.12 that shows the verifications of the created models in a graphical way. In some cases like in [AKR13][AAK+14][AAKR14][CZ15], the simulator of the design tool is external and it could be able to verify the created models by adopting model transformations. Thus, the choice of the design framework is a difficult task.

Some studies and recent researches have therefore focused on SN frameworks and motivated to adopt them. These frameworks provide graphical common interfaces for heterogeneous sensors and actuators, and ease their deployment and management [GK11]. Examples of these frameworks are: Global Sensor Networks (GSN) [AHS06], Sensor Web Enablement (SWE) [FBK+11], SENSEI (recent European research project) [Luo13]. These frameworks are not useful for our context as in order to build SN, the designer requires to adopt a framework and design tool that include a DSML that contains IT, SN concepts and enables the designer to build a SN model from different viewpoints (cf.

section 3.3).

Many frameworks are proposed in order to support the management of enterprise IT by describing the systems from IT domain, using an EA modeling language [MHO11][Fra13] such as the Enterprise Architecture Frameworks, TOGAF. These IT frameworks address wide range of domains and technologies as they allow different stakeholders to describe a system according to their different domains of experience. Thus, each stakeholder creates his proper model according to his viewpoint. These different models could face the difficulties of developing SN due to the diversity of technologies that are required [GK11][YL13]. For example, connect set of different components (hardware and software) within a network.

Therefore, TOGAF and ArchiMate can be adopted by the designers to build SN. For this purpose, the designer can adopt the framework DeVerTeS developed in [All16].

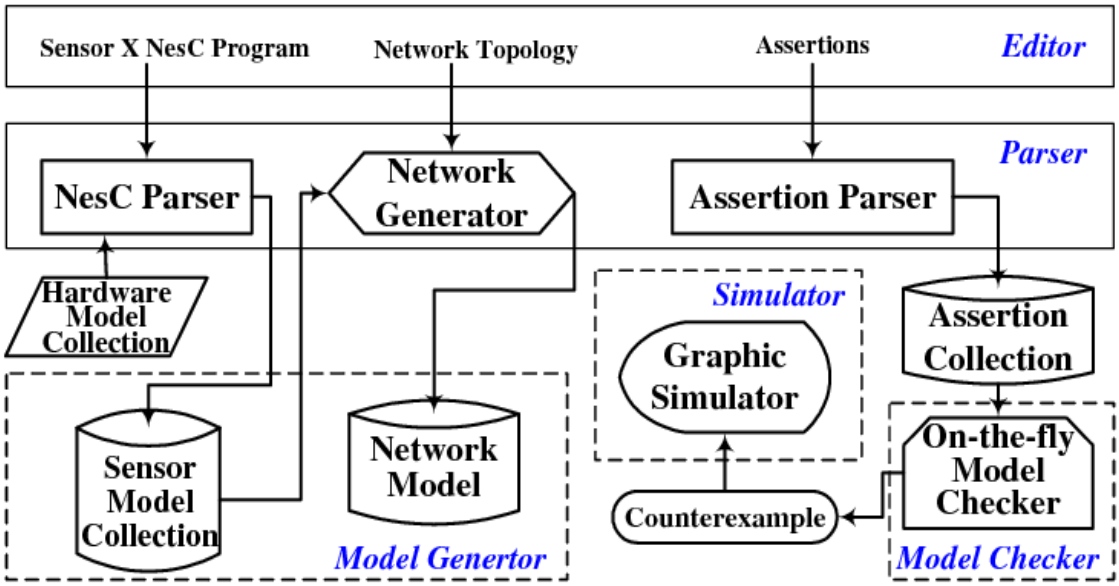

Figure 3.12: Architecture of NesC@PAT, from [oS07][ZSL+11]

3.9 DeVerTeS: A Design and Verification Framework for Telecommunication Services

One of the framework that can be used in our SN case is DeVerTeS, it relies on EA and ArchiMate. DeVerTeS is a framework that reuses existing approaches which help us to satisfy the requirements of the design environments [All16]. It relies on: (1) the modeling language that is extended ArchiMate, represented by its metamodel through all the activities of the framework; (2) the integration between the different tools, to perform the design and verification activities.

This framework added a verification activity, in order to detect and identify the

architectural design errors before the simulation of the system during the design phase, taking into consideration the different viewpoints. An advantage appears in this framework, which is not affecting the standards such as ArchiMate while adding activities and features on DeVerTeS, so it keeps the advantages and benefits of existing and extended standards such as extending the design activity by a verification one. Therefore, DeVerTeS can be adopted to be used in different domains according to its mentioned features above.

3.10 Discussion

Regarding the previous presented EA features for SN, the advantages of Archimate, and the needs of introducing specific SN concepts into the EA Framework, extending ArchiMate could be a suitable contribution in order to define Domain Specific Modeling Language (DSML) for SN. This DSML should include new specific SN concepts and constraints that are inherited from the initial ArchiMate metamodel concepts, like in [TTH11][AYG10]. Consequently, this contribution could target the ***RQ 2 Modeling Language Specialization***, and it will be elaborated on in details, in chapter 5.

EA provides for the SN, the following benefits:

1. Achieving the right balance between IT effectiveness and activities on high abstract level. It allows SN designers to create safely their models such as any data fusion architecture or network topology as it ensures the needs of the created models for an integrated IT strategy. SN designers are able to model any complex system without worrying about the availability of IT components.

2. Reducing the deployment uncertainty of SN model since it is well-defined. This is due to the existence of different layers and stakeholders, where each stakeholder is an expert in his domain so he constructs his model in his layer. And due to the interoperability between the different layers that inter-relate two or more different models in separate ArchiMate layers in order to have one consistent model. Thus, this model has minimum errors as it is well analyzed and detailed by different domain experts at different layers.

Relying on the identified EA advantages, EA is expected to contribute toward satisfying the ***Req 2 Multiple Viewpoints***. EA provides the ability for the SN designers to create several models according to their viewpoints and domain of experience (Business, Application and Technology). Also, it provides the ability to inter-relate the different models in order to have one overall and consistent model, that could be understood by all the concerned designers.

In conclusion, of this part of the state of the art, we investigated the use of modeling engineering approach for sensor networks, to address ***Req 1 Improving Architectural Design***, ***Req 3 Extensibility***, and ***Req 4 Heterogeneity Supported*** based on the ***Req 2 Multiple Viewpoints*** modeling approach with an Entreprise Architecture Framework. In order to improve the life cycle development of sensor networks, we state that we want to introduce an early validation based on a simulation support. In the context of complex systems, simulation is the most powerful approach to include early validation. This life

cycle phase must be supported by ***Req 5 Validation Tools***, so the modeling tooling must be connected automatically with the network infrastructure simulator.

Synthesis

In this chapter, we presented EA and its benefits. Thus, most of the existing and used types of EA are frameworks. We presented some of EA frameworks, and we pointed out that TOGAF is interesting since it follows a complete development method with having as advantage its iteration feature. Relying on the compatibility between TOGAF and ArchiMate by having common concepts, we adopted ArchiMate as modeling language. Next, we introduced the domain specificity in EA framework by defining the need for a domain specific modeling language (DSML). Specific concepts are needed to be used in the modeling language, since the general existing concepts does not satisfy the requirements of the sensor networks domain. At the end, we identified the advantages of adopting EA framework to model SN. And according to these advantages, we identified that EA answers **Req 2 Multiple Viewpoints**.

Nevertheless, EA are not dedicated to sensor networks. And the modeling tooling for EA is not easily connected with network simulators. So, to go beyond this state of the art, we suggest to improve the current approaches to satisfy our requirements, by extending the language of the EA modeling tooling and providing early validation, supported by introducing a model transformation towards the simulator tooling like NS-3.

Accordingly, we may adopt and use some of previous works that can be complemented by our contribution as [CKR12] and [All16]. [CKR12] extended an EA modeling language to define a DSML for telecommunication domain in order to enrich in the design activity. Thus, by relying on [CKR12] and EA and standards, [All16] added a verification design activity in order to built DeVerTeS. Therefore, we can select DeVerTeS to be the framework that must be adopted in order to apply our contribution. For this purpose, we use DeVerTeS then develop it for the Marine Observatory domain in the same manner.

Moreover, the next chapters which describe our contribution, detail the adopted life cycle based on the specialization and the improvement of EA language and tooling.

PART II : CONTRIBUTIONS

We propose, in Chapter 4, a design process for sensor networks, inspired from the several software development processes. To help designing the sensor networks, we propose a Domain Specific Modeling Language which is dedicated to the Sensor Networks designers, in Chapter 5. The tooling associated to this DSML definition is based on MDE technologies and is presented in Chapter 5. An application of the proposed Sensor Networks design process and tools on a complete case study, is provided in Chapter 6.

TELECOM
Bretagne

Our Contribution

Our contribution is based on a standard approach and tooling relative to an Entreprise Architecture Framework. Also our work complements the previous works: (1) languages and modeling from [Chi12]; (2) DeVerTeS, a framework developed by [All16].
[Chi12] extended ArchiMate modeling language to enrich the design phase of a system for a specific domain, the telecommunication infrastructure.
[All16] extended the design activity to introduce the early verification activity in the framework (DeVerTeS) by developing a model compiler to perform a simulation based on the models, (cf. section 3.9).
Our contribution is based on a standard approach and tooling relative to an Entreprise Architecture Framework. Also our work complements the previous works: (1) languages and modeling from [Chi12]; (2) DeVerTeS, a framework developed by [All16].
[Chi12] extended ArchiMate modeling language to enrich the design phase of a system for a specific domain, the telecommunication infrastructure.
[All16] extended the design activity to introduce the early verification activity in the framework (DeVerTeS) by developing a model compiler to perform a simulation based on the models, (cf. section 3.9).
As we are involved in Sensor Networks and specificaly in Marine Observatory domain, we extend the previous works that are integrated in ArchiMate modeling language by adding new required specific MO concepts, relationships and constraints. This work enriches the use of ArchiMate in the design phase for SN domain.
Also, we reuse DeVerTeS to benefit from this extension to simulate the created MO models. So the main purpose of our work is to improve the development process of the SN system by providing an early validation phase based on the use of models and simulation results. In order to perform this improvement of the SN development process, we propose the following: (1) defining a SN design process; (2) defining a SN DSML; (3) defining a SN DSML tooling; (4) providing simulation and analysis of SN defined models.

4 Sensor Networks Design Process

Reminders and Objectives

In this chapter, we present and discuss several software development processes to inspire the common features from them. Next, we select the common tasks or approaches that can be a part of the proposed SN design process. Then, we propose a SN design process that is composed of three tasks: modeling the SN from the stakeholders' point of view **Task 1 Modeling**, *ensuring consistency of models* **Task 2 Ensuring Consistency**, *and validating the created models* **Task 3 Validating**. *Next, we elaborate how to perform these tasks while using the proposed tooling, in an MDE context. Then, we elaborate the content of the proposed tasks related to our SN domain, in order to check if the proposed process fit with the requirements of SN designer. Finally, we present our supports toward the three proposed tasks.*

4.1 Context

In order to improve the design phase of the SN life cycle, we presented and discussed several approaches, tools and modeling languages, in the previous chapters. These chapters show how we selected the DeVerTeS framework [All16], based on TOGAF, to be the adopted enterprise architecture (EA) framework. TOGAF contains several related phases that are used for different domains and large scope. According to our SN context, we only use from TOGAF the three following phases [Gro09]: Information Systems Architectures, Technology Architecture, Opportunities and Solutions. Thus, the scope of our contribution is reduced by adopting only three of nine phases from TOGAF. Also, we selected Model Driven Engineering (MDE) approach and ArchiMate EA modeling language that are relevant to use for SN design aspect in the three selected phases of TOGAF.

The MDE context provides the capacity of proposing an efficient tooling to model and validate the SN system architecture. However, the purpose is to answer on the following question: How to use this tooling before building it. In order to answer this question, a set of sequential tasks must be performed. These tasks form a process to design a SN. For this purpose, we must select and/or propose a process to improve the SN design phase. In order to fulfill that, we realize and present the following points:

1. First we present briefly and discuss several software development processes (cf. section 4.2).

2. We select the common tasks or approaches that can be a part of the proposed SN design process (cf. section 4.3).

3. We elaborate how to perform these tasks or the entire proposed process while using the proposed tooling, in an MDE context (cf. section 4.4).

4. We elaborate the content of the proposed tasks related to our SN domain (cf. section 4.5).

Therefore, we define a process to provide guidelines to use our proposed tooling. Thus, this process answers the following question: How to use and improve modeling and simulation to fit early validation goal for SN. This question reflects the following specific question: How to model SN with ArchiMate. In order to reply to this question, we must specialize ArchiMate for SN. And in order to fulfill that, we can extend ArchiMate by adding new SN concepts to build a metamodel for SN. Then, a customized tooling is ready to be used for creating specific SN models. This generates the following question: How to use this customized tooling to model SN. In order to answer this question, we elaborate the procedure of using the three layers of ArchiMate while creating SN models.

Next, we must answer the following question: How to validate the SN models. The simulation of a dedicated code is the answer to this question. This generates the following question: how to improve the simulation phase. By generating a specific code to be used while the simulation phase, is the answer to this question. This accelerates the simulation and the development phases as the code is generated automatically without manual intervention from the programmers. Also, this ensures that the code to be simulated is coherent and consistent with the defined SN models.

4.2 Software Development Processes

We briefly present some development process to provide context of the task identification of our system development process. The software development process is a set of phases [Jac94]. These phases are steps for specifying, designing, developing, testing, and maintaining complex software such as Sensor Networks [AB15]. We distinguish several software development processes, such as: V-shaped, Spiral, and Agile [RP08][AB15][Ras][Som04].

1. V-shaped: it is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Before development is started, a system test plan is created. The test plan focuses on meeting the functionality specied in the requirements gathering. The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase as well, in order to test the pieces of the software system's ability to work together. The low-level design phase is where the actual software components are designed, and unit tests are created in this phase as well. The implementation phase is where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V, where the previously developed test plans, are now put to use [Lew05].

2. Spiral: the main feature of this process is dividing a project into smaller modules. In this process, the development team starts with a small set of requirements

and then cover these requirements (except installation and maintenance phases) [Hur14][TG11] [MG10]. The team develops a small module according to the small set of requirements. Then, the concatenation of the different small modules provides the final running system.

3. Agile Methods: these methods satisfy the customer needs by delivering systematically new releases of software. And they adopt the collaboration between the different stakeholders such as client, designers and testers in order to produce a consistent product. These stakeholders fix the software product on each phase of the development process. Once the current requirements are covered, a new version of the software can be built and delivered to the customer. We can iterate this action while the stakeholders find errors. Furthermore, these methods can be adopted to build a complex system with high number of heterogeneous hardware and software components [KG12][Som04].

This short presentation aims to highlight that iterative development processes are suitable to build a software system incrementally with a small set of requirements for each iteration. Also, the Agile processes emphasis on collaboration between stakeholders to build a software product for each iteration. So we want use this identified features to improve the SN development process in our context.

4.3 Selected and Proposed tasks of the Sensor Networks Design Process

4.3.1 Concept and Challenges of Sensor Networks Design Phase

As the SN are not only a simple SN, they are enterprises as they deploy and use high number of interconnected heterogeneous components (software and hardware) (cf. section 1.2), a system development process is required to be performed to implement such SN.

The software development process of a complex system should focus on the separation of concerns and their descriptions [Jac94]. These descriptions are the analysis of a system requirements within a specific context. As we presented previously in this document, we are interested in the design phase (cf. section 1.1.2). Thereby, in order to define this phase, three main questions must be answered: (1) what should be described; (2) how to model the relevant system architecture; (3) how to validate the defined models. Thus, in relation with designing SN, the questions could mean the following: (1) what are the selected concepts to be used and described relative to SN domain; (2) what is the process to describe these concepts and their interactions; (3) how do we detect architectural design errors in the produced consistent model. The answer on the first question is: the extended MO metamodel in section 5.1.3. And, for the second question, the answer is: these actors and functions are related together by using specific and general purpose relationships while respecting the constraints of the SN domains. And, for the third question, the answer is: the simulation of the generated code using a network simulator (cf. section 6.4).

According to the answer on the first question, different types and high number of components (software and hardware) are the selected concepts to be used while defining

SN. Therefore, it is difficult to define a model that requires the intervention of different domain of experiences (experts), such as domain expert, software designer and network designer. Regarding the answer on the second question, we describe an heterogeneity in the types of SN components that must be constrain regarding our SN context. These constraints should take into consideration: (1) the large number of specific SN components; (2) interactions and integrations between the software/hardware components and the core network. So, the SN designers take a lot of time to perform their tasks according their different viewpoints. According to the answer on the third question, an automatic generated code is ready to be simulated without manual intervention. So, it is a complex task to be performed.

4.3.2 Requirements for Selecting or Proposing Tasks of the Sensor Networks Design Process

Accordingly, we identify the next requirements:

- ***Requirement 1 Viewpoints Consistency:*** Providing each designer the capacity to perform his tasks in an independent viewpoint according to his domain of experience. This motivates the need of designing a SN on different abstraction levels using different layers according to different viewpoints. Thus, the designers can define one consistent SN model that contains different viewpoints.

- ***Requirement 2 Efficiency:*** Providing each designer the ability to perform his tasks rapidly in the design phase. For example, he can define models or fix rapidly the detected architectural design errors on the design phase. This affects positively the time consuming to build a new release of the SN model upon defining a new model or detecting such errors in a defined model.

- ***Requirement 3 Output Validity:*** Providing each designer the ability to validate structural architecture constraints and algorithm deployment on network infrastructure. This validation is performed on the design phase at system level in order to produce a SN model that contains no architectural design errors. Thus, the architectural SN model can be early validated.

- ***Requirement 4 Large Size:*** Providing each designer the ability to create a SN model that contains a large number of components, relationships between components and layers such as association and used by relationships, and communication paths. For instance, this helps the designer to create SN models which can be based on any type of fusion architectures such as distributed.

4.3.3 Analyzing the Relation between the Tasks of the Software Development Processes and the Identified Requirements

- ***Requirement 1 Viewpoints Consistency:*** none of these three processes deal with this requirement. The designers must intervene and perform their tasks at

any layer in the design phase. For example, they can create models on different viewpoints or enhance defined models by fixing the architectural design errors on each level of the design phase and on different layers of ArchiMate.

- ***Requirement 2 Efficiency:*** the three processes deal with this requirement. The designers can perform their tasks rapidly. For example, the designers can detect rapidly the architectural design errors in order to fix them. This rapidity reduces the time needed to perform the entire software development process.

- ***Requirement 3 Output Validity:*** the three processes deal with this requirement. All of them produce valid outputs such as a valid SN model. This due to the possibility of multiple iteration of each phase and within the same phase. For example, this iteration is possible on each error detection at any phase or any layer within the same phase.

- ***Requirement 4 Large Size:*** the three processes deal with this requirement. The designers can create a model with a large number of components, relationships and communications. For example, a SN model with a Distributed Fusion Architecture.

Consequently, the three processes fit three of the identified requirements, and they do not fit the first one. So, all the presented processes have the same shared properties and features. For example, the most central common concepts in these three processes are roles and tasks [ZYPEQ10]. Roles refer to the essential skills needed by teams in order to perform tasks such as model creation. These teams collaborate and coordinate together in the software development process in order to produce one integrated output such as SN model [SOV+11][Ale12]. Thus, tasks refer to a unit of work performed by roles. Another example about these features is that, all of these processes adopt the iteration approach in order to provide an early validation of the produced software by fixing rapidly the detected architectural design errors. However, they still have different advantages and disadvantages according to the type of software that should be realized. The main advantage of these processes is the flexibility of iterative tasks within the same process in order to build a valid created model [ZYPEQ10]. This advantage allows to repeat each task many times in the same life cycle, phase or design process that provides non adequate performances by simulation of the designed model. Therefore, the main feature of the three processes emphasize an iteration approach to improve the design. So, we inspire from these three processes the approach that provides an early validation of the defined models, which is the iteration. This iterative approach can be applied on a set of relevant tasks that can be proposed according to our SN context. For this purpose and in order to satisfy "Viewpoints Consistency" requirement, we propose several appropriate tasks to design a SN, in the next section.

4.3.4 Proposed Tasks of the Sensor Networks Design Process

We propose a process to design a SN which is divided into three tasks (see. Figure 4.1): Modeling, Ensuring Consistency and Validating. These proposed tasks should be performed through the role of SN designer. We define these tasks, in the next paragraph:

- ***Task 1 Modeling***: modeling the SN from several stakeholders point of view. Each designer describes the SN from his point of view, according to his concern and domain of experience. This due to the SN that require different design experts for three ArchiMate layers: business, application and technology (cf. the "Modeling" task in Figure 4.1).

- ***Task 2 Ensuring Consistency***: Once the models are created by different stakeholders such as design experts, an interoperability between these models is required in order to have one consistent SN model. This interoperability is performed between the SN model that is created on the business layer and the one created on the application layer. Also, it can be performed between the one created on the application layer and the other created on technology layer (cf. the "Ensuring Consistency" task in Figure 4.1).

- ***Task 3 Validating***: During the design of a complex system, the architectural model must be evaluated and occasionally updated by the SN designer in order to detect the architectural design errors. This validation is performed on the consistent model that contains concepts from the three ArchiMate layers, and inter-relationships to relate two separate layers (cf. the "Validating" task in Figure 4.1). For this purpose, an early validation of the consistent produced model is required in order to detect such errors as soon as possible. This is to avoid the errors in the later development phases.

In addition, we propose to apply on these three proposed tasks, the selected iteration approach that is inspired in the previous section. This approach is applied by re-performing sequentially the three proposed tasks of the SN design process while errors are existing in the defined model (cf. the "Iteration" approach in Figure 4.1). Thus, in order to implement these tasks, we perform them in an Model Driven Engineering (MDE) context, in the next section.

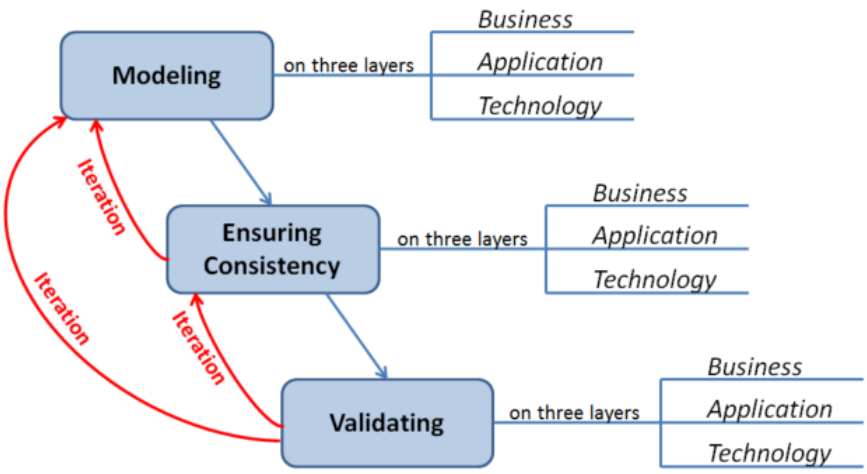

Figure 4.1: Proposed Tasks and Approaches of Sensor Networks Design Process

4.4 The Proposed Sensor Networks Design Process and Model Driven Engineering

A tool can be used in one or more processes, and each process can be used with one or more tools. This means, by having a process we can define one more tool to be used, and by having a tool we can define one or more processes to be used. Therefore, we can exploit this aspect in our case, as we propose to define a specific SN tool. So, we define a SN design process. In order to implement such process, we adopt and use MDE. So, a question must be asked here: Why can MDE be used in this context?

