A High Abstraction Level Constraint for Object Localization in Marine Observatories

A Thesis

Presented to

the Faculty of Natural and Applied Sciences

at Notre Dame University - Louaize

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science in Computer Science

by

Jad Moussa

January 2021

Notre Dame University - Louaize

Faculty of Natural and Applied Sciences

Department of Computer Science

We hereby approve the thesis of

Jad Moussa

Candidate for the degree of Master of Science in Computer Science

Dr. Nazir Hawi	Supervisor, Chair
Dr. Hoda Maalouf	Committee Member
Dr. Maya Samaha Rupert	Committee Member

Acknowledgments

First and foremost, I am extremely grateful to my supervisor Dr. Nazir Hawi for his invaluable advice, continuous support, and patience during my Master study. My gratitude extends to all faculty members at Notre Dame University.

Declaration

I hereby declare that I am the sole author of this thesis. To the best of my knowledge, this thesis contains no material previously published by any other person except where due acknowledgment has been made.

Table of Contents

List of Figures

List of Abbreviations

SN	Sensor Network
UW-SN	Underwater Sensor Network
UML	Unified Modeling Language
EAML	Enterprise Architecture Modeling Language
DSML	Domain-Specific Modeling Language
SOS	System of Systems
DSLC	Distributed System Life Cycle
GPS	Global Positioning System
NEPTUNE	North East Pacific Time-series Undersea Networked Experiments
GLONASS	Global Navigation Satellite System
VENUS	Victoria Experimental Network Under the Sea
DFA	Data Fusion Architecture
CA-PSCF	Context-Aware Pervasive Service Creation Framework
DSM	Domain-Specific Model
ITSML	Intelligent Transportation Systems Modeling Language
MDE	Model Driven Engineering
OMG	Object Management Group
MT	Model Transformation

OCL	Object Constraint Language
ATL	Atlas Transformation Language
MDA	Model Driven Architecture
MOF	Meta Object Facility
XMI	XML Metadata Interchange
CWM	Common Warehouse Metamodel
IDE	Integrated Development Environment
EMF	Eclipse Modeling Framework
EA	Enterprise Architecture
TOGAF	The Open Group Architecture Framework
FEAF	Federal Enterprise Architecture Framework
ADM	Architecture Development Method
GSN	Global Sensor Marketing
SWE	Sensor Web Enablement
SS	Smart Sensor
DFS	Dara Fusion Server
AS	Algorithm Selection
DT	Data Transmission

DA	Data Acquisition
OLA	Object Localization Algorithm
SSS	Smart Sensor System
FS	Fusion System
MR	Manage Resources
IMS	IP Multimedia Subsystem
NOAA	Natural Oceanic and Atmospheric Administration

Abstract

Marine observatories based on sensor networks provide continuous ocean monitoring. The design phase of such systems, which is part of the complete development life cycle, is a complex and challenging task. The design difficulties may induce the designers to make architectural design errors during the design phase. This study aims to identify the best design approach that helps sensor networks designers in preventing errors and validating models at an early phase. In addition, it introduces a new environmental constraint that should be taken into consideration when building design models.

To determine the best approach, a comparison among several sensor networks design approaches has been conducted, based on the requirements of sensor networks designers. The results showed that extending an Enterprise Architecture Modeling Language, by adding new domain components and constraints, contributes toward satisfying all the designers requirement.

Our contribution is based on a research paper titled "A High Abstraction Level Constraint for Object Localization in Marine Observatories" (Aoun at al., 2017). In this research paper, we implemented the proposed constraint in ArchiMO, a design tool that extends ArchiMate metamodel by adding domain concepts.

This work aims to demonstrate that we can improve the development process of such complex systems based on the use of Model-Driven Engineering methodology and Domain-Specific Modeling Languages. The improvement is achieved by providing an early validation step via models' validation to consolidate the system design.

Keywords: sensor networks, marine observatories, enterprise architecture, model-driven engineering, domain-specific modeling language

Chapter1: Introduction

1.1 General Context

The internet exploded across the world and has become an essential element of humans' everyday life. Everyone wanted to connect his computer to the web. Then, the demand increased for portability. Laptops have become as popular as pocket calculators. That hasn't been enough until smartphones were invented to get access to the internet from everywhere at any time. Researchers went beyond connecting just computers to the web. These researches led to the birth of the Internet of Things. Now, billions of devices are connected to the internet, from simple sensors to smartphones and wearables, all connecting and sharing data. By integrating these connected devices with automated systems, it is possible to gather information, analyze it, and create an action to help someone with a specific task or learn from a process. However, recent developments in the field of embedded devices have led to smart things becoming increasingly popular in our daily lives. All of these technologies are related to the Web of Things architecture and rely mostly on Sensor Networks (SNs).