We use MDE due to its characteristics, it provides the ability to define new tooling by using the metamodel and model transformation (see. Figure 4.2). And it provides the ability to support a development process. More specifically, by focusing on the metammodel, MDE allows us to define specific concepts and relationships that will be used during the proposed "Modeling" task. Thanks to the metamodel, we are able to define rules of coherence that is used during the proposed "Ensuring Consistency" task. Then, by focusing on the model transformation, MDE allows us to generate code that is used during the proposed "Validating" task. This last is performed by simulating the generate code.

Consequently, MDE is used to improve the proposed SN design process by proposing a SN metamodel associated with its consistency rules, and by using model transformation to produce the simulation source code. Thus, MDE is used to support the proposed tasks of the SN design process.

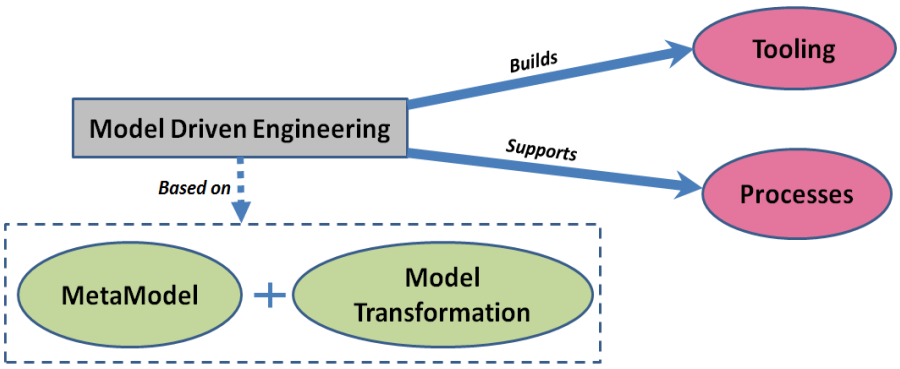

Figure 4.2: Features and Aspects of Model Driven Engineering

4.5 Content of the Proposed Tasks of the Sensor Networks Design Process

Accordingly, and in relation with our previous selection for ArchiMate in chapter three to be adopted as the modeling language while performing the design phase, the proposed tasks are performed by the SN designers using ArchiMate (cf. all the rectangles in Figure 4.3 except the gray one) such as "Define the Model from his Viewpoint" task and "Coordinate with other Designer" task.

The different designers perform the tasks of creating several separated SN models according to the different ArchiMate layers. As ArchiMate relies on EA, so they can create three models based respectively on ArchiMate's business layer, application layer and technology layer (see. Figure 4.3). Thus, to perform the modeling task by the designers, we require an expert designer for each ArchiMate layer.

Then, they must establish relationships between these different models that are created by different experts. Each relationship is specific to relate two different concepts (functions for example) that are located in two different models according to two different layers such as business and application ArchiMate layers (cf. the blue lines and arrows between the three layers in Figure 4.3).

In order to perform this inter-connection between models, the designer should be expert by selecting the appropriate relationship that allows the connection between the two selected different concepts. The goal of the use of these relationships is to have one consistent model built from several different viewpoints (cf. the gray rectangle in the technology layer in Figure 4.3).

In case they detect architectural inconsistencies in the models during this interconnection, they can iterate on the concerned models. At the end, the designers can validate the produced consistent model by a validation tooling such as a network simulator. In our case, if the validation fails so the network performances are not reached, the designers iterate and fix the detected errors wherever in the adopted design process (cf. the red lines and arrows in Figure 4.3).

In order to describe the entire SN design process, we should describe the content of the different proposed tasks according to each viewpoint and the interoperability between viewpoints, then the validation of the produced model. The designer begins by creating and describing a model. He adds information to this model according to his concerns, from his point of view. This can be performed by adopting the first proposed task, ***Task 1 Modeling***. His modeling may be influenced by constraints and knowledge coming from other viewpoints in the process. These constraints may result from coordination with designers from other viewpoints, from different background and domains of experience. This coordination may allow the designers to discover what is the criteria to relate two separate models from different viewpoints and how to relate them, and through what. For example, to relate two different models, a specific relationship should be created between two specific components, each component is in a different model such as when we relate a business function from a business layer viewpoint to an application function from an application layer viewpoint. This can be performed by adopting the second proposed task, ***Task 2 Ensuring Consistency***. Once, we create the first version of the model, the designer might invoke an automatic metamodel conformance tooling. This can be performed by adopting the third proposed task, ***Task 3 Validating***. According to the obtained results, he might change the models, then test it again. This change could be done through modifications or enhancements of any existing task in the different viewpoints. This can be

performed by adopting the proposed iteration action between the different proposed tasks.

Thereby, this iteration continues until the produced model reaches a satisfactory state by having no architectural design errors. At the end of the design process, the final designed and validated SN model will be ready to be developed and to enter the deployment phase.

In order to elaborate our proposed tasks of the SN design process, our contributions for each task are presented in the next paragraphs in this section.

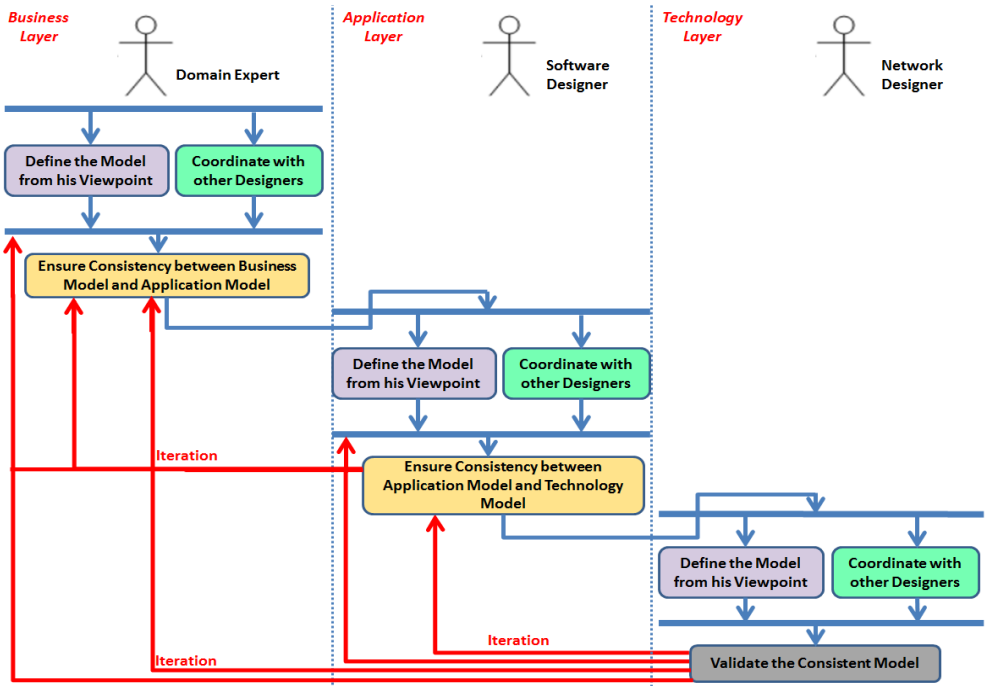

Figure 4.3: Proposed Tasks and Approaches to be Performed by the Different Sensor Networks Designers using ArchiMate Layers

4.5.1 Modeling

Our proposal for ***Task 1 Modeling*** is to integrate DSML for each domain of experience into the process of SN Design. In this way, all the concerned designers can model exploiting the benefits of DSML. More specifically, what meets the following requirements for the SN:

- ***Req 1 Improving Architectural Design:*** DSML aims to define concepts, constraints, and the necessary rules that are required for a specific domain such as SN. In order to define and cover the most needed components and relationships to model a SN system, we can analyze all the possible execution scenario of SN system with different fusion architecture (Centralized, Hierarchical, Distributed) (cf. section 1.5).

For example, after this analysis we can find criteria that are required to be respected during the SN run-time such as the communication rate of the Distributed Fusion Architecture (DFA).

- ***Req 3 Extensibility:*** By adopting the metamodeling approach, the DSML are defined according to the different ArchiMate viewpoints and layers, and the new added required SN concepts are implemented in the generated design tool. These concepts can be displayed as new components, actors, functions and relationships in the palette of generated design tool with their icons. Thus, these new icons can be used by the designers during the proposed modeling task.

- ***Req 4 Heterogeneity Supported:*** DSML could comprise and describe different types of components (hardware and software) and constraints such as communication types that are related to different contexts and domains. Thus, during the design phase, the designer is able to create a complex model that contains different types of relationships that connect different types of components and functions.

In order to use specific concepts of DSML while performing the task of modeling of a viewpoint, we propose to customize ArchiMate to include specific domain concepts such as sensors, fusion servers. For this purpose, we create ArchiMO, a DSML for Marine Observatory.

4.5.2 Ensuring Consistency

Our proposal for ***Task 2 Ensuring Consistency*** is to use the built-in ArchiMate relationships that are specific to relate two separate models from different layers. These relationships connect these two separate models through two components such as actors and functions where each one is in a separate model. These two components are selected to be connected by the experts according to each layer. This reflects the concept of interoperability between several models at different layers. This interoperability is provided by ArchiMate, and it can be performed by allowing the exchange of information between several separate models from different layers through the built-in ArchiMate relationships such as Used by relationship.

Hence, to perform ***Task 2 Ensuring Consistency***, we propose to rely on the interoperability of ArchiMate metamodels by using the relationships that manage the inter-operations between models. The built-in interoperability in ArchiMate metamodel also becomes a built-in interoperability in the new extended DSMLs. This interoperability is provided by the EA frameworks which could be established automatically between the different layers/viewpoints by using built-in ArchiMate inter-relations. So, ArchiMate provides the ability to produce one consistent model.

Consequently, this meets the ***Req 2 Multiple Viewpoints*** SN requirement. This is due to the different viewpoints and these inter-relations that are available for stakeholders according to their specialties and domains of experience such as the domain expert, software designer and network designer (see. Figure 4.3).

4.5.3 Validating

Validation is an assurance task to confirm that the specified requirements have been fulfilled [NAS15]. These requirements are presented in a model that contains physical and conceptual entities and relationships between these entities. The latter, relationships, and their exchange of information, describe the possible required business processes. Once the model is created by using the extended ArchiMate MetaModel (DSML), its validation is required. This validation ensures that the model has no architectural design errors in and between the different concerned viewpoints.

As we adopt and use MDE, we use its model transformation aspect (cf. section 2.2.3). This latter provides the ability to generate code from a model by using code generator. This code generator takes a model as input, and provides a simulation code automatically as output. Thus, we can execute this output on a simulator in order to ensure the validity of the generated code. For this purpose, we can validate a model by simulation.

Next, in case we got architectural errors after the model simulation, we can iterate to modify and adjust the needs in ***Task 1 Modeling*** or in ***Task 2 Ensuring Consistency*** (cf. the red lines and arrows in Figure 4.1). This scenario can be repeated using the iteration approach until we have acceptable simulation results in the create models.

In conclusion, our proposal for ***Task 3 Validation*** is to simulate the defined models during the design phase. The errors that may be made by the designer while defining SN models could be detected before continuing the SN life cycle. Thus, by performing ***Task 3 Validation***, our created SN model could be early validated before the implementation phase by finding, then fixing the architectural errors [ABB+14][MSB11]. Consequently, this meets the ***Req 5 Validation Tools Supported*** SN requirement.

4.6 Discussion

We support the proposed ***Task 1 Modeling*** by defining DSML for SN or by relying on a previous defined DSML [CKRS14]. In order to achieve this task, we can profit from the existing modeling languages ArchiMate, by extending their concepts to obtain our required DSML.

Next, we support ***Task 2 Ensuring Consistency*** by relying on an automatic interoperability between DSML at the semantic levels. This could be ensured as: (1) ArchiMate provides specific relationships to manage the link between business, application, and technology layers; (2) we extend ArchiMate metamodels to define new DSML, so this last contains the same benefits of ArchiMate, as we are not manipulating concepts and rules of ArchiMate metamodels.

Then, we support ***Task 3 Validation*** by selecting a simulator and adopting it in order to validate the created SN models. This early validation activity/step helps the designers by preventing architectural design errors in the later stages such as deployment stage where the maintenance operations are more costly.

Synthesis

In this chapter, we inspired the iteration approach to be performed from the three presented software development approach. Next, we proposed to apply the iteration approach on the three proposed tasks of the proposed SN design process: modeling the SN from the stakeholders' point of view **Task 1 Modeling**, *ensuring consistency of models* **Task 2 Ensuring Consistency**, *and validating the created models* **Task 3 Validating**. *After, we elaborated that MDE is interesting to be used while performing our proposed design process, and to be used while building our proposed tools. Next, we elaborate the content of the three proposed tasks related to our SN domain, and we argue that our proposed process fit with the requirements of SN designer. Then, to support* **Task 1 Modeling**, *we proposed to integrate DSML into the SN design process. This proposed DSML will be elaborated on, in Chapter 5. To support* **Task 2 Ensuring Consistency**, *we proposed to adopt an existing approach, the interoperability between the different layers of the selected modeling language, ArchiMate. This interoperability will be elaborated on, in Chapter 5. To support* **Task 3 Validating**, *we proposed the possibility of simulating models using simulators. This simulation scenario will be elaborated on, in chapter 6.*

5 Domain Specific Modeling Languages and Design Tools for Sensor Networks Design

Reminders and Objectives

In this chapter, we propose a DSML, ArchiMO as an extension of the ArchiMate EA, to deal with **Task 1 Modeling** *that is proposed in the previous chapter. To enable the use of the created DSML during the design process, we propose a design tool for Marine Observatory (MO). To deal with* **Task 2 Ensuring Consistency** *between the suggested DSMLs, predefined relationships are used and extended.* **Task 3 Validation** *of the DSML models is handled through the use of a network simulator. Then, we discuss the advantages of our contributions, and we analyze how they satisfy our SN requirements and research questions.*
Several parts of this chapter are published in [AAKR14], [AAK+14], [AAK+15] and [CGA15].

5.1 ArchiMO Definition

Domain Specific Modeling Language (DSML) is proposed to enable designers to specify their needs and their solutions using domain specific concepts [DBST10]. In our case, we define a DSML, ArchiMO, that is customized for designers for the Sensor Networks domain based on different domains of experience and backgrounds of the system design. In our context, we apply our DSML to Marine Observatory (MO) domain. And in order to define this DSML, we should take into consideration the specific MO components. Thus, like any DSML, ArchiMO comprises three parts: the abstract syntax, the concrete syntax, and the semantics [CGS12]. These parts are detailed in the next sections.

5.1.1 Marine Observatory Context

As we want to first apply our DSML to the Marine Observatory domain, we focus our considerations on this domain. So ArchiMO abstract syntax must define MO concepts, their relationships and constraints by taking into consideration the underwater environmental constraints. In order to define these concepts, we should know on what the MO is based such as [LLL09][**?**]: (1) what is the adopted approach to localize and monitor underwater moving objects, why this approach is adopted, and for what types of applications; (2) what we need as concepts to ensure the execution of the localization; (3) what are the rules that ensure the interaction between the selected concepts; (4) what are the cri-

teria and rules to establish connections between these concepts using specific relationships.

For the first question, data fusion approach were developed primarily for military applications such as radars tracking a moving object since fused data from multiple sensors provide several advantages over data from a single sensor [LLL09] such as more position accuracy of the underwater moving objects. This accuracy is due to the number of receivers (smart sensors in our case) as it will be improved by increasing the number of these sensors [KH05]. However, we cannot ignore the inaccuracy that may happen while localizing moving objects by combining data from two or more smart sensors. This is due to the imprecise object coordinates that may be provided by the sensors to the fusion servers.

Concerning the three other questions, they can be answered and elaborated during the description of the existing architectures for data fusion concept. As we selected previously the appropriate architecture for data fusion concepts which is the distributed one (cf. section 1.6) to be adopted, so the required MO concepts, relationships and constraints are related to the Distributed Fusion Architecture (DFA) [LLL09].

Accordingly, to build consistent MO models, some of the predefined ArchiMate concepts and relationships are useful but they are not specialized enough to satisfy the MO constraints in the design phase. This due to not defining all the possible components in ArchiMate that can be used while defining models in different specific domains. However, they only defined what they thought are essentials. This means, ArchiMate is not a complete language that can be used to create a required model within a specific domain. In addition, ArchiMate is used to design systems for general purposes, so it is not useful for specific domains such as MO. For this purpose, we always need to extend ArchiMate with new specific components or relationships that we find important. Therefore, we present in the next section, what the selected ArchiMate concepts and relationships are that we want to extend and why, in order to satisfy the MO constraints in the design phase.

5.1.2 Selected ArchiMate Concepts and Relationships

The MO designers define a consistent model that reflects a real description of detecting the under moving objects. This description requires software and hardware concepts to be used by the MO designers while performing the design phase. These concepts are elaborated below according to the ArchiMate business and application layers:

5.1.2.1 Business Layer

In order to define a MO model in ArchiMate business layer, the domain expert requires behavioral, structural concepts and relationships to be used. The role of structural concepts is to perform the behavioral concepts by using structural and dynamic relationships, and by taking into consideration the required MO constraints. The structural concepts are: Smart Sensors (SS) that can never be connected with other SS and it can be only connected with DFS under a criteria, Data Fusion Servers (DFS). Regarding the behavioral concepts are: Algorithm Selection (AS), Data Transmission (DT), Data

Acquisition (DA) and Object Localization Algorithm (OLA). Concerning the structural relationships are: Assignment, Association and Used By. Regarding the dynamic relationships are: triggering. More specifically, the defined MO model describes the following scenario:

SS get underwater data from the moving objects and process them, then transfer them to DFS. However, DFS aggregate data from the different SS in order to apply the object localization algorithm. Next, we assign a specific code to AS in order to select the appropriate algorithm as we consider we may have multiple different algorithms that are performed by the DFS. Once the SS detect the moving objects by DA and the OLA is selected, so the localization is obtained and transferred by DFS on the network using DT. Then, in order to precise that a DFS performs AS or DT, we need to use the assignment relationship between them. The association relationship can be used when we want to relate two components without restrictions. The Used By relationship can be used when we want to relate two components in two separated models, each one is in a separate ArchiMate layer. This can be performed by relating a specific concept from a layer to another concept from another layer. After, by using triggers between the different behavioral concepts, the designer can organize and present these concepts in the created model.

Consequently, as we need to apply the MO constraints while defining MO models, the predefined ArchiMate concepts and relationships are not enough to be used for building consistent MO models. However, some predefined structural concepts and relationships such as the business actor has the ability to perform some behavioral concepts such as the business function using relationships. The business functions acquire data, localize fixed and moving objects. So, we can exploit the properties of these concepts by using them in our MO model. However, they miss the required MO constraints that can be applied while defining MO models. For this purpose, we select the business actor and business function to be extended by adding the missing MO constraints, in the ArchiMate business layer.

5.1.2.2 Application Layer

In order to define a MO model in ArchiMate application layer, the software designer requires behavioral, structural concepts and relationships that are specific for application layer. The structural concepts are: Smart Sensor Systems (SSS) and Fusion Systems (FS) that are the correspondence concepts respectively of the SS and DFS concepts in the business layer. Regarding the behavioral concepts are: Manage Resources (MR), Coordinates Storage Handling (CSH), Compute Coordinates (CC), Transmit Localization Data (TLD), Inform Server (IS), Voice Streaming and Video Streaming. Concerning the required relationships are the same as the required ones in the business layer. More specifically, the defined MO model describes the following scenario:

FS is used to perform the detail or the correspondence sequential application functions of OLA, which are: MR, CSH, CC, TLD. SSS is used to perform the IS, Voice Streaming and Video Streaming application functions. The first one is used to inform the

fusion server about the detection of an underwater moving object. This function is useful to notify the DFS by performing the localization algorithm upon any detection by SS of moving objects. The second one is used in case the SS are hydrophones. And the third one is used in case the SS are underwater cameras.

Consequently, the presented concepts are not available in the ArchiMate application layer. However, some predefined concepts in application layer such as the application component has the ability to perform some application functions using relationships. So, we can exploit the properties of these concepts by using them in our MO model. However, they miss the required MO constraints that can be applied while defining MO models. For this purpose, we select the application component and application function to be extended by adding the missing MO constraints, in the ArchiMate application layer.

5.1.3 ArchiMO MetaModel

As ArchiMO extends ArchiMate, ArchiMO metamodel contains the semantics of the language ArchiMate such as: association, assignment and generalization relationship. According to the previous section, ArchiMO metamodel should contain specific MO concepts, relationships and constraints. MO concepts are extended from ArchiMate Business Actor and Business Functions, MO association relationship that is extended from ArchiMate association relationship and a constraint to establish such a relationship.

As ArchiMate is composed of three layers: business, application and technology layer, we can extend these three layers in order to introduce our ArchiMO concepts and extend only two layers, the business and application. [Chi12] extended ArchiMate technology layer to define a DSML for IP Multimedia Subsystem (IMS), for telecommunication domain. This DSML is useful for different types of application in different domains, as IMS provides functions that can be used with different domains such as MO [All16]. For example, IMS allows to exchange messages between terminals such as cameras, hydrophones, smart sensors, and Fusion Servers. Therefore, the IMS metamodel is required for our MO systems and to model the deployment of our application, or on a technical infrastructure.

Concerning ArchiMO, the two proposed part of the metamodel are shown in figures (5.1, 5.2). The white concepts in the figures are ArchiMate predefined ones. However, the green (or grey) are the MO concepts. The new added constraint, is illustrated in the red X in figure 5.3. And, the new extended relationship is shown in the red text in figure 5.4 and in the red rectangle in the 5.6. We will select and present these extended concepts, relationships and constraints, in the next sub sections.

5.1.3.1 Concepts

ArchiMO is composed of two views: one for the business layer, and another for the application layer. We present our metamodel as the following:

1. Business Layer, see Figure. 5.1: we have extended the business actor of ArchiMate through two new concepts, the Smart Sensor and the Data Fusion. Smart Sensor

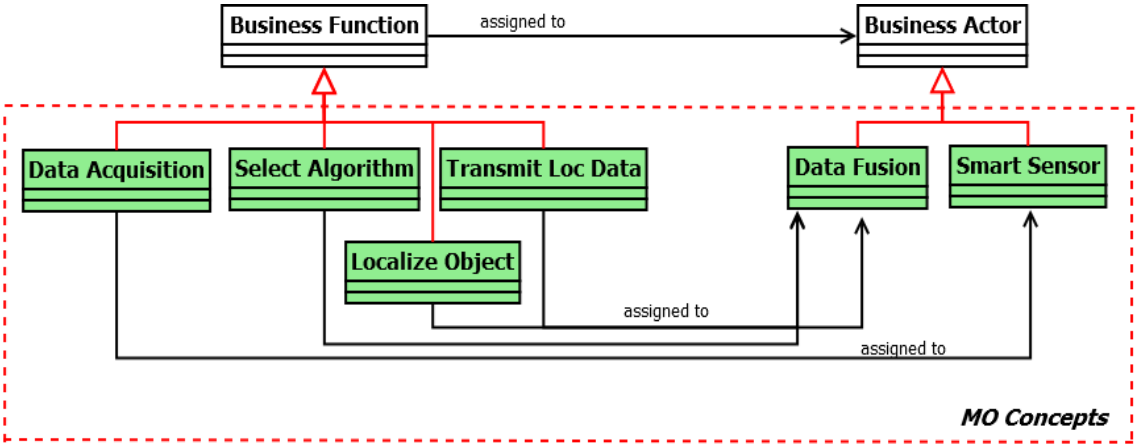

Figure 5.1: ArchiMate Business Layer

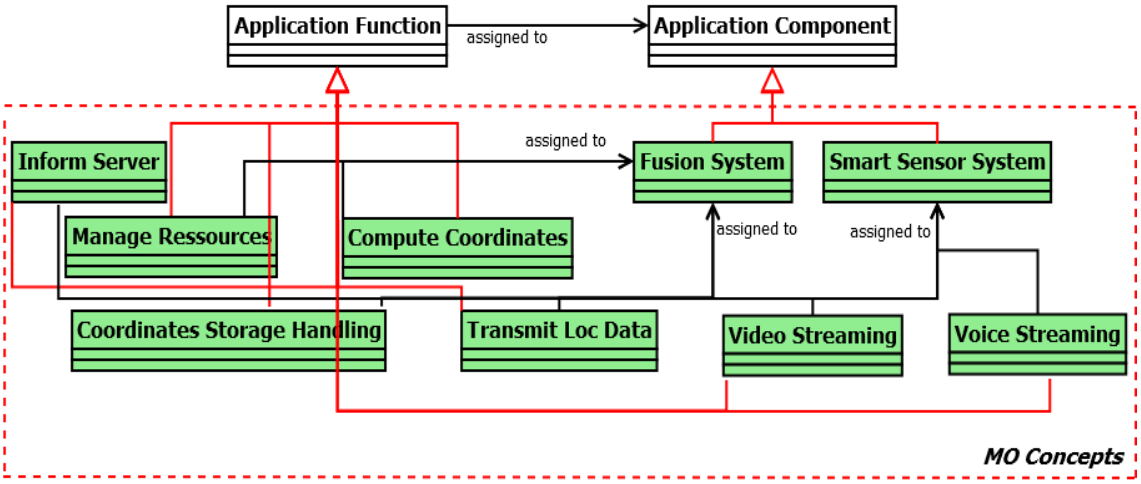

Figure 5.2: ArchiMate Application Layer

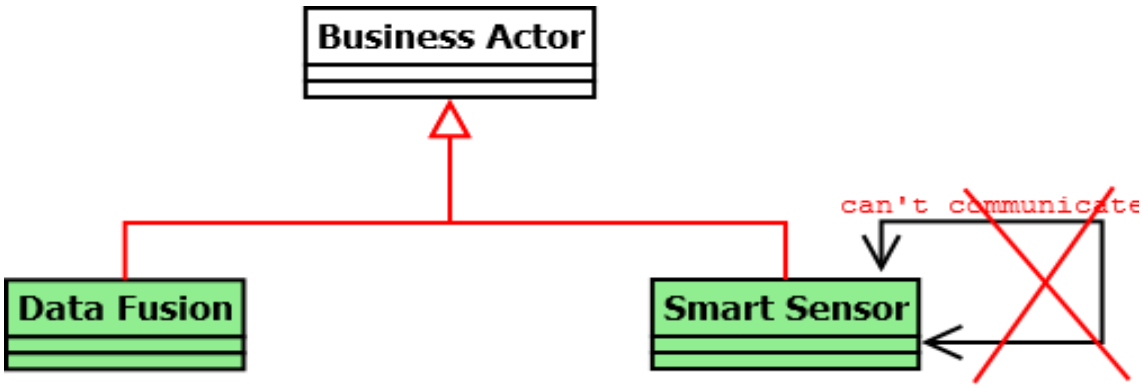

Figure 5.3: Communication Constraint between two Smart Sensors

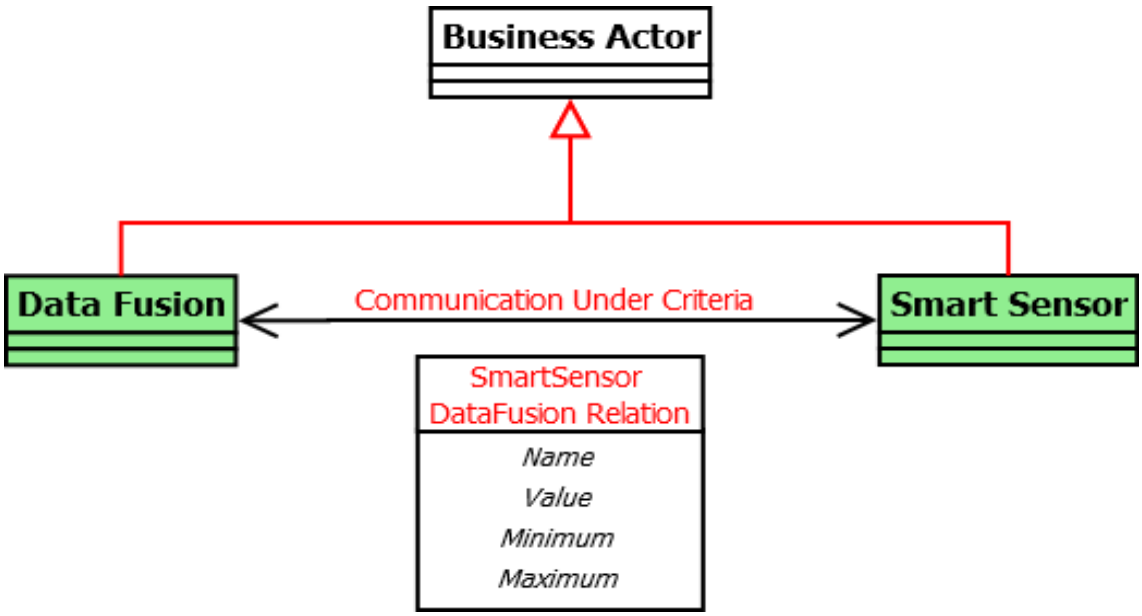

Figure 5.4: Extended Relationship between Smart Sensor and Data Fusion Server

is responsible for performing the raw Data Acquisition, while the Data Fusion is responsible for other functions: (1) Algorithm Selection performs a procedure to fetch then select the proper algorithm in case we have several algorithms with different functionalities on the same Data Fusion server; (2) Data Transmission transmits the data between the different Data Fusion components existing on the SN; (3) Object Localization makes the necessary actions to call the localization algorithm of the underwater moving objects. These functions extend the business function concept in ArchiMate. The Data Acquisition function has a main role which is receiving the signals that are thrown by the underwater moving objects in order to be processed and performed by the Smart Sensors.

2. Application Layer, see Figure. 5.2: we have extended the application component of ArchiMate through two new concepts, the Data Fusion System, and the Smart Sensor System. Data Fusion System is responsible for performing the following functions:

   (a) Manage Resources: to manage the resources needed for the algorithm execution. This due to the structure of any algorithm, a reservation and initialization of a set of variables are required, points' coordinates such as x, y, z of each smart sensor. Data Fusion System (DFS) starts managing the resources for the object localization algorithm.

   (b) Coordinates Storage Handling: to store the coordinates correlated with time. This is due to the needs of storing temporarily the received values (e.g. distance between the underwater moving object and a Smart Sensor) by the Smart Sensors in order to use them in the object localization. DFS starts handling the

storage of received coordinates by saving these received values in the reserved coordinates according to each Smart Sensor.

(c) Compute Coordinates: to compute the position according to a specific algorithm selected previously by the Date Fusion actor. This is due to the applying of data fusion concept. The data comes from several SSs in order to be fused and calculated. DFS starts computing the stored coordinates of different SSs by calculating the intersection between the different covered detected areas (e.g. intersection of several circles or spheres) by the SS.

(d) Transmit Localization Data: to exchange information between the fusion servers/systems. This is due to the needs of exchanging the localization information between the existing Data Fusion severs on the SN. DFS starts transmitting the localization information once the calculation is performed on it. And the same DFS can receive a calculated localization information from another DFS.

The Smart Sensor System is responsible to perform the following functions:

(a) Inform Server: to inform the fusion server about the detection of an underwater moving object. This function is useful to notify the DFS by performing the localization algorithm upon any detection by SS of moving objects.

(b) Voice Streaming: this function is useful in case the Smart Sensors are hydrophones.

(c) Video Streaming: this function is useful in case the Smart Sensors are underwater cameras.

The functions presented above extend the application function concept in ArchiMate.

In order to formalize the extended concepts, we define constraints using Object Constraint Language (OCL). This is due to the hard task and challenges that will face the SN designer while specifying and defining such complex constraints.

According to [CMSD04][CT07][EKW92], as the designer uses ArchiMO as a modeling language and adopts MDE (cf. chapter 2) to model MO systems, he faces some challenges such as: (1) the ability of defining and executing the transformations of models (source and target) as the model transformation is one of the main aspect of MDE (cf. section 2.4); (2) many constraints (e.g. MO constraints) cannot be expressed using only conceptual modeling languages such as UML, ArchiMate or ArchiMO .

As we know, the model transformation should contain two types of constraints [CMSD04]: (1) appoint constraints on the input (source) and output (target) models; (2) define constraints on the relationships among input and output models. Thus, we can define the model transformation as a set of three constraints:

- a set of constraints to be fulfilled for a model to be selected as an input model of the transformation.

- a set of criteria (constraints and rules) on the relationships and development of concepts from the input to the output model.
- a set of constraints as a valid output model generated by the transformation criteria.

Thereby, in order to face the two challenges, the SN designer should fulfill the different mentioned constraints above. These constraints can be appointed, defined and expressed by using Object Constraint Language (OCL). OCL is set by the Object Management Group (OMG) that is a language that enables the designer to describe expressions, terms, rules and constraints on models that are created by conceptual modeling languages [Mar08]. Thus, the SN designer should use OCL with any conceptual modeling language in order to produce a valid output model which could be as a preparatory phase for its simulation. Once the output models are validated, so the formalization of the defined models is achieved and succeeded. For this purpose, we formalize the extended MO concepts, in the next section.

5.1.3.2 Concepts Formalization

The new extended constraints, the assignment relationships between specific MO concepts in the both ArchiMate layers, business and application. These constraints are expressed using OCL (cf. the line 5 in the frame below), and the evaluation must return true for this constraint (cf. the line 7 in the frame below). Therefore, this constraint is used to assign only a Data Acquisition function to a Smart Sensor actor according to some MO and DFA criteria, that is verified during its evaluation in the frame below. For the other similar constraints, they are formalized within the same way.

```
1  // Assignment Relationship between DataAcquisition and SmartSensor

3  Context:
   Data Acquisition
5  Evaluating:
   self.assignTo.oclIsKindOf(SmartSensor)
7  Results:
   true
9
   Context:
11 Data Acquisition
   Evaluating:
13 self.assignTo.oclIsKindOf(DataFusion)
   Results:
15 false
```

The new extended constraints, the association relationship between two Smart Sensors. This constraint is expressed using OCL (cf. start at line 5 in the frame below), and it returns false as a result (cf. the line 11 in the frame below). Therefore, this constraint is used between two Smart Sensors according to some MO and DFA criteria. In case, the association relationship is between two Data Fusion actors, the returned result is true. This constraint is to prevent the association between two smart sensors only without affecting the other MO and ArchiMate concepts.

```
1  // Association Relationship between two SmartSensors

3  Context:
   Smart Sensor
```

```
Evaluating:
(self.archimateModel.folders->collect(elements)->
  select(e | e.oclIsKindOf(AssociationRelationship)))->
    exists(e | e.oclAsType(AssociationRelationship).source=self and
     e.oclAsType(AssociationRelationship).target.oclIsKindOf(SmartSensor))
Results:
false

Context:
Data Fusion
Evaluating:
(self.archimateModel.folders->collect(elements)->
  select(e | e.oclIsKindOf(AssociationRelationship)))->
    exists(e | e.oclAsType(AssociationRelationship).source=self and
     e.oclAsType(AssociationRelationship).target.oclIsKindOf(DataFusion))
Results:
true
```

5.1.3.3 Relationships

Mainly, ArchiMate contains different types of relationships such as association, assignment, see Figure. 5.5. Assignment relationships are used to link the structural and functional elements in the design modeling language. These type of relationships help us to represent the behavioral aspects of the design in the verification tools and relate them to the right structural element. For instance, in the technology layer, if a function1 is assigned to node1, this means that function1 should be performed by the node1. This concept can be mapped in the programming languages such as C++ for the NS-3 simulator using the concept of classes and their own functions. Thereby, the presented ArchiMate relationships can not only satisfy our MO domain in the design phase.

For this purpose, to fulfill the needs of the SN designer in the MO domain concerning the MO constraints, we have specialized the definition of these relationships with adding new constraints related to the new added concepts. In our context, we have defined the association relationship for the smart sensor according to the constraints of DFA, such as smart sensor that could be only associated to the data fusion and two smart sensors that could not be associated together, see Figure. 5.3. Furthermore, we have defined the assignment relationship for the smart sensor according to the constraint of MO [**?**] such as Smart Sensor could be only assigned to the Data Acquisition.

ArchiMO metamodel contains new relationships that are specific to the MO domain. This is due to the architectural design errors that may be made by the SN Designer while relating a Smart Sensor to a Data Fusion server. These errors are costly to be occurred in implementation stage of MO life cycle system. For example, a smart sensor communicate with fusion server in the defined model, however, in the implementation phase, the expert people can not deploy this communication for an imprecise length in the marine cable. Therefore, a new specific relationship for the Smart Sensor and Data Fusion elements is added to the semantics. This relationship is used only to connect a Smart Sensor element to a Data Fusion element. We consider this new type of relationship in the business layer as a logical relationship. In order to implement this new relationship, we have extended the Association Relationship by a new one which is the Smart Sensor

and Data Fusion Relation (SDR), see Figure. 5.6.

Consequently, this section deals with the ***Task 1 Modeling***. Our extended DSML answer ***Req 1 Improving Architectural Design***, ***Req 3 Extensibility*** and ***Req 4 Heterogeneity Supported***.

In order to formalize the extended MO relationships, also we define constraints using Object Constraint Language (OCL). For this purpose, we formalize these extended relationships, in the next section.

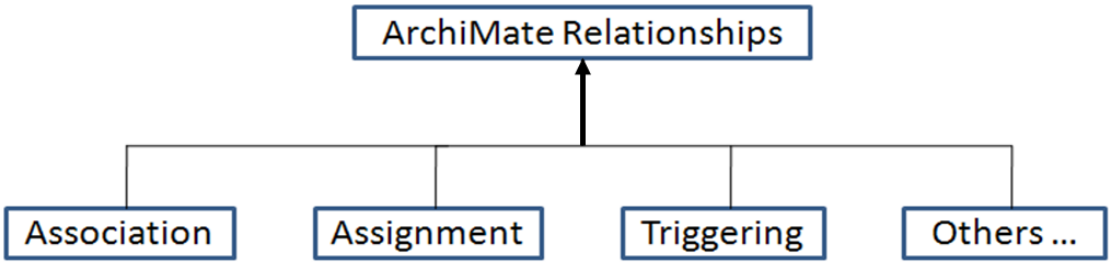

Figure 5.5: Conceptual ArchiMate Relationships

5

5.1.3.4 Relationships Formalization

The new extended constraints, the association relationship between Smart Sensor and Data Fusion (SDR), see Figure. 5.4 is evaluated using OCL (cf. the line 5 in the frame below), and it returns true as a result (cf. the line 7 in the frame below). Therefore, SDR is used to connect only a Smart Sensor element to a Data Fusion element by satisfying the MO criteria such as the length of marine cable that is entered by the designer. This cable should be between 10 and 50 meters. This constraint is verified during its evaluation in the frame below. Otherwise, SDR is not formalized as the return results is false (cf. the line 15 in the frame below).

```
1  // SmartSensor DataFusion Relationship (SDR)

3  Context:
   SmartSensor DataFusion Relation
5  Evaluating:
   self.value > 10 and self.value < 50
7  Results:
   true
9
   Context:
11 SmartSensor DataFusion Relation
   Evaluating:
13 self.value > 10 and self.value < 50
   Results:
15 false
```

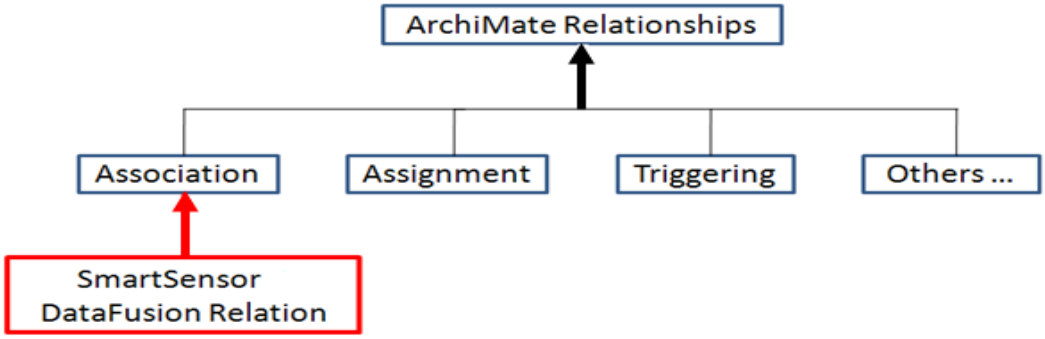

Figure 5.6: Extended Relationship

5.1.3.5 Formal Constraints

According to the use of OCL in the previous sections, ArchiMO metamodel contains formalized constraints that are specific to the MO domain. These constraints are according to the selected distributed fusion architecture (DFA) from [LLL09], our MeDON case study [**?**], and the adopted data fusion approach from [MSDW01][LLL09]. For Smart Sensor: (1) communication between two Smart Sensor elements is not allowed; (2) communication between Smart Sensor and Data Fusion element is allowed; (3) Smart Sensor is only allowed to be related to the Data Acquisition function. For Data Fusion: (1) communication between two Data Fusion elements is allowed; (2) Data Fusion is only allowed to be related to Algorithm Selection, Data Transmission and Object Localization functions.

5.1.4 ArchiMO MetaModel Layer Consistency

ArchiMO metamodel ensures consistency between the two extended business and application metamodels, and between the one extended by [CKR12] which is the metamodel of technology layer. The consistency between these extended metamodels of the ArchiMate layers is supported by the predefined relations between ArchiMate layers such as Used By and Realization. For example, the consistency between the business layer and the application layer of ArchiMate, also between the application layer and the technology layer. Therefore, the consistency is available at metamodels level, so the SN designer inter-relates several models from different layers. All the predefined ArchiMate relationships, Used By and Realization, can be used in the design phase, see. Figure 5.7. Thus, by using these predefined relationships between ArchiMate layers, the SN designer exploit this advantage to have one MO model that contains concepts and constraints from multiple ArchiMate layers.

In addition, the consistency between layers has also the advantage of providing the base of an automatic generation of concepts and relationships in the application layer

according to a satisfaction of a constraint in the business layer related to the new extended relationship, the SDR. In this case, the SN Designer can assign the proper value to the relationship that connects a smart sensor to a data fusion server, see. Figure 5.8. This ability is available for the SN designer as a modeling facility and based on a generative approach.

In order to implement this automatic generation, we throw the corresponding MO, ArchiMate, and relationships automatically in the application layer during the modeling time of the design phase in the business layer, the green boxes in Figure 5.8. The creation of these mix elements between specific MO and Archimate elements is occurred upon establishing the SDR, see. Figure 5.8.

The generated ArchiMate concepts are: Application Function and Application Component. The generated MO concepts are: Smart Sensor System, Data Fusion System, Inform Server, Manage Resources, Compute Coordinates, Coordinates Storage Handling and Transmit Loc Data. The triggering and the assignment relationships are ArchiMate elements. This is due to MO domain and specifically to the satisfaction of SDR which requires the creation of these mix generated elements in the level in the design phase. Thus, instead of creating them by the SN Designer, he can get them automatically after satisfying the SDR constraint.

Therefore, by adopting this generative approach on all the activities such as concepts, relationships of MO domain in the business layer, then on all the activities of the application layer, we can have on modeling time an automatic mapping between the three layers.

Consequently, this section deals with ***Task 2 Ensuring Consistency***. We decided to rely on the different forms of ArchiMO consistency, predefined relationships and generation concept between layers between metamodels, particularly in order to answer ***Req 2 Multiple Viewpoints***.