Sensor networks are the basis of the environmental monitoring systems infrastructure. According to Yang et al. (2008, pp. 224–236), the environmental monitoring system is based on an integrated sensors concept that is structured to store important data with signal processing hardware in a single compact device. These smart sensors are naturally integrated into a distributed data processing, data storage, and data presentation system. These systems present various sensor data on web clients, such as pressure, temperature, humidity, smoke, gas, and sound (Lauterbach et al., 2004, pp. 256-266). These presentations are coupled with data processing to provide high-level services (e.g., location of moving objects) based on the

composition of basic functions. This involves configuring multiple devices on the SN, and each device has specific functions and offers different services.

The design of such systems has therefore become increasingly complex based on the growing number of functionalities, processing functions, sensors, and incorporation into an information system. The design phase of these systems, which is part of the complete system life cycle, is concerned with the complexities and challenges of mapping high-level services and the collection of functions on the SN architecture. During the design process, these design difficulties can induce the designers to make architectural design errors which may have significant implications on the entire system's functioning and performance.

To avoid the problems of architectural design, the design of such complex systems should be validated at an early phase. To this end, a new approach should be provided to SN designers to build accurate models, with minimum margin of errors, in order to reduce the design phase's complexity.

1.2 Sensor Networks

1.2.1 Sensor Networks Definition

"Sensor Networks are in which sensed data are periodically gathered at a single point, or sink, for external transmission and processing" (Cuzzocrea, 2009). This definition may be regarded as a two-phase procedure (Xianwei et al., 2012, pp. 120-131): (1) observation/measurement, which means the accumulation of the data collected at each sensor node; and (2) transfer of the data collected to a certain processing center within the SN.

SNs can be used for environmental monitoring, intrusion detection and target tracking, infrastructure monitoring, precision agriculture, environmental monitoring, etc. Also, they can be used for marine monitoring since they have a number of advantages such as unmanned

operation, easy deployment, and real-time monitoring. The focus of this study is the ocean survey, which provides a continuous way to track and observe objects moving underwater. This observation includes underwater environmental measurements and the collection of estimated variable object (Champeau et al., 2009, pp. 1-6). Underwater Sensor Networks (UW-SNs) are therefore expected to be utilized in this context. The Underwater Sensor Network conducts tasks like data collection, then transfers data from one system to another for all types of underwater environmental monitoring (Lee et al., 2008, pp. 322-329).

1.2.2 Sensor Networks Implementation

There are different forms of SN (e.g., wireless and wired) (Ahmed et al., 2006, pp. 1-3). Series of connected monitoring equipment such as sensors, servers, and communication infrastructure are needed for the implementation of these networks. Such devices have several specifications that differ from environmental constraints.

According to our field of study, we are engaged in aquatic environmental constraints. To this end, attention should be given to many underwater environmental restrictions during the implementation of the UW-SNs. We cannot disregard the existence of underwater communication constraints that should be taken into consideration during the implementation of UW-SNs, such as the type of cable used to connect a sensor to a server, which is a physical constraint, or even the length of the cable being used, which is a logical constraint. Otherwise, this will adversely impact the underwater communication efficiency and activities, such as the delay in transmitting data between sensors and servers (Reed, 2015). Therefore, many specifications are required to deploy the UW-SNs with the appropriate equipment (Heidemann et al., 2005): acoustic communication, such as marine cables between sensors (hydrophones) and

workstations (fusion servers); network configuration (e.g., sensor configuration and fusion servers); application (e.g., trilateration algorithm).

1.2.3 Adopted Definition of Sensor Networks

Relying on Erol et al (2007, pp. 44-54) and Moradi et al (2012, pp. 4352-4380), and based on our context, we define Underwater Sensor Networks as follows: "An Underwater Sensor Network is a group of anchored sensors with an infrastructure for underwater communication designed to get, share, monitor, and combine data between different nodes. Then, processing and passing information to provide high-level services such as tracking or finding an underwater moving object".

1.2.4 Complexity and Challenges of Sensor Networks

The implementation of SN is necessary to attain the observation and monitoring missions of a given area. SN is based on a series of sophisticated sensors with a communication network for monitoring and recording data at different locations. It consists of components (software/hardware) with different levels of computational and communication capabilities with specific protocols for interaction. As such, SN is considered as a complex distributed system in (Champeau et al., 2011).

We distinguish three sources of life-cycle difficulty: the complexity of the system itself relates to the services and the number of functions rendered; the development and design activities; and the implementation of the system (Cuzzocrea, 2009) (Bejar el al., 2005, pp. 117-147).