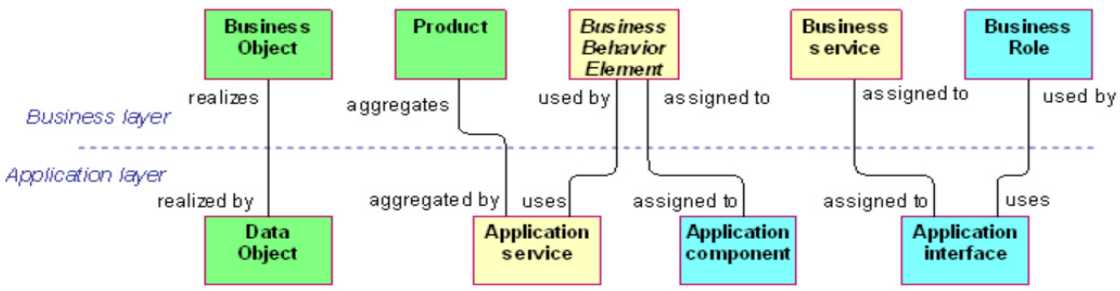

Figure 5.7: Consistency between Business Layer and Application Layer

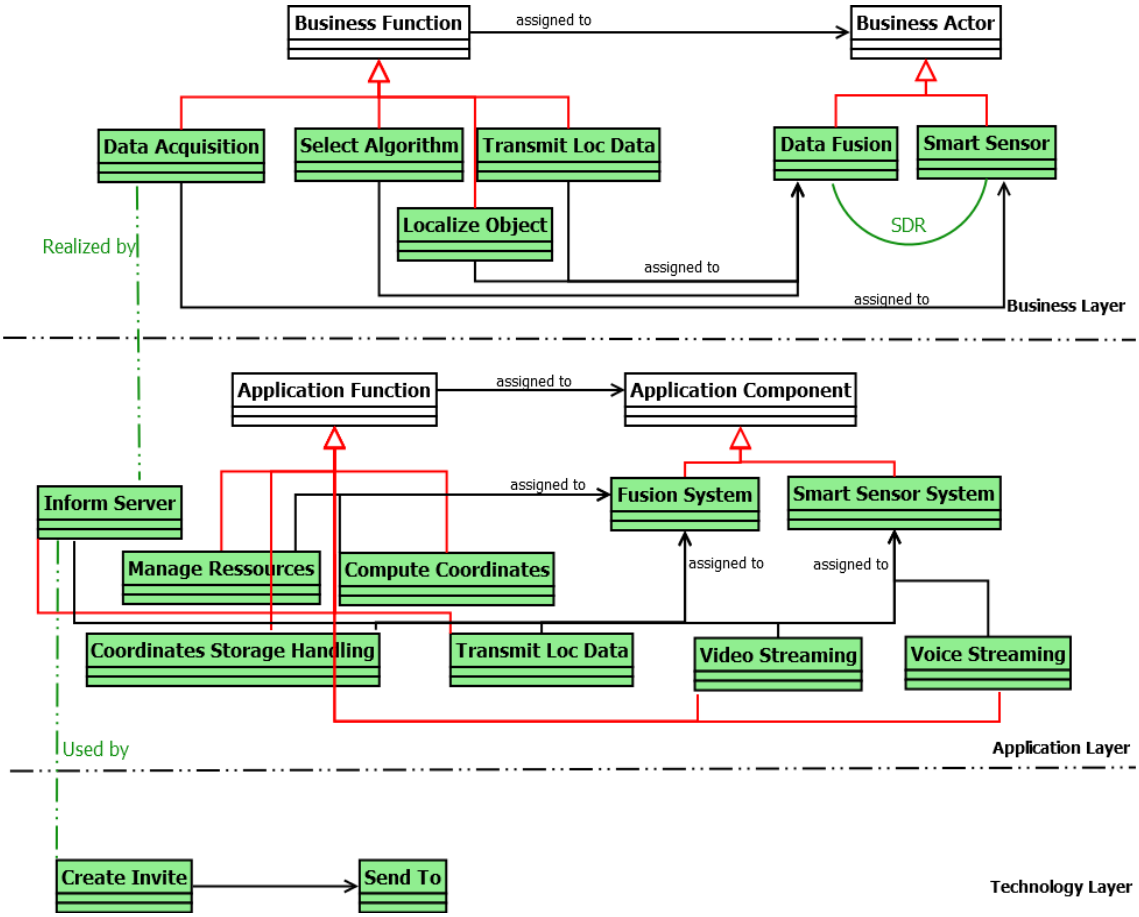

Figure 5.8: Generated MO Concepts and Relationships in the Application and Technology Layer

5.1.5 Formalization of Layers Interoperability

The MO business models are transformed to be MO application models. This Model to Model transformation is formalized by using the Realization ArchiMate relationship. This relationship must be unique in the consistent model that relate the Data Acquisition business function to the Inform Server application function (cf. the green line between Data Acquisition and Inform Server in Figure 5.8). This constraint is evaluated using OCL (cf. start at line 5 in the frame below), and it returns true as a result (cf. the line 11 in the frame below). This means, there is a Realization relationship between Data Acquisition and Inform Server. However, the code that starts at line 15 in the frame below, evaluate if the Realization relationship relate Inform Server to Object Localization, the returned results is false (cf. the line 21 in the frame below). Therefore, this constraint is formalized, and this relationship is unique and only allowed between Data Acquisition and Inform Server according to some MO and DFA criteria.

```
1  // Realization Relationship between Data Acquisition business function and Inform
       Server application function

3  Context:
   Inform Server
5  Evaluating:
   (self.archimateModel.folders->collect(elements)->
7    select(e | e.oclIsKindOf(RealisationRelationship)))->
       exists(e | e.oclAsType(RealisationRelationship).source=self and
9       e.oclAsType(RealisationRelationship).target.oclIsKindOf(DataAcquisition))
   Results:
11 true

13 Context:
   Inform Server
15 Evaluating:
   (self.archimateModel.folders->collect(elements)->
17   select(e | e.oclIsKindOf(RealisationRelationship)))->
       exists(e | e.oclAsType(RealisationRelationship).source=self and
19      e.oclAsType(RealisationRelationship).target.oclIsKindOf(ObjectLocalization))
   Results:
21 false
```

Also, the MO application models are transformed to be MO technology models. This Model to Model transformation is formalized by using the Used by ArchiMate relationship. This relationship must be unique in the consistent model that relate the Inform Server application function to the Create Invite technology service that is extended by [Chi12] (cf. the green line between Inform Server and Create Invite in Figure 5.8). This constraint is using OCL (cf. start at line 5 in the frame below). This means, there is a Usedby relationship between Inform Server and Create Invite. However, the line 15 in the frame below, evaluate if the Usedby relationship relate Create Invite to Compute Coordinates, the returned results is false (cf. the line 21 in the frame below). Therefore, this constraint on the relationship is only allowed between Inform Server and Create Invite according to some MO and DFA criteria, that is verified during its evaluation in the frame below.

```
1  // Usedby Relationship between Inform Server application function and Create Invite
       IMS infrastructure service

3  Context:
   Create Invite
5  Evaluating:
```

```
(self.archimateModel.folders->collect(elements)->
  select(e | e.oclIsKindOf(UsedbyRelationship)))->
    exists(e | e.oclAsType(UsedbyRelationship).source=self and
     e.oclAsType(RealisationRelationship).target.oclIsKindOf(Inform Server))
Results:
true

Context:
Create Invite
Evaluating:
(self.archimateModel.folders->collect(elements)->
  select(e | e.oclIsKindOf(UsedbyRelationship)))->
    exists(e | e.oclAsType(UsedbyRelationship).source=self and
     e.oclAsType(RealisationRelationship).target.oclIsKindOf(ComputeCoordinates))
Results:
false
```

5.1.6 ArchiMO Design Tool

The new extended metamodel ArchiMO enables us to generate design tool that is coherent with Archi design tool but contains additional concepts, relationships and constraints that are specific to the MO domain [MSDW01][LLL09].

5.1.6.1 ArchiMO Tool Generation

The figure 5.9 describes the process to create ArchiMO design tool based on the extended metamodel. We implemented the new extended concepts, relationships and constraints that are specific to the MO domain and for data fusion concepts [LLL09]. This implementation is performed according to the business and application ArchiMate Layer by weaving java classes, corresponding to MO concepts, in the source code of the ArchiMate tooling.

The ArchiMO Design Tool is generated from the Archi design tool that is based on the java code of ArchiMO metamodel, see. Figure 5.9. ArchiMate is based on Eclipse Modeling Framework (EMF).

5.1.6.2 ArchiMO Concrete Syntax

For each extended concept or relationship, a graphical concrete syntax should be defined [CKRS14]. The concrete syntax is associated with a set of rules which defines the representation of the abstract syntax [BJKV06]. In order to have a graphical view for the added concepts, relationships of each ArchiMate layer. In addition, these metamodels elements are presented in graphical views such as the palettes in order to allow the specific designers to use them by dragging and dropping.

In relation with ArchiMO metamodel, the SN designers are able to define a model using concepts from ArchiMO metamodel. This model can be composed from several inter-related models. Each model can contain many different related elements from a specific viewpoint according to the concerned ArchiMate layer. ArchiMO is divided into three: (1) ArchiMO metamodel for business layer; (2) ArchiMO metamodel for

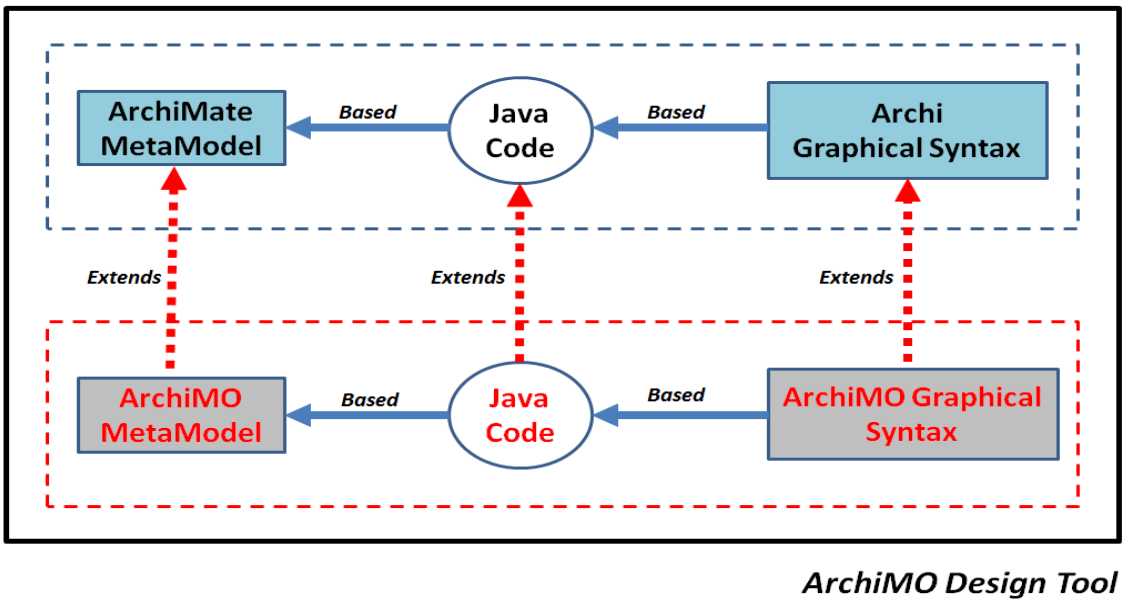

Figure 5.9: Generated ArchiMO Design Tool after the Extension of ArchiMate

application layer; (3) IMS metamodel for technology layer [CAKR11]. Thereby, the elements of the defined models are composed of the three layers with their graphical syntax.

The concrete syntax that is associated with these added concepts and relationships can be implemented in the design tool such as ArchiMO Design Tool for our context. Our proposed concrete syntax is shown in the palettes of the business, in the red circles on the right of Figure. 5.10, and the application, in the red circles on the left of Figure. 5.10 layers. Also, the extended SmartSensor and DataFusion Relationship is shown in the red circle of Figure. 5.11. These palettes are coherent with MO specific concepts and relationships from which the designer can select and use to create MO models.

5.1.6.3 Constraint Implementation

During the model edition, all the constraints specified for the MO extension are checked: (1) prevent designer to associate two Smart Sensor elements together; (2) the designer is able to associate a Smart Sensor element to Data Fusion, Business Actor or other actors, see Figure. 5.12; (3) the assignment is only allowed from SmartSensor to the DataAcquisition function, see Figure. 5.12. Concerning the Data Fusion element: (1) the association between two Data Fusion elements is allowed; (2) the designer is able to associate Data Fusion element to Smart Sensor element, see Figure. 5.12; (3) the designer is able only to assign the Data Fusion to the Algorithm Selection, Data Transmission and Object Localization functions, see Figure. 5.12.

The constraint of the relationship between SmartSensor and DataFusion must be

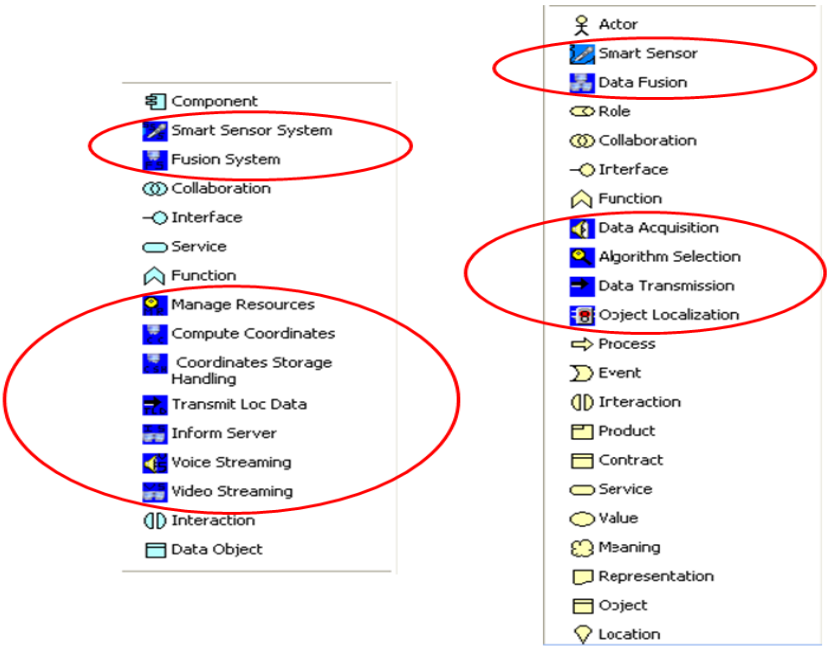

Figure 5.10: Business and Application Layers (Palettes)

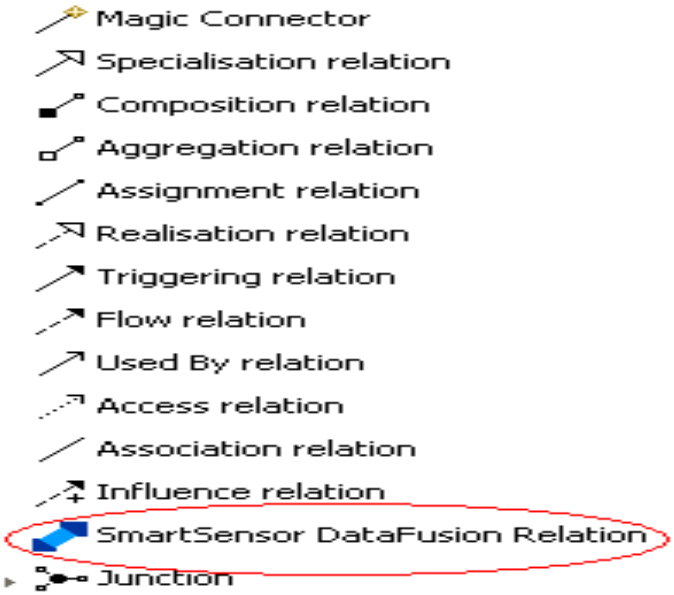

Figure 5.11: Extended SDR Relationship in Palette

checked. It requires the designer to enter a proper value in order to associate a Smart Sensor to a Data Fusion using the SDR, see Fig. 5.13. For example, SDR reflects the marine cable during the modeling task, to connect a Smart Sensor to a Data Fusion Server. However, the length of this cable should be well defined by respecting the minimum and maximum length that could be entered by the designer. Otherwise, the designer is not able to associate a Data Fusion to a Smart Sensor using this SDR, see Fig. 5.14.

By extending SDR, we distinguish the following features: (1) generating the required MO concepts and relationships in the application layer; (2) generating the required ArchiMate concepts and relationships in the application layer. So, once the SDR constraint is checked and verified:

1. The required and necessary MO elements and relationships are generated in the application layer (Inform Server, Manage Resources, Coordinates Storage Handling, Compute Coordinates, Transmit Localization Data, Smart Sensor System, Data Fusion System). More specifically, in the application layer of the Figure. 5.15, Inform Server class is represented by Inform ServerA instance, Manage Resources class is represented by System Resources ReservationA instance, Compute Coordinates class is represented by Compute CoordinatesA instance, Coordinates Storage Handling class is represented by Storage CoordinatesA instance, Transmit Loc Data class is represented by Coordinates Transmission To B or Coordinates Transmission To C instance, Smart Sensor System class is represented by Smart Sensor SystemA instance, and Data Fusion System class is represented by Fusion SystemA instance.

2. The required ArchiMate elements and relationships are created in the application layer (triggering and Assignment are ArchiMate relationships). More specifically, in the application layer of the Figure. 5.15, triggering relation is represented by the different existing relation between the following instances of MO classes: Inform ServerA, System Resources ReservationA, Compute CoordinatesA, Storage CoordinatesA, Coordinates Transmission To B or Coordinates Transmission To C. Assignment relation is represented by the relation between the Smart Sensor SystemA and Inform ServerA instances, and between the Fusion SystemA with the rest of existing instances.

The constraint on the SDR enables the designer of the business layer to create automatically both MO specific and ArchiMate concepts in the viewpoint of the application layer. Once the designer of the business layer finishes the design of MO business layer, the designer of the application layer will get automatically his specific MO models/instances. For example, when the designer of the business layer connects two MO elements such as SmartSensor and DataFusion, the necessary corresponding MO elements such as Smart Sensor System Component, Inform Server Function. And relations are automatically created in the application layer, see Figure. 5.15.

Thereby, ArchiMO design tool helps the designers of each ArchiMate layer to model the system in a highly abstract way, by dragging and dropping the elements and relations from the palette. Also, it helps by avoiding syntax errors that may be made during the design phase.

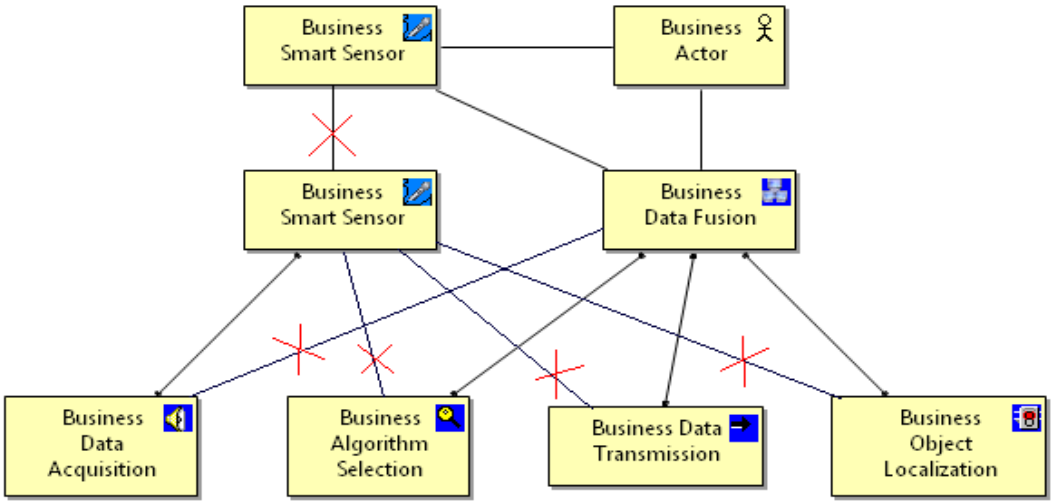

Figure 5.12: Association and Assignment Relationships

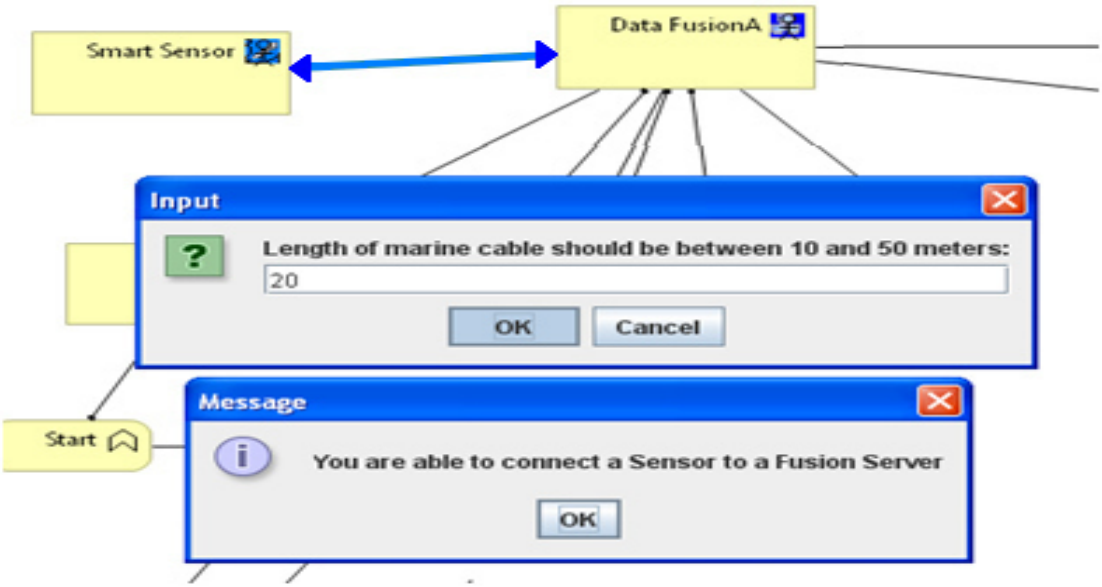

Figure 5.13: Smart Sensor and Data Fusion Relationship is allowed

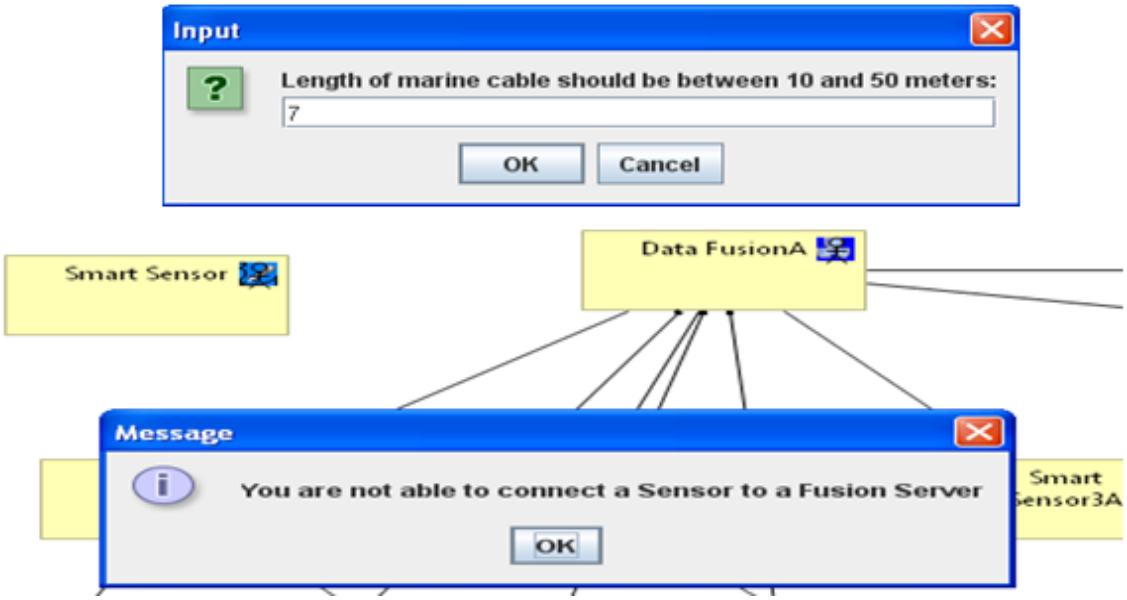

Figure 5.14: Smart Sensor and Data Fusion Relationship is not allowed

5

5.2 Generation of Simulation Code

Once the MO application models are transformed to be MO technology models, the consistent MO model is ready to be the input for the code generator XPAND that is developed by [All16]. Each model transformation depends on a set of rules that describes and controls the transformation process [Par12]. XPAND contains transformation similar to the previous transformation that transforms the architecture of multiple viewpoints of ArchiMate from high abstract MO design to the functions and actions of Networks simulators. The structure of this process is based on the object oriented approach, classes and operations concepts. In order to perform this transformation process, mapping rules are implemented through XPAND. Figure 5.16 illustrates the concepts of business and application viewpoints with their correspondences concepts in the network simulator. This figure indicates the concepts of viewpoints by a black text, and the concepts of network simulator by red arrows and text. Each business actor/role or application component is transformed to an object oriented class. For example, each Smart Sensor or Data Fusion in business and application layers become a C++ class. Each business or application function is transformed to an implemented function in a class. For example, the object localization function in the business layer become an operation in the C++ class. Each relationship that is specific to relate two ArchiMate layers, is transformed to an association relationship. Each trigger relationship is transformed to a call between two functions. The start business function is transformed to a start technology function.

This Model to Text transformation is formalized by relying and adopting the implemented mapping rules in XPAND [All16]. The red concepts in 5.16 are elements of a programming language code such as elements of C++ that can be executed and simulated to obtain specific results.

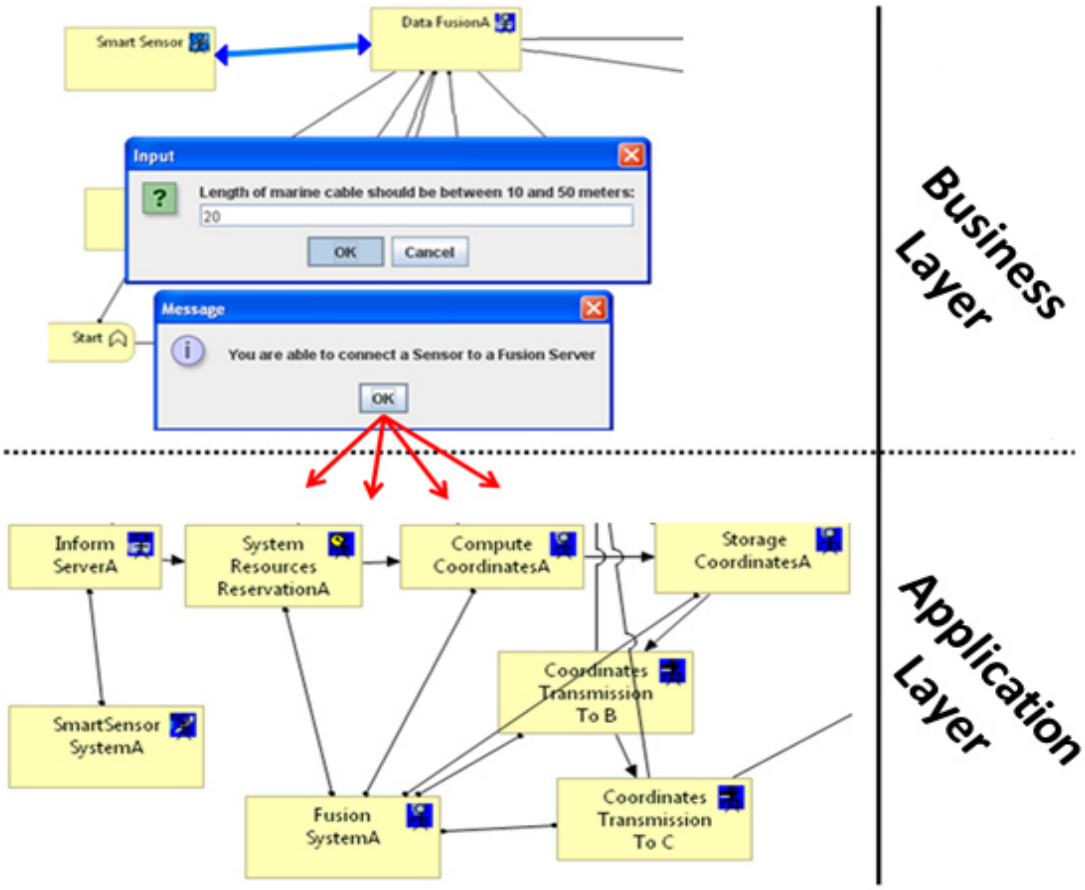

Figure 5.15: Generated MO concepts, Relationships and Constraints in the Application Layer after Entering a Proper Value

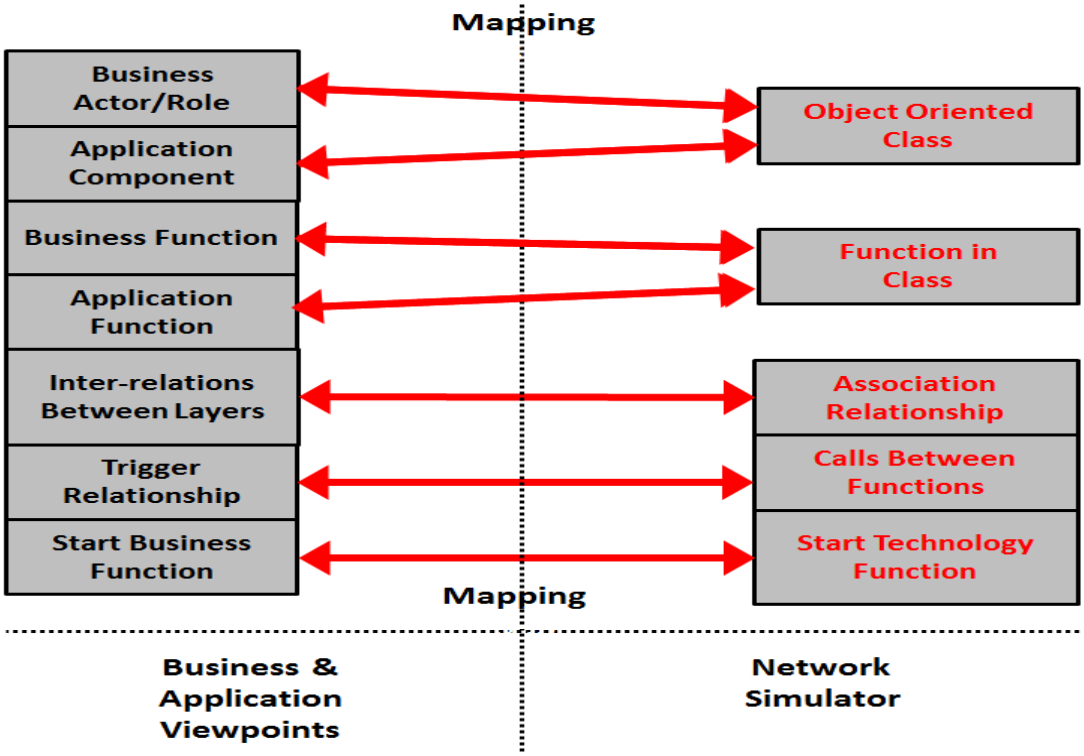

Figure 5.16: Mapping of Business and Application Viewpoints with Network Simulator

5.3 ArchiMO and Iterative Approach

In order to have the two forms of consistency between the different created models according to the different ArchiMate layers, several specific elements are required to be inter-related through specific relationships. These elements and relationships can be predefined ones such as used by and business object. Or new extended ones such as SDR, smart sensor and data fusion. This is due to the specialization of predefined and new extended ArchiMate concepts and relationships, as each relationship can be used between set of specific predefined elements in ArchiMate metamodel, or set of new extended elements in ArchiMO. Thus, there are specific relationships to relate specific elements within the same model, and at the same time, each element can be located in a separate and different model. For example, the SDR is specialized to relate a Smart Sensor to a Data Fusion server within the same model which is the MO business layer model. And, the Realization relationship is specialized to relate a Business Object (from business layer model) to Data Object (from application layer model), and the Used by relationship is specialized to relate a Business Role (from business layer model) to an Application Interface (from application layer model), see. Figure 5.7.

Accordingly, to perform these forms of consistency, several SN Designers must be involved. These designers should cooperate and discuss together to select the proper relationships and elements that enable to establish the interoperability between their different created models. At this stage, the different concerned designers may discover

architectural design errors in the created layers. For example, when the expert designer in the business layer discover that a Smart Sensor is connected to a Data Fusion server through a predefined association relationship, which is not acceptable as a Smart Sensor should be connected to a Data Fusion server through SDR. This may cause an error in the implementation, such as an imprecise length of the marine cable between a Smart Sensor and a Data Fusion server. Another example is when the designers decided to use the Realization relationship presented above in order to relate the business with the application layer, and they find that there is no Business Object in the business layer model, or a Data Object in the application layer model in order to establish the Realization relationship. This makes the interoperability between layers as an impossible operation to be performed by designers.

Therefore, in both examples, the proposed SN design process in chapter 4 should be blocked and re-considered the detected architectural design errors until taking appropriate actions that can unblock this process. These actions can be the possibility of iterating ***Task 1 Modeling*** many times until having no errors (cf. the red arrow issued from "Ensuring Consistency" task to "Modeling" task in Figure 4.1). Furthermore, the before mentioned deals with ***Task 2 Ensuring Consistency*** as the SN designers fix the detected architectural design errors in the model on each iteration according to their different viewpoints, which will minimize the number of detected errors in this task.

In order to validate the defined model that is composed of three separate inter-related models: business, application and technology. This model should be simulated to detect the technical performances errors that may be made by the SN Designers. At this stage, the different concerned designers may discover their architectural design errors in their defined separated models. For example, when the expert designer in the simulator discovers errors in the technical performances as the exchange of messages between terminals is not setup. In this case, the sources of these errors can be from the first two proposed tasks of SN design process, in chapter 4, ***Task 1 Modeling*** and/or ***Task 2 Ensuring Consistency***. Thus, in case the source of technical errors is from ***Task 1 Modeling***, it can be from business layer model and/or application layer model and/or technology layer model. The SN designers may make architectural design errors in the three inter-related models according to their different viewpoints. However, in case the source of technical errors is from ***Task 2 Ensuring Consistency***, it can be from the relationships that are specific to relate business layer model to application layer model, and/or this latter to technology layer model.

Therefore, in the two cases, the proposed SN design process in chapter 4 should be blocked and re-considered the detected technical performances errors until taking appropriate actions. These actions can be the possibility of iterating ***Task 1 Modeling*** many times until having zero error (cf. the red arrow issued from "Validating" task to "Modeling" task in Figure 4.1). And/or the possibility of iterating ***Task 2 Ensuring Consistency*** many times until having zero error (cf. the red arrow issued from "Validating" task to "Ensuring Consistency" task in Figure 4.1).

Furthermore, the before mentioned deals with ***Task 3 Validating*** as the SN de-

signers fix the technical performances errors in the simulated model on each iteration according to their different viewpoints, which will minimize the number of detected errors in this task.

5.4 Discussion

The ArchiMO design tool, and the iteration approach proposed in this chapter contribute towards fulfilling the requirements of SN Design mentioned in Chapter 1:

1. ***Req 1 Improving Architectural Design***: the MO Domain Specific Modeling Languages, and the required and forbidden rules regarding to the SN domain provide a support to the designer of the system architecture. In addition, the iteration between the different process activities reduce the number of the architectural design troubles, as on each iteration the designers improve the model to build a new release in order to surpass the current one.

2. ***Req 2 Multiple Viewpoints***: through the use of Enterprise Architecture, by extending an Enterprise Architecture Modeling Language, ArchiMate. The predefined automatic consistency between the different layers of ArchiMate provides the ability to work independently in a viewpoint, and to define a consistent model. The model includes at the three layers of ArchiMate from different designer's viewpoint, and provides a unified, overall model of business, application and technological views.

3. ***Req 3 Extensibility***: through the metamodel extension, and new design tool generation. This feature allows to add new MO elements and constraints to the design tool. Thus, ArchiMO extends an open, standard, and classical design tool.

4. ***Req 4 Heterogeneity Supported***: by proposing different SN specific Domain Specific Modeling Languages. It provides the possibility of having different components and communication types that are related to our domain. Thus, the deployment of different physical components (Sensors and Servers), and logical components such as acquisition/localization algorithms is supported on a network infrastructure.

5. ***Req 5 Validation Tools Supported***: by adopting existing network simulators. The created models are simulated as early as possible in the life cycle in order to prevent architectural design errors in the deployment phase. The simulation results are interpreted by specialists who formulate recommendations for changing the MO models.

We present the advantages of our contributions below:

- As ArchiMO DSML is based on the existing definition of ArchiMate, we reuse all the concept and relationship definitions of the ArchiMate metamodel, in order to inherit from the context of general purpose information systems. Thereby, this advantage fits ***RQ 2 Modeling Language Specialization***.

- The generated ArchiMO Design tool considers different domains of experience, each domain expert works in his specific layer (Business, Application or Technology) as

a part of an overall model. This design tool prevents syntax and semantics inconsistencies that can be made during the design activity. Also, this design tool ensures the consistency between different layers due to the use of the relationship definition provided by ArchiMate. These advantages enhance and increase the consistency and unity of the design. ArchiMO provides facilities and support for the designer to build a consistent model. Thereby, these advantages fit ***RQ 2 Modeling Language Specialization***.

- Relying on model transformation approach, we can extend Archi tool and generate automatically the ArchiMO design tool that contains the new MO elements, relationships and constraints. This advantage produces an efficient tooling support to achieve the main concerns of the design process. It provides a design tool to support the designer by performing the different tasks of the proposed design process. Thereby, this advantage fits ***RQ 3 Tool Building Process***.

- The SDR extension provides a mapping approach between the different layers of EA to throw directly the needed components and relations to each domain expert. It reduces the time of the design phase for the two lower layers, application and technology. Thereby, this advantage fits ***RQ 2 Modeling Language Specialization***.

- We can rapidly define a DSML and generate its corresponding design tool by extending TOGAF framework and ArchiMate modeling language. This extension allows to integrate in TOGAF new specific concepts, relationships and constraints which are displayed in the palettes of the design tool, and they are ready to be used by the designers. This is an advantage, as in case we adopt MDE to define a DSML and its corresponding design tool, we should perform a long and complex task to create them from the beginning. Thereby, this advantage fits ***RQ 3 Tool Building Process***.

- We inspire from [Chi12] work, then we adopt the same extension approach to define our ArchiMO metamodels as they defined a metamodel for telcommunication domain. In our case, we extend the ArchiMate business layer and application layer metamodels. As we integrate our work in the same TOGAF framework, We inherit in our tooling of the previous extension and of course our MO extensions. The last layer allows us to generate an ArchiMO design tool that contains MO and IMS concepts, relationships and constraints. Thus, by using this generated tool, the SN designers are able to create a consistent model that is composed of different component from the three extended ArchiMate metamodels, and from different relationships. The IMS components and relationships that are in the consistent model, allow the exchange of information between physical components such as terminals. Therefore, our defined MO model in the application layer, is deployed on the technology layer by using its corresponding concepts and relationships in IMS. For example, the connection between a Smart Sensor System and a Data Fusion System component in the application layer is mapped to the technology model on two terminals that exchange messages.

At this stage, we can consider that the work that is performed by [Chi12]

and us is complementary and provide a powerful tooling. This is due to: (1) the returned benefits to both sides, [Chi12] and our side; (2) due to the non negative affection on the work of each other. For our contribution, once the application model is fully represented with the required concepts and relationships in the technology model, the consistent model is ready to be simulated as it satisfies the inputs constraints of the network simulator. However, for [Chi12] contribution, they are validating their extended IMS metamodel by ensuring the reusability of this metamodel in another case study and another domain, the MO. Thus, the utilization of the extended IMS metamodel is not limited for case studies in telecommunication domain.

- [All16] extended the design activity to introduce the early verification activity in its framework (DeVerTeS). They validated this activity by generating the code, then simulate it for a telecommunication service. In addition, [All16] integrates our ArchiMO metamodel in their work to be as another case study for their contribution. At the same time, we exploit their added activity by simulating our MO produced models in order to early validate in the design phase.

  At this stage, we can consider that the work that is performed by [All16] and us is complementary. This is due to: (1) the returned benefits to both sides, [All16] and our side; (2) due to the non negative affection on the work of each other. For our contribution, we are able to perform the proposed ***Task 3 Validating*** in order to early validate our MO defined models by simulating them using the NS-3 network simulator. However, for [All16] contribution, they are validating their extended activity by ensuring its reusability in another case study and another domain which is MO. Thus, the utilization of this extended activity is not limited for case studies in telecommunication domain.

Synthesis

In this chapter, we extended ArchiMate modeling language in order to obtain two new SN specific DSMLs, one for the business layer and another for application layer. For the technology layer, we relied on the extended previously in [CAKR11], an IMS underlying platform. These DSMLs and the iteration approach serve the designer during the design phase by improving architectural design. Next, we generated ArchiMO design tool that contains all the extended MO concepts and restrictions in order to be used by the designers during the design process. An automatic consistency between the different designer's viewpoints and at different abstract levels is proposed, which is provided by ArchiMate that is relying on Enterprise Architecture. This consistency allows to have an overall and consistent model that contains the three created models and viewpoints with inter-relations between them. Then, we proposed to apply the simulation on the created models using the network simulator NS-3. This simulation is executed on models that are created using the ArchiMo design tool in order to test and verify these models as early as possible in the SN life cycle. At the end, we discussed and analyzed the advantages of our contributions. Accordingly, we concluded that, as [Chi12] and [All16] are involved in the same domain and their work is complementary to each other, so they introduced their extensions in the same EA framework, DeVerTeS. And, as our extension is complementary to their work as presented above, so we introduced our extension in DeVerTeS. Therefore, our contributions satisfy and serve two research questions: **RQ 2 Modeling Language Specialization** *and* **RQ 3 Tool Building Process**. *In order to test the usability and the effectiveness of the proposed contributions and especially the ArchiMO generated design tool, a consistent model will be created using DeVerTeS in the next chapter.*

6 Application of the Proposed Sensor Networks Design Process to a Case Study

Reminders and Objectives

In this chapter, we illustrate the benefits of our proposed contributions detailed in the previous chapters. In order to approve and verify the usability and the efficiency of the proposed DSML and design tool (ArchiMO) for Marine Observatory (MO), we use an example of underwater moving objects localization based on Underwater Acoustic Sensor Networks. First, we model the MO under study on ArchiMate Business and Application layers using ArchiMO, and we model it on technology layer relying on the extended IMS metamodel in [Chi12]. Second, we illustrate how the proposed ensuring consistency ensures the interactions between the business and application layers in order to have one consistent model. Then, we validate the consistent model using the network simulator NS-3, in order to support the design phase of the SN life cycle by minimizing the architectural design errors.

6.1 Underwater Object Localization Case Study

Underwater Acoustic Sensor Networks (UW-ASNs) play in MO an essential role, that aims at environmental data acquisition and in the development of the future large data acquisition systems [SBT+08]. Ocean survey requires UW-ASNs, as they allow the data to be exchanged and processed between the different devices such as Data Fusion servers and Smart Sensors. An implementation of a distributed fusion architecture (DFA) [LLL09] is adopted for these devices. On all these devices, we can have software components to process and store the data in order to localize underwater moving objects. Thus, software components could be localization algorithms that are implemented on different specific devices of the UW-ASNs. For example, the trilateration algorithm is implemented on the different Data Fusion servers, and each server require at least three sensors (cf. section 1.3) that require a software component to be implemented on each one. This software component processes the data detected by transforming it to information in order to transfer it to the concerned Data Fusion server.

An example about MO is the project Marine e-Data Observatory Network (MeDON) [**?**]. MeDON is a seafloor observatory information system made of several sensors and a computing system aiming at detecting and localizing marine mammals through Passive Acoustic Monitoring [Zim11]. MeDON is an interesting example for the use of multiple

modeling formalism.

In this context, the designer should be able to include N acoustic sensors or hydrophones that are connected to the Y fusion servers as shown in Figure. 6.1. The acoustic data acquired by the hydrophones, are analyzed and processed by the Smart Sensors and Data Fusion servers, then diffused on the network. A database is used to store the data from the servers. Then, the processed and filtered data are provided by the Database server to the web server where the configuration of a web server is performed. Thus, the web server broadcasts to the web clients, the marine mammals acoustic sounds detected by the hydrophones.

We defined a model for a part of MeDON system that aims at localizing the dolphin underwater using ArchiMO design tool. In order to localize dolphins, we implemented the DFA and the required software components on each device as we presented at the beginning of this section. This defined model is composed of three views according to the layers of ArchiMate, see. Figure 6.2: Business, Application, and Technology. In Figure. 6.2, we present parts of the large model that is created by using ArchiMO design tool. We illustrate the created MO model of each layer, and the relationships between layers, in the next sections.

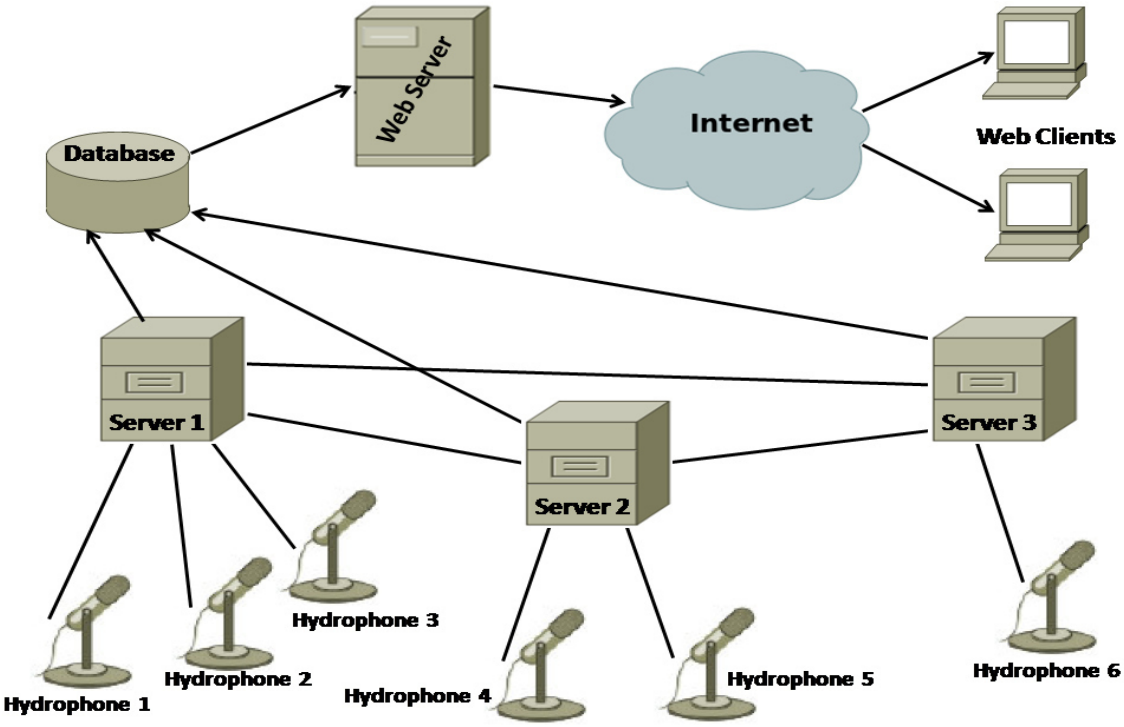

Figure 6.1: Structure of MeDON - An Example: N=6, Y=3

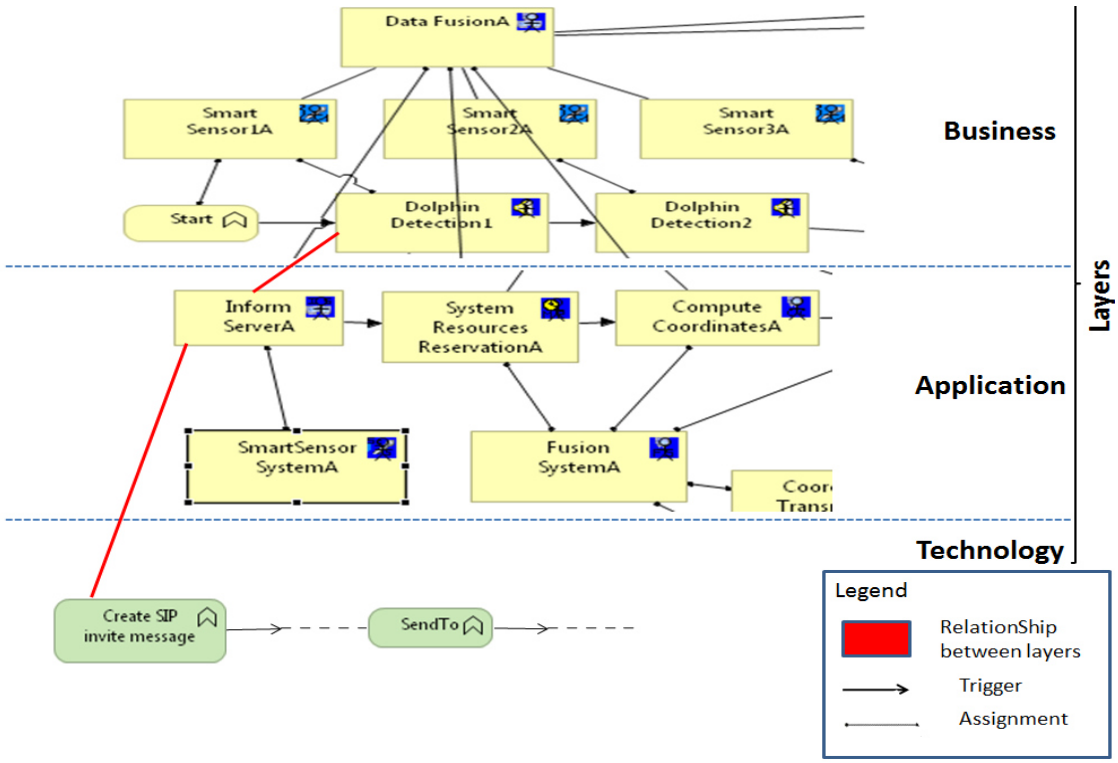

Figure 6.2: Underwater Object Localization according the three ArchiMate Layers

6.2 Modeling a Marine Observatory Case Study using ArchiMO DSML and Design Tool

This section illustrates the use of ArchiMO DSML and Design Tool, an extension of ArchiMate and Archi for MO is proposed in section 5.1.

In order to validate our proposed ArchiMO DSML and Design Tool, we use it to model an underwater object localization application using the new proposed MO concepts, relationships and constraints in section 6.2.

6.2.1 The Business Model Design

The business model aims to present the different required functions and actions to localize Dolphins underwater. As our case MeDON is constituted from six Smart Sensors and three Data Fusion servers, so we present the functions and actions that are performed by these sensors and servers.

The first action is performed by the Smart Sensors, which is the detection of the dolphin. Next, the detected data is transformed to information through the Smart Sensors. Then, this information is transferred to Data Fusion Servers. These servers fused the transferred information and applied the localization algorithm in order to identify the position of the dolphin.

The scenario presented above, is presented in a model that is created in the ArchiMate business layer, see. Figure 6.3. This model contains MO concepts such as all the components and SDR relationship that appear in Figure 6.3, and ArchiMate concepts such as all the other relationships that are established between the different components. This layer indicates that the Dolphin Detection function performs the different detections of Smart Sensors: 1A, 2A, 3A, 1B, 2B, C. These detections are performed according to the selected DFA. Thus, these detection functions are assigned to their proper Smart Sensors, and these Smart Sensors are associated with the different Data Fusion servers. Then, each Data Fusion server is assigned to select the proper Trilateration algorithm to localize the Dolphin. This Trilateration algorithm is executed by Data Fusion servers, and then the information is exchanged among Data Fusion servers through the Data Fusion Transmission to be estimated and determine the coordinates of the Dolphin.

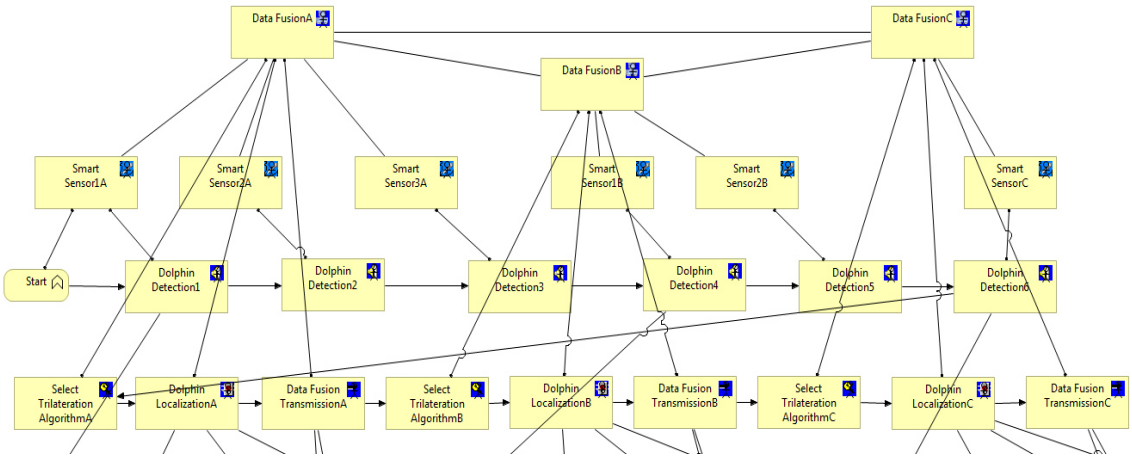

Figure 6.3: Model of a Dolphin Localization that is presented in the ArchiMate Business Layer

6.2.2 The Application Model Design

The application model aims to describe the behaviors of each action that is created in the business layer to localize Dolphins underwater. The first action is performed by the Smart Sensors Systems, which is the Inform Server functions to transfer the information to System Resources Reservation functions. Next, these functions allow the Fusion Systems to perform the next functions which are Compute Coordinates and Store Coordinates. Then, these coordinates are transferred to the next Fusion Systems through the Coordinates Transmission functions.

The scenario presented above, is presented in a model that is created in the ArchiMate application layer, see. Figure 6.4. This model contains MO concepts such as all the components that appear in Figure 6.4, and ArchiMate concepts such as all the other relationships that are established between the different components. This exchange scenario continues until the coordinates are determined, taking into consideration the

information that are detected by the different Smart Sensors.

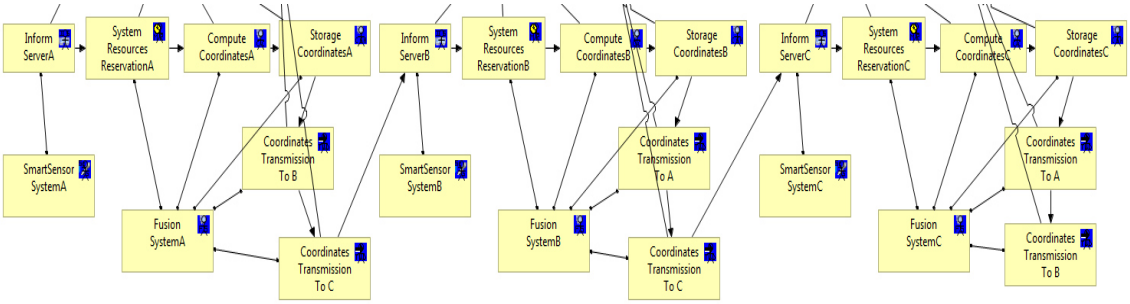

Figure 6.4: Model of a Dolphin Localization that is presented in the ArchiMate Application Layer

6.2.3 The Technology Model Design

The technology model aims to describe the topology of the different existing nodes on the network. Also, it describes the functions that should be performed by each node and between nodes.

According to MeDON [**?**], the number of Smart Sensors and Data Fusion servers is high, and it could be increased according to the needs. For example, when we need to enlarge the area that detects and monitors the dolphin, the number of sensors and servers will increase. This is useful and possible as we are adopting DFA which allows to add sensors and servers as much as it is required. For this purpose, a large number of technology functions are required to ensure the communication between the different nodes, sensors and servers. These types of functions are provided by IMS platform that can be used for different contexts such as MO, Telecommunication.

Accordingly, in this layer we rely on IMS metamodel that is extended by [Chi12]. For example, the Send To function can send or forward messages from a node to another. More specifically, exchanging messages among terminals such as between cameras, hydrophones, smart sensors, data fusion servers. This function extends the ArchiMate technology function in the IMS metamodel [Chi12]. Thus, this model contains IMS concepts such as Send To and Create Invite functions, and ArchiMate concepts such as Nodes and Communication Path functions, see. Figure 6.5.

Consequently, we defined three separate MO models according to the three ArchiMate layers (business, application and technology) by using ArchiMO DSML and Design Tool. Therefore, ArchiMO handles ***Modeling Task 1***.

6.3 Consistency between Model Layers

The MO designer of the business layer should use the extended ArchiMO metamodel of ArchiMate business layer (cf. section 5.1.3.1, Business Layer). Regarding the application

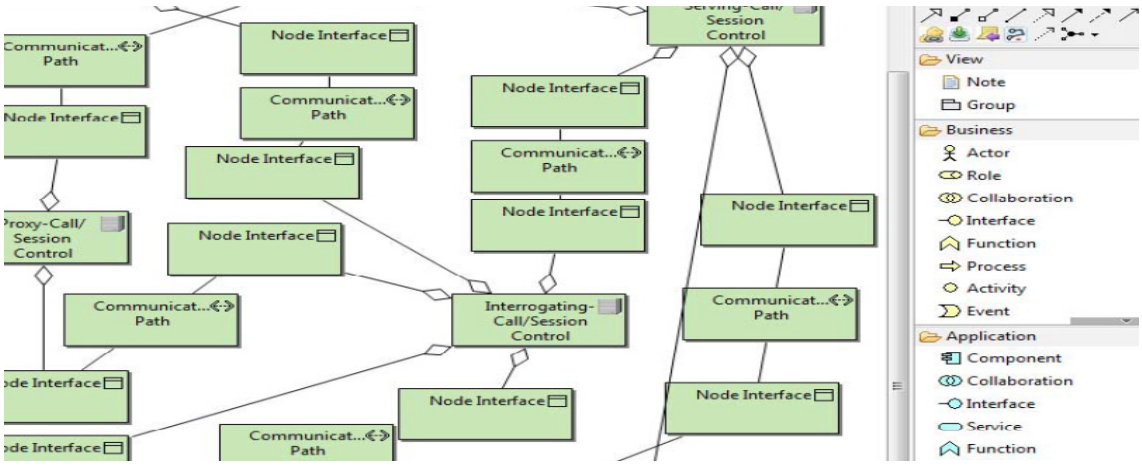

Figure 6.5: Model of a Dolphin Localization that is presented in the ArchiMate Technology Layer, from [All16]

6

layer, the MO designer should use the extended ArchiMO metamodel of ArchiMate application layer (cf. section 5.1.3.1, Application Layer). However, the MO designer of the technology layer should use the extended IMS metamodel of ArchiMate technology in [Chi12].

In order to relate the different created models according to the different ArchiMate layers, this section illustrates the use of specific relationships such as Used by and Realization to ensure the interoperability between layers.

Figure 6.6 presents how the business layer inter-operate with application layer through the Used by relationship. The Dolphin Localization business functions inter-operate with System Resources Reservation, Compute Coordinates and Stores Coordinates applications functions, through the Used by relationship (cf. Figure 5.7, section 3.5.1). Also, Figure 6.2 presents how the business layer inter-operate with application layer and this last inter-operates with technology layer through the Realization relationship. The Dolphin Detections business functions inter-operate with Inform Server application functions, and these latter inter-operate with Create Invite technology functions, through the Realization relationship (cf. Figure 3.8, section 3.5.1). These relationships are presented by the red lines in Figures 6.6 and 6.2. These are examples of the use of the inter-relations between the proposed DSML for business and application layers within MO context.

Consequently, using these inter-relations between the different layers of ArchiMate allow the different concerned designers to have one consistent MO model from their separated created models: business model, application model and technology model. Therefore, these types of relationships handle ***Ensuring Consistency Task 2***.

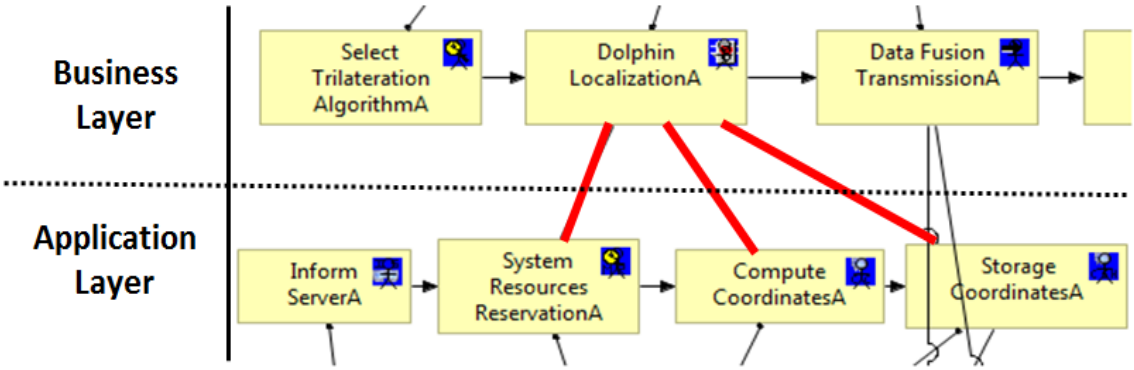

Figure 6.6: Consistency between Business and Application Layers

6.4 Simulation Code

In order to generate a code for simulation, we rely on the code generator that is proposed in [All16] (cf. section 5.2). The proposed ArchiMO design tool generates an XMI file that contains different ArchiMate and ArchiMO concepts and relationships. In order to get as output a generated simulation code, the XMI file is the input for the code generator XPAND.

Once the XMI file is ready to be the input for XPAND, the model transformation rules can be performed. These rules could allow to generate simulation code automatically and could also be integrated in model frameworks to migrate models [VKMB15] or to generate tooling [CBC+16]. This simulation code runs in a tool that is a standard and classical simulator in the networking domain [All16].

Several simulators are compared by [SH11], who points out that the most frequently used network simulators in scientific articles are NS-3 and OPNET. While NS-3 is open source, OPNET is commercial. We select the NS-3 as a network simulator, in order to simulate the MO created models.

NS-3 runs set of classes and functions as we previously presented in (cf. section 5.2). In relation with our MO models, the C++ classes and functions must be MO classes and functions. An example about the MO concepts and functions are illustrated in 6.7. This figure indicates the concepts and functions of MO by black text, and the concepts and functions of NS-3 by blue text and arrows. The Smart Sensor and Data Fusion concepts from the business layer are C++ classes. Also, the Smart Sensor System and Data Fusion System from the application layer are C++ classes. The Algorithm Selection, Data Transmission, Object Localization, Data Acquisition from the buinsess layer are functions in C++ classes. The other MO functions in 6.7 are also functions in C++ classes.

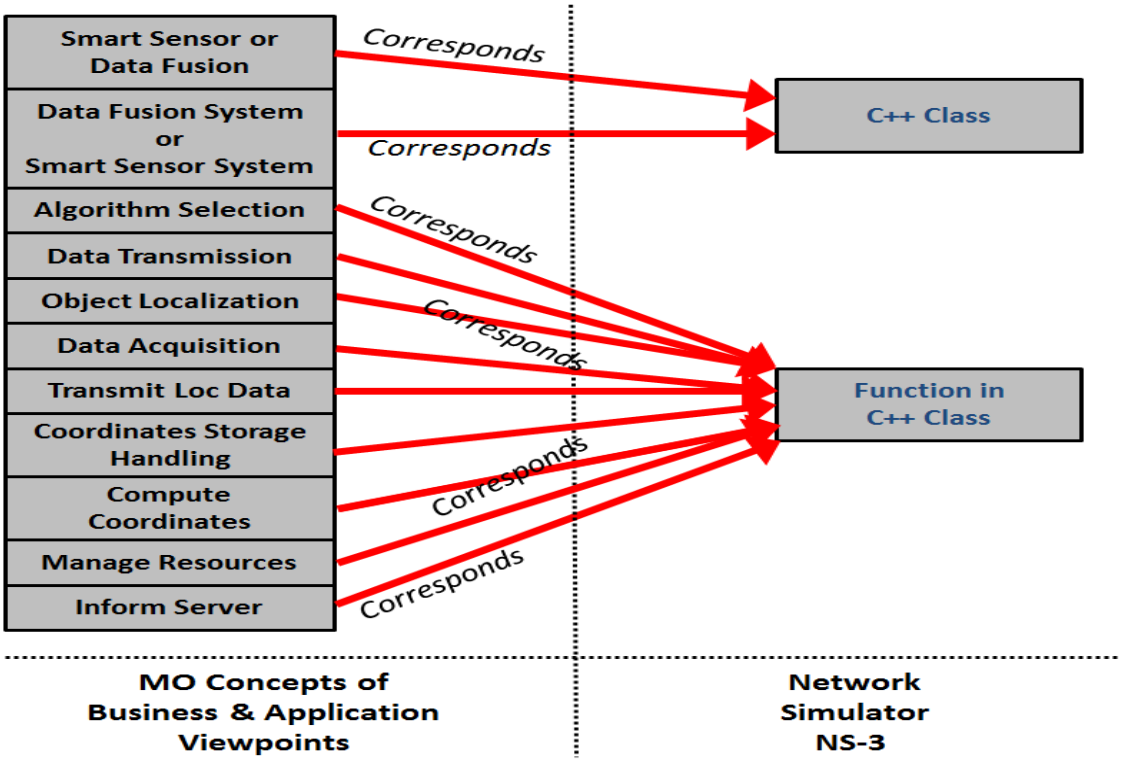

Figure 6.7: The corresponding MO concepts in NS-3

6.5 Validation of Marine Observatory Model

The XPAND code generator provides C++ code that contains the transformed concepts as classes and methods to implement our model [All16], see. Figure 6.7. After that, this model implementation is running through NS-3 platform and then the simulation results are provided in an animation file automatically generated by the NS-3 simulator [BRIH17].

Next, we import this animation file in the NetAnim tool in order to display the results of the simulation. The figure 6.8 is a snapshot from the Net Animator tool that represents the topology of the networks for the underwater object localization case study. More specifically, it represents the hardware concepts of the defined consistent MO model that are required to be defined in the technology layer after the definition of these concepts respectively in the business and application layers. This representation is illustrated by showing the messages that are exchanged between the different existing nodes on the network in Figure 6.8. An example about this exchange in Figure 6.8, is the blue line that relates the SmartSensor1A node to the PCSCF1 node. These messages indicate the traffic between the sensor and the physical node which supports the processing application specify at model level.

This figure 6.7 shows that the simulation results are conformed to the expected results, as we can verify that the main communication links between the different nodes are performed by the number of exchanged messages between the sensor and the first

application node responsible for the first step of the localization algorithm.

So, the obtained simulation results are the main artifacts to validate the application model regarding the deployment on the physical infrastructure. These results close the ***Task 3 Validating***.

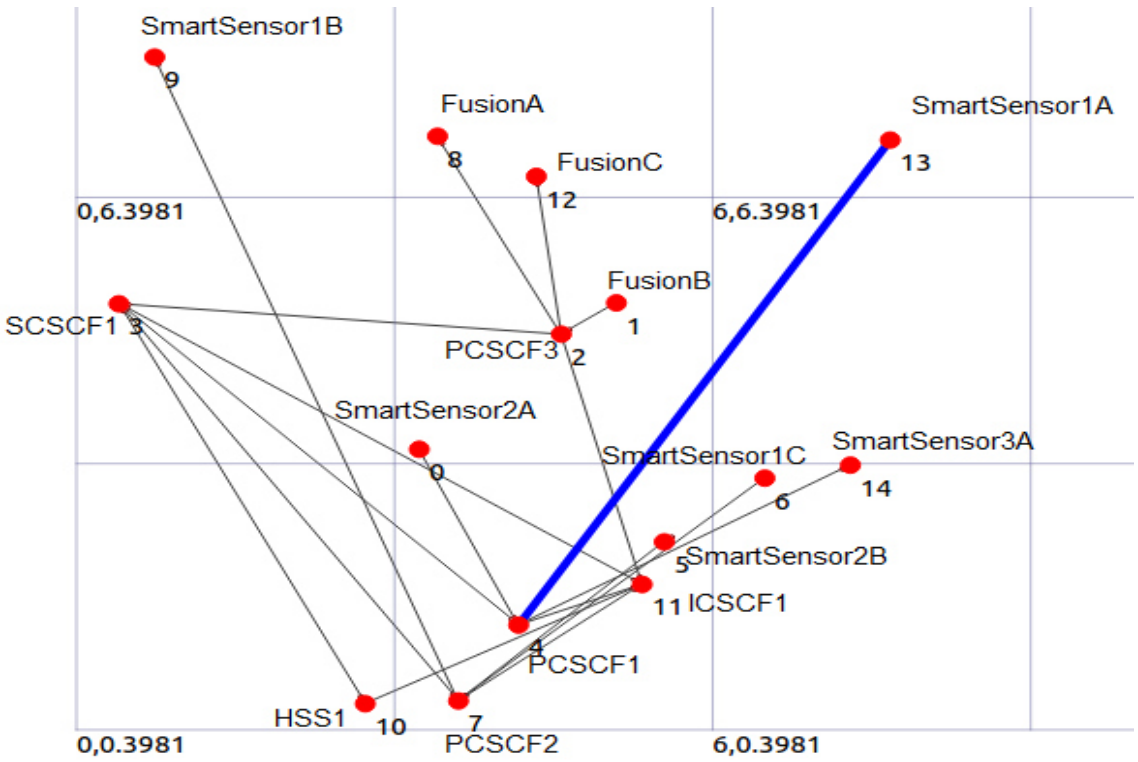

Figure 6.8: A part of the animation through NetAnim tool after running NS-3, from [All16]

6.6 Iteration of the Proposed Sensor Networks Design Process

In order to test the entire proposed SN design process in chapter 4, we still need to show the usefulness of the iterative approach on the different proposed tasks or the whole process (cf. Figure 4.1). This usefulness can be shown only whenever errors are detected while using the model compiler that is developed by [All16]. In addition to the code generation, this model compiler contains an error part. It is used by taking the defined consistent MO model as input, in order to produce a generated simulation code that is ready for simulation (cf. section 5.2). For this purpose, we present an example of errors that can be detected before transforming the MO model that is defined previously in section 7.2 and section 7.3, to a simulation code.

We have selected an example which contains an error just to illustrate the iterative process when we encounter an error. We selected an error in the communication between an interface of a smart sensor and P-CSCF1 node of IMS. This error is related to a conflict in IP addresses. We modify one of the network addresses of the P2P link (see. Figure 6.9). The model compiler generates a log file that contains explanation about this error [All16] (see. Figure 6.9). The following question can be asked here: How can these errors be detected by the model compiler?

In fact, some error detection rules are implemented through the template of XPAND code generator by [All16]. These rules precede the generation rules that are responsible to produce the simulation scenario. This method is responsible to generate a log file that contains the explanation about the detected errors in the design model (see. Figure 6.9). As this figure shows an error, we iterate the concerned task where the error is occurred to fix it and to build a new version of MO model to be simulated. Next, we iterate the validation task on the new version by using the model compiler and we get a result without architectural design errors. However, it is difficult to forecast how many iterations we need, in order to obtain a valid model.

6

Therefore, the cycle based on modeling, generating, simulating and analysing is useful to produce a coherent design model. Next, this model provides the input of the next task of the development process.

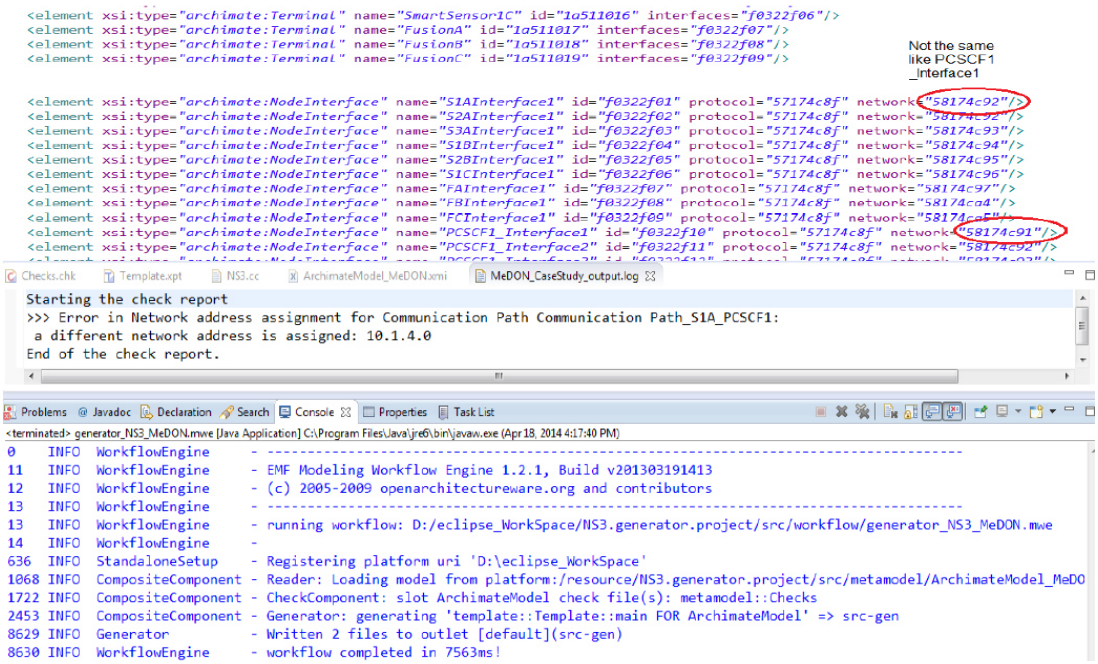

Figure 6.9: An example of an error detection in design model of Marine Observatory system, from [All16]

6.7 Discussion

Relying on the validation result of a Marine Observatory Model, and after ensuring the usefulness of the iterative approach, we conclude that the entire proposed SN design process improves the processes without the use of models and simulation.

Actually, our proposed process helps the SN designer by detecting the architectural design errors, and to fix them by adopting the iteration approach. Our approach is based on two main steps to validate the design task: first, the generated design tool ArchiMO, in chapter 5, helped the SN designers to build validated models by implementing some additional concepts and mainly domain constraints on relationships (cf. section 5.1.6); secondly, the simulation step provides a validation of the system model through the model compiler constraints and mainly the analysis of the simulation results.

So the SN designer has the ability to validate the defined models as early as possible in the adopted SN development life cycle. For this purpose, we keep this kind of simulator to satisfy the ***Requirement 5 Validation Tools Supported*** .

Synthesis

In this chapter, we defined a model for a part of MeDON system that aims at localizing the dolphin underwater using ArchiMO design tool. Then, we defined three separate MO models according to the three ArchiMate layers (business, application and technology) by using the proposed ArchiMO DSML and design tool in chapter 5. Next, we defined the ArchiMate inter-relations between the three defined models, and we built a consistent MO model. Later, we validated this model using the model compiler of [All16], and we obtained result which is the localization of a dolphin is displayed without errors. Then, we fixed a detected communication error between two nodes, by iterating the concerned task in this process. This is to show the usefulness of the iteration approach for our proposed SN design process. Finally, we pointed out the applicability of our SN Design Process as we defined a model for a marine observatory case study.

Conclusion and Perspectives

To conclude this document, we summarize our modeling approach for sensors networks, and we finish by some perspectives related to our work.

Answering the Research Questions

To achieve the observation and monitoring missions, complex operations are needed in the context of Marine Observatory. The sensor networks developed to accomplish these missions is considered as a complex system as it aggregates different types of software and hardware components. Multiple tasks are assigned to each component and different communication protocols are used. To face these challenges, we focus on the designer activities to improve the SN development and deployment phases, and we answer the three research questions listed below:

1. ***RQ 1 Design Process:*** To face the design complexity of SN applications, how can we provide an assistance to guide the design phase?
2. ***RQ 2 Modeling Language Definition Specialization:*** Is it relevant to use a domain specific language to enhance and increase the consistency and unity of the design?
3. ***RQ 3 Tool Building Process:*** How an efficient tooling support can be produced to achieve the main concerns of the design process?

The SN designers should perform several tasks. For each of the proposed tasks, we have proposed an approach based on an iterative design process, a DSML or an existing simulation tool.

To answer ***RQ 1 Design Process***, we discussed several SN design processes, based on Enterprise Architecture frameworks and inspired from software development life cycles. We select an iterative process to provide early validation based on the use of system models. In order to enable the design process, SN designers should perform several tasks. We focused on three of them:

1. ***Task 1 Modeling***: modeling the SN from the stakeholder's point of view: business, application and technology viewpoints.
2. ***Task 2 Ensuring Consistency***: inter-relating the different viewpoints in order to define a consistent model.
3. ***Task 3 Validating***: validating the created model by a simulation approach.

In order to perform our proposed SN design process, we perform the three tasks listed above. This process is enough flexible to enable code generation using model transformations as it relies on model driven engineering (MDE) approach. The inputs of the code

generators tools are models in order to generate executable code as outputs, that is the input code for network simulators.

Once we perform the first two proposed tasks, ***Task 1 Modeling*** and ***Task 2 Ensuring Consistency***, a consistent model is created by the designer. Then, in order to enable the designer to validate his defined model, we generate executable code from model, for network simulators. Simulation helps designers to detect troubles in early phases of software development life cycles. However, simulation is a validation approach that cannot guarantee that there are no errors in the model, but it can detect only architectural design errors.

Once we perform ***Task 3 Validating***, and by adopting the iteration action, several iterations between the three adopted tasks are possible to build a consistent model. On each validation, some architectural design troubles are detected so the designer will re-perform ***Task 1 Modeling*** or ***Task 2 Ensuring Consistency*** or both of them. This iteration can repeat the same tasks until no errors are detected during the validation task.

The validated model is ready now, so the other phases of SN life cycle could be performed, such as the implementation phase. During the SN design phase, the validated models could help the SN designer to avoid the detection of the architectural design errors in later phases of the adopted SN life cycle. Thereby, by ensuring a support for the SN designer as the early validation, it will affect positively the SN design phase by minimizing the architectural design errors that may occur during this phase.

To answer ***RQ 2 Modeling Language Specialization***, we have defined ArchiMO, Domain Specific Modeling Language (DSML) extending an existing Enterprise Architecture Modeling Language, ArchiMate. This extension is done for the different ArchiMate layers: business, application and technology. Also, we proposed to use predefined ArchiMate relationships to inter-relate separate models. This allows the interoperability between models. Then, we introduced this extended DSML into the SN development tool used by the designers.

By introducing DSML into the design tools, designers are able to prevent architectural design errors that can be made during the design phase. It supports ***Task 1 Modeling*** to model the SN from different points of view, and the designers use the new extended concepts and relationships of this DSML. Also, to support the ensuring consistency task ***Task 2 Ensuring Consistency***, the use of predefined relationships that are specific to inter-relate two ArchiMate layers such as Used by and Realization, enable the designers to have one consistent model. Thereby, by ensuring a support for the designer in the modeling and ensuring consistency tasks, it will affect positively the SN design phase by preventing the architectural design errors and having a consistent model.

To answer ***RQ 3 Tool Building Process***, we generated a design tool ArchiMO that contains the extended DSML, ArchiMO metamodel. In order to enable the designer to generate this design tool, we relied on MDE approach. This approach is based on

models, metamodels and model transformations.

By generating ArchiMO design tool, the designers are able to build consistent models using the extended concepts and relationships by dragging and dropping model elements from palettes. This supports ***Task 1 Modeling*** and ***Task 2 Ensuring Consistency*** while applying the new added concepts, relationships and constraints in a less complex way.

We illustrated the proposed design process, domain specific modeling language and design tool, using a marine observatory case study. We presented a defined model for marine observatory showing their different views: business, application, and technology. These models are designed using our extended version of ArchiMO tool that is generated from the ArchiMate metamodel. Our proposed ArchiMO design tool relies completely on the metamodel that forms the origin of the modeling language and is used in every action of this design tool. Then, the resulting consistent model which is simulated using the NS-3 network simulator in order to validate the system model.

Perspectives

In order to develop and improve the results of this work, we suggest the following perspectives:

- We can extend the relevant requirements in the context of MO particularly environmental constraints, new sensors features or new processing node constraints. All these requirements must be taken into account during the development life cycle. And to satisfy these new requirements, we should need to extend the list of the concepts and relationships used to specify the business, the application and the constraints of SN should be made. In this case, we will include in our life cycle iteration of the extension of our MO language, and we will re-apply our methodology from the beginning. Thus, we consider that our ArchiMO metamodel definition should include the life cycle of the MO development.

- More error detection rules should be added to support the designer and facilitate the validation process. This should be done by domain experts who know the restrictions and constraints of the system components. The goal is to analyze what kind of properties that we can verify in the simulator and extend the modeling constraints related to these simulator properties.

Bibliography

[AAK+14] Charbel Geryes Aoun, Iyas Alloush, Yvon Kermarrec, Oussama Kassem Zein, and Joel Champeau. Domain specific modeling language for object localization in marine observatories. In *SENSORCOMM 2014*, 2014. 41, 61

[AAK+15] Charbel Geryes Aoun, Iyas Alloush, Yvon Kermarrec, Joel Champeau, and Oussama Kassem Zein. A modeling approach for marine observatory. *Sensors & Transducers*, 185(2):129, 2015. 61

[AAKR14] Iyas Alloush, Charbel Geryes Aoun, Yvon Kermarrec, and Siegfried Rouvrais. A domain-specific framework for creating early trusted underwater systems relying on enterprise architecture. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 120–125. IEEE, 2014. 41, 61

[AB15] Adel Alshamrani and Abdullah Bahattab. A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model. *International Journal of Computer Science Issues (IJCSI)*, 12(1):106, 2015. 50

[ABB+14] Wolfgang Ahrendt, Bernhard Beckert, Daniel Bruns, Richard Bubel, Christoph Gladisch, Sarah Grebing, Reiner Hähnle, Martin Hentschel, Mihai Herda, Vladimir Klebanov, et al. The key platform for verification and analysis of java programs. In *Verified Software: Theories, Tools and Experiments*, pages 55–71. Springer, 2014. 59

[AGF05] M Azmoodeh, N Georgalas, and S Fisher. Model-driven systems development and integration environment. *BT Technology Journal*, 23(3):96–110, 2005. 28

[AHS06] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Global sensor networks. Technical report, 2006. 41

[AKR13] Iyas Alloush, Yvon Kermarrec, and Siegfried Rouvrais. A generalized model transformation approach to link design models to network simulators-ns-3 case study. In *SIMULTECH*, pages 337–344. 2013. xvii, 41

[Ale12] Paulo Alencar. *Handbook of Research on Mobile Software Engineering: Design, Implementation, and Emergent Applications: Design, Implementation, and Emergent Applications*. IGI Global, 2012. 53

[All16] Iyas Alloush. *DeVerTeS: A Design and Verification Framework for Telecommunication Services*. PhD thesis, Télécom Bretagne, Université de Bretagne-Sud, 2016. xii, 42, 44, 47, 49, 64, 80, 86, 87, 94, 95, 96, 97, 98, 99

[And13] Cyrille André. *Approche crédibiliste pour la fusion multi capteurs décentralisée*. PhD thesis, Université Paris Sud-Paris XI, 2013. 10, 12

[AR07] Peter Alriksson and Anders Rantzer. Experimental evaluation of a distributed kalman filter algorithm. In *Decision and Control, 2007 46th IEEE Conference on*, pages 5499–5504. IEEE, 2007. 12

[ASMT+12] Oleksandr Artemenko, Tobias Simon, Andreas Mitschele-Thiel, Dominik Schulz, and Rheza Satria Ta. Comparison of anchor selection algorithms for improvement of position estimation during the wi-fi localization process in disaster scenario. In *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*, pages 44–49. IEEE, 2012. 9

[AYG10] Achilleas Achilleos, Kun Yang, and Nektarios Georgalas. Context modelling and a context-aware framework for pervasive service creation: A model-driven approach. *Pervasive and Mobile Computing*, 6(2):281–296, 2010. 14, 15, 28, 41, 43

[BBD+00] Eerke Boiten, Howard Bowman, John Derrick, Peter Linington, and Maarten Steen. Viewpoint consistency in odp. *Computer Networks*, 34(3):503–537, 2000. 25

[BBDS99] Howard Bowman, Eerke A. Boiten, John Derrick, and MWA Steen. Strategies for consistency checking based on unification. *Science of Computer Programming*, 33(3):261–298, 1999. 25

[BBL12] Stefan Bente, Uwe Bombosch, and Shailendra Langade. *Collaborative Enterprise Architecture: Enriching EA with lean, agile, and enterprise 2.0 practices*. Newnes, 2012. 33

[BDF+05] Ramón Béjar, Carmel Domshlak, Cèsar Fernández, Carla Gomes, Bhaskar Krishnamachari, Bart Selman, and Magda Valls. Sensor networks and distributed csp: communication, computation and complexity. *Artificial Intelligence*, 161(1):117–147, 2005. xv

[Béz04] Jean Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004. 22

[BJ02] J.-L. Bakker and R. Jain. Next generation service creation using xml scripting languages. *ICC 2002. IEEE International Conference on*, 4:2001–2007, 2002. 41

[BJKV06] Jean Bézivin, Frédéric Jouault, Ivan Kurtev, and Patrick Valduriez. Model-based dsl frameworks. In *Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA*, pages 22–26. Citeseer, 2006. 25, 75

[BONL08] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo F Nakamura, and Antonio AF Loureiro. Vehicular ad hoc networks: A new challenge for localization-based systems. *Computer communications*, 31(12):2838–2849, 2008. 8

[BRIH17] Amel Berrachedi, Messaoud Rahim, Malika Ioualalen, and Ahmed Hammad. Validation of a sysml based design for wireless sensor networks. In *AIP Conference Proceedings*, volume 1863, page 330002. AIP Publishing, 2017. 96

[BS08] Pruet Boonma and Junichi Suzuki. Middleware support for pluggable non-functional properties in wireless sensor networks. In *Services-Part I, 2008. IEEE Congress on*, pages 360–367. IEEE, 2008. 28

[BS10] Pruet Boonma and Junichi Suzuki. Moppet: A model-driven performance engineering framework for wireless sensor networks. *The Computer Journal*, page bxp129, 2010. 28

[BT10] Kai Beckmann and Marcus Thoss. A model-driven software development approach using omg dds for wireless sensor networks. In *Software Technologies for Embedded and Ubiquitous Systems*, pages 95–106. Springer, 2010. 28

[Bur14] Erik Burger. *Flexible Views for View-based Model-driven Development*, volume 15. KIT Scientific Publishing, 2014. 34

[CAKR11] Vanea Chiprianov, Iyas Alloush, Yvon Kermarrec, and Siegfried Rouvrais. Telecommunications service creation: Towards extensions for enterprise architecture modeling languages. In *6th Intl. Conf. on Software and Data Technologies (ICSOFT)*, volume 1, pages 23–29, Seville, Spain, 2011. 17, 76, 87

[Cas13] Federico Castanedo. A review of data fusion techniques. *The Scientific World Journal*, 2013, 2013. 9

BIBLIOGRAPHY

[CBC+16] Benoit Combemale, Cédric Brun, Joël Champeau, Xavier Crégut, Julien Deantoni, and Jérome Le Noir. A tool-supported approach for concurrent execution of heterogeneous models. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016. 95

[CCAN14] Srimathi Chandrasekaran, Eunmi Choi, Jemal H Abawajy, and Rajesh Natarajan. Sensor grid middleware metamodeling and analysis. *International Journal of Distributed Sensor Networks*, 2014, 2014. 34

[CDV08] David Chen, Guy Doumeingts, and François Vernadat. Architectures for enterprise integration and interoperability: Past, present and future. *Computers in industry*, 59(7):647–659, 2008. 32

[CEKS01] Tony Clark, Andy Evans, Stuart Kent, and Paul Sammut. The mmf approach to engineering object-oriented design languages. 2001. 22, 24

[CF01] Paul G Carlock and Robert E Fenton. System of systems (sos) enterprise systems engineering for information-intensive organizations. *Systems engineering*, 4(4):242–261, 2001. 4

[CGA15] Yvon kermarrec Joel Champeau Oussama Kassem Zein Charbel Geryes Aoun, Iyas Alloush. A mapping approach for marine observatory relying on enterprise architecture. In *OCEANS, 2015 IEEE*. IEEE, 2015. 61

[CGLP05] Andrea Caiti, Andrea Garulli, Flavio Livide, and Domenico Prattichizzo. Localization of autonomous underwater vehicles by floating acoustic buoys: a set-membership approach. *Oceanic Engineering, IEEE Journal of*, 30(1):140–152, 2005. 8, 9

[CGS12] Hyun Cho, Jeff Gray, and Eugene Syriani. Creating visual domain-specific modeling languages from end-user demonstration. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, pages 22–28. IEEE Press, 2012. 37, 61

[Chi12] Vanea Chiprianov. *Collaborative construction of telecommunications services. An enterprise architecture and model driven engineering method*. PhD thesis, Télécom Bretagne, Université de Bretagne-Sud, 2012. 28, 32, 34, 35, 37, 47, 64, 74, 85, 86, 87, 89, 93, 94

[CKR12] Vanea Chiprianov, Yvon Kermarrec, and Siegfried Rouvrais. Extending enterprise architecture modeling languages: Application to telecommunications service creation. In *The 27th Symposium On Applied Computing*, pages 21–24, Trento, 2012. ACM. 17, 35, 44, 71

[CKRS14] Vanea Chiprianov, Yvon Kermarrec, Siegfried Rouvrais, and Jacques Simonin. Extending enterprise architecture modeling languages for domain specificity and collaboration: application to telecommunication service design. *Software & Systems Modeling*, 13(3):963–974, 2014. 35, 59, 75

[CL10] Byoung-Suk Choi and Ju-Jang Lee. Sensor network based localization algorithm using fusion sensor-agent for indoor service robot. *IEEE Transactions on Consumer Electronics*, 56(3), 2010. 8

[CM98] Charles Consel and Renaud Marlet. Architecture software using: a methodology for language development. In *Principles of Declarative Programming*, pages 170–194. Springer, 1998. 26

[CM05] C-Y Chong and Shozo Mori. Distributed fusion and communication management for target identification. In *Information Fusion, 2005 8th International Conference on*, volume 2, pages 8–pp. IEEE, 2005. 10, 12, 13

[CM13] Brian H Cameron and Eric McMillan. Analyzing the current trends in enterprise architecture frameworks. *Journal of Enterprise Architecture*, 9(1):60–71, 2013. 32

[CMSD04] Eric Cariou, Raphaël Marvie, Lionel Seinturier, and Laurence Duchien. Ocl for the specification of model transformation contracts. In *OCL and Model Driven Engineering, UML 2004 Conference Workshop*, volume 12, pages 69–83, 2004. 67

[Com] European Commission. *Stakeholders Consultation Network Technologies Work Programme 2016-2017.* https://ec.europa.eu/digital-agenda/en/news/stakeholders-consultation-workshop-network-technologies-work-programme-2016-2017. xvi

[CT07] Jordi Cabot and Ernest Teniente. Transformation techniques for ocl constraints. *Science of Computer Programming*, 68(3):179–195, 2007. 24, 67

[Cuz09] Alfredo Cuzzocrea. *Intelligent techniques for warehousing and mining sensor network data.* IGI Global, 2009. xiv, xv

[CWXG14] Nengcheng Chen, Ke Wang, Changjiang Xiao, and Jianya Gong. A heterogeneous sensor web node meta-model for the management of a flood monitoring system. *Environmental Modelling & Software*, 54:222–237, 2014. 34

[CZ15] Joel Champeau and Oussama Kassem Zein. A modeling approach for marine observatory. 2015. 41

[DBST10] Zekai Demirezen, Barrett R Bryant, Anthony Skjellum, and Murat M Tanik. Design space analysis in model-driven engineering. *Journal of Integrated Design and Process Science*, 14(1):1, 2010. 61

[DNR03] Dionisio De Niz and Raj Rajkumar. Time weaver: a software-through-models framework for embedded real-time systems. In *ACM SIGPLAN Notices*, volume 38, pages 133–143. ACM, 2003. 13

[Ecl15] Eclipse Foundation: Eclipse Modeling Framework (EMF). *http://www.eclipse.org/modeling/emf/*, Last visited 27-September-2015. 27

[EG10] James Brusey Michael Allen Geoffrey Challen Elena Gaura, Lewis Girod. *Wireless Sensor Networks: Deployments and Design Frameworks.* Springer, 2010. 5

[EIS] KCAUW EISENECKER. Generative programming-methods, tools, and applications. 2000. *Disponible en.* 22

[EKW92] David W Embley, Barry D Kurtz, and Scott N Woodfield. *Object-oriented systems analysis: a model-driven approach.* Yourdon Press Englewood Cliffs, NJ, 1992. 67

[EVG07a] Melike Erol, Luiz FM Vieira, and Mario Gerla. Auv-aided localization for underwater sensor networks. In *Wireless Algorithms, Systems and Applications, 2007. WASA 2007. International Conference on*, pages 44–54. IEEE, 2007. xv

[EVG07b] Melike Erol, Luiz FM Vieira, and Mario Gerla. Auv-aided localization for underwater sensor networks. In *Wireless Algorithms, Systems and Applications, 2007. WASA 2007. International Conference on*, pages 44–54. IEEE, 2007. 9

[EZ14] Nicole El Zoghby. *Fusion distribuée de données échangées dans un réseau de véhicules.* PhD thesis, Université de Technologie de Compiègne, 2014. 10

[FBK+11] Alexander Funk, Claas Busemann, Christian Kuka, Susanne Boll, and Daniela Nicklas. Open sensor platforms: The sensor web enablement framework and beyond. In *MMS*, pages 39–52, 2011. 41

[FDFL+14] Giancarlo Fortino, Giuseppe Di Fatta, Wenfeng Li, Sergio F Ochoa, Alfredo Cuzzocrea, and Mukaddim Pathan. *Internet and Distributed Computing Systems: 7th International Conference, IDCS 2014, Calabria, Italy, September 22-24, 2014, Proceedings*, volume 8729. Springer, 2014. 34

BIBLIOGRAPHY

[FIFF15] Alberto Fernández-Isabel and Rubén Fuentes-Fernández. Analysis of intelligent transportation systems using model-driven simulations. *Sensors*, 15(6):14116–14141, 2015. 14, 15, 17

[FMSB11] Franck Fleurey, Brice Morin, Arnor Solberg, and Olivier Barais. Mde to manage communications with and between resource-constrained systems. In *International Conference on Model Driven Engineering Languages and Systems*, pages 349–363. Springer, 2011. 34

[Fon07] Frédéric Fondement. *Concrete syntax definition for modeling languages*. PhD thesis, Citeseer, 2007. 35, 37, 40

[For99] IFIP-IFAC Task Force. Geram: Generalised enterprise reference architecture and methodology. *IFIP-IFAC Task Force on Architectures for Enterprise Integration March Version*, 1(3), 1999. 32

[FR07] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007. 22, 23

[Fra13] The Open Group Architecture Framwork. Integrating togaf and the banking industry architecture network (bian) service landscape whitepaper. In *Open Group Enterprise Architecture Conference London.*, 2013. 42

[GBN10] Dawud Gordon, Michael Beigl, and Martin Alexander Neumann. dinam: A wireless sensor network concept and platform for rapid development. In *Networked Sensing Systems (INSS), 2010 Seventh International Conference on*, pages 57–60. IEEE, 2010. 4, 5

[GGB+10] Elena Gaura, Lewis Girod, James Brusey, Michael Allen, and Geoffrey Challen. *Wireless sensor networks: Deployments and design frameworks*. Springer Science & Business Media, 2010. 5

[GK03] Jeff Gray and Gábor Karsai. An examination of dsls for concisely representing model traversals and transformations. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2003. 24

[GK11] Janis Grabis and Marite Kirikova. *Perspectives in Business Informatics Research: 10th International Conference, BIR 2011, Riga, Latvia, October 6-8, 2011, Proceedings*, volume 90. Springer, 2011. 41, 42

[GKH07] Dragan Gašević, Nima Kaviani, and Marek Hatala. On metamodeling in megamodels. In *Model Driven Engineering Languages and Systems*, pages 91–105. Springer, 2007. 22

[GR04] Martin Große-Rhode. On model integration and integration modelling. In *Integration of Software Specification Techniques for Applications in Engineering*, pages 567–581. Springer, 2004. 24, 25

[Gro09] The Open Group. Archimate 1.0 specification. In *xxix, 25, 27, 29, 30, 31, 32, 33, 34, 35, 78, 98*, 2009. xi, 33, 35, 36, 37, 38, 41, 49

[Hau14] Michael Hauck. *Automated Experiments for Deriving Performance-relevant Properties of Software Execution Environments*, volume 13. KIT Scientific Publishing, 2014. 34

[HJS+12] Guangjie Han, Jinfang Jiang, Lei Shu, Yongjun Xu, and Feng Wang. Localization algorithms of underwater wireless sensor networks: A survey. *Sensors*, 12(2):2026–2061, 2012. 9

[HLS+05] John Heidemann, Yuan Li, Affan Syed, Jack Wills, and Wei Ye. Underwater sensor networking: Research challenges and potential applications. *Proceedings of the Technical Report ISI-TR-2005-603, USC/Information Sciences Institute*, 2005. xiv

[HML02] Andreas Hoffmann, Heinrich Meyr, and Rainer Leupers. *Architecture exploration for embedded processors with LISA*. Springer, 2002. 13

[Hot13] Asilomar Hotel. Forty-seventh asilomar conference on signals, systems, and computers. 2013. 8

[HR04] David Harel and Bernhard Rumpe. Meaningful modeling: what's the semantics of" semantics"? *Computer*, 37(10):64–72, 2004. 24

[HSM10] Kenn Hussey, Bran Selic, and Toby McClean. An extended survey of open source model-based engineering tools. Technical report, Revision E, 2010. 27

[HSZ12] John Heidemann, Milica Stojanovic, and Michele Zorzi. Underwater sensor networks: applications, advances and challenges. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 370(1958):158–175, 2012. xv

[Hur14] Jim Hurst. Comparing software development life cycles. *SANNS Software Security*, 2014. 51

[HYW+06] John Heidemann, Wei Ye, Jack Wills, Affan Syed, and Yuan Li. Research challenges and applications for underwater sensor networking. In *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE*, volume 1, pages 228–235. IEEE, 2006. xiv

[Ini12] Krzysztof Iniewski. *Optical, acoustic, magnetic, and mechanical sensor technologies*. CRC Press, 2012. 8

[Jac94] Michael Jackson. Software development method. *A Classical Mind: Essays in Honour of CAR Hoare; AW Roscoe ed*, pages 211–230, 1994. 50, 51

[Jam08] Mo Jamshidi. *Systems of systems engineering: principles and applications*. CRC press, 2008. 3, 4

[JE14] Andrew Josey and Bill Estrem. *ArchiMate® 2 Certification Study Guide*. Van Haren, 2014. 32

[JLVB+04] Henk Jonkers, Marc Lankhorst, Rene Van Buuren, Stijn Hoppenbrouwers, Marcello Bonsangue, and Leendert Van Der Torre. Concepts for modeling enterprise architectures. *International Journal of Cooperative Information Systems*, 13(03):257–287, 2004. 35

[KBJV06] Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. Model-based dsl frameworks. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 602–616. ACM, 2006. 22, 24

[KC90] Barbara Kitchenhani and Roland Carn. Research and practice: software design methods and tools. 1990. 41

[KG12] Gourav Khurana and Sachin Gupta. Study and comparison of software development life cycle models. *IJREAS*, 2(2):1–9, 2012. 51

[KGC] Deepak Khosla, James Guillochon, and Howard Choe. Distributed fusion and tracking in multi-sensor systems. 10, 12, 13

[KH05] Elliott Kaplan and Christopher Hegarty. *Understanding GPS: principles and applications*. Artech house, 2005. 8, 10, 62

[Kim07] Duk-Hyun Kim. Towards an architecture modeling language for networked organizations. In *Establishing the Foundation of Collaborative Networks*, pages 309–316. Springer, 2007. 35

BIBLIOGRAPHY

[KK03] Mika Katara and Shmuel Katz. Architectural views of aspects. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 1–10. ACM, 2003. 6

[KK07] Mika Katara and Shmuel Katz. A concern architecture view for aspect-oriented software design. *Software & Systems Modeling*, 6(3):247–265, 2007. 6

[KMB+96] Richard B Kieburtz, Laura McKinney, Jeffrey M Bell, James Hook, Alex Kotov, Jeffrey Lewis, Dino P Oliva, Tim Sheard, Ira Smith, and Lisa Walton. A software engineering experiment in software component generation. In *Proceedings of the 18th international conference on Software engineering*, pages 542–552. IEEE Computer Society, 1996. 24

[KRV07] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Integrated definition of abstract and concrete syntax for textual languages. In *Model Driven Engineering Languages and Systems*, pages 286–300. Springer, 2007. 22, 24

[KT08] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008. 34

[Lan09] G. Berrisford & M. Lankhorst. Using archimate with togaf-part 1: Answers to nine general questions about methods. In *Via Nova Architectura, https://doc.novay.nl/dsweb/Get/Document-101474*, 2009. 40

[LCD+14] J Loicq, L Clermont, W Dierckxb, T Van Achteren, and Y Stockmana. A 100 m ground resolution global daily coverage earth observation mission. In *International Conference on Space Optics*, volume 7, page 10, 2014. 8, 10, 12

[LCK+97] Martin E Liggins, Chee-Yee Chong, Ivan Kadar, Mark G Alford, Vincent Vannicola, Stelios Thomopoulos, et al. Distributed fusion architectures and algorithms for target tracking. *Proceedings of the IEEE*, 85(1):95–107, 1997. 10, 12, 13

[Lew05] Raymond Lewallen. Software development life cycle models. *Codebetter. com. Available online at www. codebetter. com/blogs/raymond. lewallen/archive/2005/07/13/129114. aspx*, 2005. 50

[LGS+05] Christl Lauterbach, Rupert Glaser, Domnic Savio, Markus Schnell, Werner Weber, Susanne Kornely, and Annelie Stöhr. A self-organizing and fault-tolerant wired peer-to-peer sensor network for textile applications. In *Engineering Self-Organising Systems*, pages 256–266. Springer, 2005. xiii

[LIHL17] Martin Liggins II, David Hall, and James Llinas. *Handbook of multisensor data fusion: theory and practice*. CRC press, 2017. 10

[LKS08] Jaehan Lee, Jangsub Kim, and Erchin Serpedin. Clock offset estimation in wireless sensor networks using bootstrap bias correction. In *Wireless Algorithms, Systems, and Applications*, pages 322–329. Springer, 2008. xiv

[LLL09] Martin E. Liggins, David L.Hall, and James Llinas. *Multisensor Data Fusion, Theory and Practice*. Taylor & Francis Group, LLC, 2009. 10, 12, 61, 62, 71, 75, 89

[Luo13] ZongWei Luo. *Technological Solutions for Modern Logistics and Supply Chain Management*. IGI Global, 2013. 41

[LVCÁ+07] Fernando Losilla, Cristina Vicente-Chicote, Bárbara Álvarez, Andrés Iborra, and Pedro Sánchez. Wireless sensor network application development: An architecture-centric mde approach. In *Software Architecture*, pages 179–194. Springer, 2007. 28

[LWW11] Xiang-Yang Li, Yajun Wang, and Yu Wang. Complexity of data collection, aggregation, and selection for wireless sensor networks. *Computers, IEEE Transactions on*, 60(3):386–399, 2011. xv

[Mar08] Slaviša Markovic. *Model refactoring using transformations*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2008. 24, 68

[MFHH02] Samuel Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002. 13

[MG10] Nabil Mohammed Ali Munassar and A Govardhan. A comparison between five models of software engineering. *IJCSI*, 5:95–101, 2010. 51

[MHO11] Martin Meyer, Markus Helfert, and Conor O'Brien. An analysis of enterprise architecture maturity frameworks. In *Perspectives in Business Informatics Research*, pages 167–177. Springer, 2011. 42

[Mil07] Kevin L Mills. A brief survey of self-organization in wireless sensor networks. *Wireless Communications and Mobile Computing*, 7(7):823–834, 2007. 7, 8

[Mit04] Nikolas Mitrou. *Networking 2004: Networking Technologies, Services, and Protocols; Performance of Computer and Communications Networks; Mobile and Wireless Communications; Third International IFIP-TC6 Networking Conference, Athens, Greece, May 9-14, 2004; Proceedings*, volume 3042. Springer Science & Business Media, 2004. xiv, xv, 9

[Mit07] HB Mitchell. Introduction. In *Multi-Sensor Data Fusion*, pages 3–13. Springer, 2007. 10

[MM07] Wassim Masri and Zoubir Mammeri. Middleware for wireless sensor networks: A comparative analysis. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 349–356. IEEE, 2007. 28

[MMD15] P Maurya, R Madhan, and E Desa. Potential of autonomous underwater vehicles as new generation ocean data platforms. Indian Academy of Sciences, 2015. 8

[MRI12] Marjan Moradi, Javad Rezazadeh, and Abdul Samad Ismail. A reverse localization scheme for underwater acoustic sensor networks. *Sensors*, 12(4):4352–4380, 2012. xv

[MSB11] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011. 59

[MSDW01] Somajyoti Majumder, Steve Scheding, and Hugh F Durrant-Whyte. Multisensor data fusion for underwater navigation. *Robotics and Autonomous Systems*, 35(2):97–108, 2001. 71, 75

[MT11] Amsterdam Cape Town Dubai London Madrid and Milan Munich Paris Montreal Toronto. Ian sommerville. 2011. 24, 34

[NAS15] NASA. *Assurance Plan for Complex Electronics: Assurance Process: Verification and Validation*, 15-Oct-2015. `http://www.hq.nasa.gov/office/codeq/software/ComplexElectronics/p_vv.htm`. 59

[Nor03] Ovidiu Noran. An analysis of the zachman framework for enterprise architecture from the geram perspective. *Annual Reviews in Control*, 27(2):163–183, 2003. 35, 37

[OLPW+08] Martin Op't Land, Erik Proper, Maarten Waage, Jeroen Cloo, and Claudia Steghuis. *Enterprise architecture: creating value by informed governance*. Springer Science & Business Media, 2008. 32

[OMG03] OMG. Mda guide version 1.0.1. In *http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf*, June 2003. 27

[OMG10] OMG. Meta object facility (mof) core specification. In *V. 2.4*, 2010. 27

[OMG15] OMG MOF OMG Meta Object Facility Specification v1.4, OMG Document formal. *http://www.omg.org*, Last visited 24-September-2015. 22

BIBLIOGRAPHY

[oS07] National University of Singapore. http://pat.comp.nus.edu.sg. In *PAT: Process Analaysis Toolkit*, 2007. xi, 40, 41, 42

[Pao94] Lucy Pao. Distributed multisensor fusion. In *Guidance, Navigation, and Control Conference*, page 3549, 1994. 10, 12

[Par12] Fernando S Parreiras. *Semantic Web and Model-Driven Engineering*. John Wiley & Sons, 2012. 21, 22, 25, 80

[PGSP11] Juan Pavón, Jorge Gómez-Sanz, and Adolfo López Paredes. The sicossys approach to sos engineering. In *System of systems engineering (SoSE), 2011 6th international conference on*, pages 179–184. IEEE, 2011. 15

[PMDC+07] Jorge-Luis Pérez-Medina, Sophie Dupuy-Chessa, et al. A survey of model driven engineering tools for user interface design. In *Task Models and Diagrams for User Interface Design*, pages 84–97. Springer, 2007. 22, 27, 35, 38

[QEJVS09] Dick Quartel, Wilco Engelsman, Henk Jonkers, and Marten Van Sinderen. A goal-oriented requirements modelling language for enterprise architecture. In *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, pages 3–13. IEEE, 2009. 35, 37

[Ras] Vanshika Rastogi. Software development life cycle models-comparison, consequences. 50

[RBR10] Anthony G Rowe, Gaurav Bhatia, and Raj Rajkumar. A model-based design approach for wireless sensor-actuator networks. 2010. 13, 14, 16, 17

[Rec08] Jörg Rech. *Model-Driven Software Development: Integrating Quality Assurance: Integrating Quality Assurance*. IGI Global, 2008. 23

[Ree15] Brooks L Reed. Controller design for underwater vehicle systems with communication constraints. Master's thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution, 2015. xiv

[RGK+11] GT Raju, DK Ghosh, T Satish Kumar, S Kavyashree, and V Nagaveni. Wireless sensor network lifetime optimization. 2011. 10

[RKM02] Kay Römer, Oliver Kasten, and Friedemann Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):59–61, 2002. 28

[Roo08] Tanya Gazelle Roosta. *Attacks and defenses of ubiquitous sensor networks*. ProQuest, 2008. xiv, xv

[RP08] Raman Ramsin and Richard F Paige. Process-centered review of object oriented software development methodologies. *ACM Computing Surveys (CSUR)*, 40(1):3, 2008. 50

[RW11] Nick Rozanski and Eowin Woods. Applying viewpoints and views to software architecture. *Whitepaper, http://www. viewpoints-andperspectives. info/vpandp/wpcontent/themes/secondedition/doc/VPandV_WhitePaper. pdf (accessed 2012-05-23)*, 2011. xvi, 6

[RWO03] William Robinson, Gisele Welch, and Gary O'Neill. The need for a systems engineering approach for measuring and predicting the degradation of aging systems and how it can be achieved. Technical report, DTIC Document, 2003. 4

[RZ00] Zebhauser Rothacher and Benedikt Zebhauser. Einführung in gps. *Tutorial zum*, 3, 2000. 9

[SBT+08] J Sorribas, A Barba, E Trullols, A Manuel, M de la Muela, et al. Marine sensor networks and ocean observatories. a policy based management approach. In *The Third International Multi-Conference on Computing in the Global Information Technology*, pages 143–147. IEEE, 2008. 89

[Sch06] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006. 28

[SCK11] Jean-Philippe Schneider, Joël Champeau, and Dominique Kerjean. Domain-specific modelling applied to inteegration of smart sensors into an information system. In *ICEIS 2011*, page XX, 2011. xv

[SGV+06] Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, Carlos Ferraz, and Judith Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, 10(1):37–44, 2006. 28

[SH11] Nurul I Sarkar and Syafnidar A Halim. A review of simulation of telecommunication networks: simulators, classification, comparison, methodologies, and recommendations. 2011. 95

[Sha06] Ali Fatolahi & Fereidoon Shams. An investigation into applying uml to the zachman framework. *Information Systems Frontiers*, 8:133–143, 2006. 32

[SHL12] Xianwei Sun, Scott C-H Huang, and Minming Li. Lower bounds on data collection time in sensor networks. In *Wireless Algorithms, Systems, and Applications*, pages 120–131. Springer, 2012. xiv

[SHS01] Andreas Savvides, Chih-Chieh Han, and Mani B Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 166–179. ACM, 2001. 8

[Sof15] Software Development Magazine - Programming, Software Testing, Project Management, Jobs. *http://www.methodsandtools.com/archive/archive.php?id=69p3*, Last visited 20-October-2015. xi, 23

[Som04] Ian Sommerville. *Software Engineering (7th Edition)*. Pearson Addison Wesley, 2004. 50, 51

[SOV+11] Marjana Shammi, Sietse Overbeek, Robert Verburg, Marijn Janssen, and Yao-Hua Tan. Agile process for integrated service delivery. *Delft University of Technology*, 2011. 53

[Sri10] Neelam Srivastava. Challenges of next-generation wireless sensor networks and its impact on society. *arXiv preprint arXiv:1002.4680*, 2010. xv

[TG11] Sanjana Taya and Shaveta Gupta. Comparative analysis of software development life cycle models. *ijcst*, 2(4), 2011. 51

[THE16] THE Open GROUP. *http://pubs.opengroup.org/architecture/archimate2-doc/chap08.html*, Last visited 13-03-2016. xi, 6, 27

[TOG] TOGAF. *THE Open GROUP*. http://theopengroup.org/. 33

[TTH11] Luc Touraille, Mamadou K. Traoré, and David R. C. Hill. A model-driven software environment for modeling, simulation and analysis of complex systems. In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 229–237, San Diego, CA, USA, 2011. 14, 15, 17, 28, 41, 43

[vdB09] MGJ van den Brand. Model-driven engineering meets generic language technology. In *Software Language Engineering*, pages 8–15. Springer, 2009. 21, 22, 25

BIBLIOGRAPHY

[VDKV00a] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000. 24, 26

[VDKV00b] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000. 24

[VDSMVB09] Ragnhild Van Der Straeten, Tom Mens, and Stefan Van Baelen. Challenges in model-driven software engineering. In *Models in Software Engineering*, pages 35–47. Springer, 2009. 22

[Vey16] Loeyssiere. Comparison of centralized and distributed fusion architectures in a sensor network. 2016. 10, 13

[VG09] Ehsan Vossough and Janusz R Getta. Micro implementation of join operation at clustering nodes of heterogenous sensor networks. In *International United Information Systems Conference*, pages 75–90. Springer, 2009. 7

[VKMB15] Paola Vallejo, Mickaël Kerboeuf, Kevin JM Martin, and Jean-Philippe Babau. Improving reuse by means of asymmetrical model migrations: An application to the orcc case study. In *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*, pages 358–367. IEEE, 2015. 95

[VMP14] Vladimir Vujović, Mirjana Maksimović, and Branko Perišić. A dsm for a modeling restful sensor web network. In *10th Annual International Conference on Information Technology & Computer Science*, pages 19–22, 2014. 14, 15

[Wan08] Chong Wang. *Localization and its applications in self-configurable wireless networks*. ProQuest, 2008. 8, 9

[WBH+13] Gerold Wefer, David Billet, Dierk Hebbeln, Bo Barker Jorgensen, Michael Schlüter, and Tjeerd CE Van Weering. *Ocean margin systems*. Springer Science & Business Media, 2013. 8

[YL13] Eric Yu and Alexei Lapouchnian. Architecting the enterprise to leverage a confluence of emerging technologies. In *Proc. 1st International Workshop on Advancement from Confluence of Emerging Technologies (ACET 2013) at CASCON*, 2013. 42

[YZL+08] Jue Yang, Chengyang Zhang, Xinrong Li, Yan Huang, Shengli Fu, and Miguel Acevedo. An environmental monitoring system with integrated wired and wireless sensors. In *Wireless Algorithms, Systems, and Applications*, pages 224–236. Springer, 2008. xiii

[ZCKA09] Oussama Kassem Zein, Joel Champeau, Dominique Kerjean, and Yves Auffret. Smart sensor metamodel for deep sea observatory. In *OCEANS 2009-EUROPE*, pages 1–6. IEEE, 2009. xiv, xvii, 34

[ZD13] Marco Zuniga and Gianluca Dini. *Sensor Systems and Software: 4th International ICST Conference, S-Cube 2013, Lucca, Italy, June 11-12, 2013, Revised Selected Papers*, volume 122. Springer, 2013. 6

[Zim11] Walter MX Zimmer. *Passive acoustic monitoring of cetaceans*. Cambridge University Press, 2011. 89

[ZSL+11] Manchun Zheng, Jun Sun, Yang Liu, Jin Song Dong, and Yu Gu. Towards a model checker for nesc and wireless sensor networks. In *Formal Methods and Software Engineering*, pages 372–387. Springer, 2011. xi, 13, 42

[ZYPEQ10] Filip Zavoral, Jakub Yaghob, Pit Pichappan, and Eyas El-Qawasmeh. *Networked Digital Technologies, Part I: Second International Conference, NDT 2010, Prague, Czech Republic*. Springer Science & Business Media, 2010. 53