According to Srivastava (2010), there are two main challenges when it comes to SN deployment: (1) The system architecture, because there is no single framework and networking architecture to construct different applications at the top; (2) Hardware prices, as the current cost

of a single sensor unit is very high. Therefore, the installation of the collection of underwater sensors (hydrophones) is a costly process, as we are interested in the underwater environment. This is also due to the appropriate equipment, such as unique boats, marine cables, and diving experts, etc. Also, we cannot neglect the risky deployment process, and the location of the underwater sensors and servers should be in the right place.

Many stakeholders engage in the implementation of SN and are part of the life cycle. “A stakeholder is a person, group, or entity with an interest in concerns about the realization of the architecture” (Rozanski & Woods, 2005).

The SN life cycle design process is divided into two key levels: behavioral (logical analysis) and architectural networking. Each level needs a designer that is different from the other and that is for a system's better implementation of network technologies. The design process needs various stakeholders as per their domains. In this document, we're concentrating on SN designers among these stakeholders.

To face the complexities and challenges of developing such distributed system architecture, SN designers should be provided with support at the design level to cover, elaborate, and evaluate all aspects of a SN.

The goal of the design process is to provide the physical network infrastructure with an architecture that includes all the aspects of the service architecture concept. One of the important approaches to achieving that aim is to provide abstractions of the final structure to concentrate on the architectural intent.

In this context, the use of a modeling language is sufficient to concentrate on our intent, given that the modeling language contains adequate abstractions. Several modeling languages are candidates for the modeling phase, ranging from general-purpose languages such as UML to

more network-based languages such as the Enterprise Architecture Modeling Language (EAML) or the Domain-Specific Modeling Language (DSML).

An effective modeling approach trade-off is the reuse of current or common modeling language with a specialization in our SN context. In this case, to get early validation on the architectural hypothesis, the models must be effective. In our context, a network infrastructure simulation that supports our application's high-level services can achieve the validation process. These phases of modeling and simulation may be iterative to strengthen the mapping of services on the network infrastructure. To this end, a strong tooling is needed to ease the phases of modeling and simulation.

1.3 Research Question

The research problem that drives this thesis deals with preventing losses in the deployment phase while performing the design phase of SNs.

The research questions are as follow:

1- How to prevent errors at design time by adding a new environmental constraint?

2- How to implement the proposed constraint in an existing design tool?

Chapter 2: Sensor Networks Development Process

2.1 Sensor Networks

Sensor Networks (SN) are composed of heterogeneous sensors with capabilities in communication and heterogeneous components dedicated to one or more application domains. These systems position themselves on a wide spectrum of domains, such as environmental monitoring, ocean monitoring, transportation monitoring, etc.

To provide a generic definition of these systems, we rely on Jamshidi's (2017) Systems of Systems (SoS) definition, SoS is: "large-scale integrated systems which are heterogeneous and interdependently operable on their own but are networked together for a common goal".

2.1.1 Sensor Networks Life Cycle

One of the crucial phases of the SoS life cycle is the logical and physical allocation of functional architecture to the SoS architecture. This phase includes an exploration of the architecture to target SoS requirements. Also, Jamshidi (2017) defines SoS as: "large-scale concurrent and distributed systems that are comprised of complex systems", so the associated life cycle is considered as a distributed systems life cycle (DSLC) to take into account the geographic, operational, and managerial independence and the different temporal evolutions.

To define a DSLC life cycle, two main types of phases should be taken into account: (1) for development purposes such as the phases of design and analysis; (2) for implementation purposes such as the phase of deployment. Accordingly, we differentiate several DSLC definitions. Gordon (2010) 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.

More to identify the preferred life cycle for our SN context, we would like to highlight some suggestions that Gordon (2010) insists on:

1- 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.
2- 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.
3- The importance of iteration: errors are detected during testing or deployment re- quires 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 suggestions presented above elaborate on the problems that can occur during the life cycle of the SN design and deployment phases. They also elaborate that the errors that may occur in the deployment phase can be prevented by detecting them in the design phase by adopting the iteration approach. This reflects the need for effective support for the tasks to be carried out during the design phase to reduce the complexity of performing these tasks. This support can be provided by emphasizing the allocation of functional software components to the physical architecture with minimum architectural errors. Then, having a validation tool for the defined physical architecture. To this end, in the next section, we focus on the design phase among the various phases of the SN life cycle.

2.1.2 Design Phase of the Sensor Network Life Cycle

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, as can be seen in Figure 1. In general, a system architecture description is defined in a view that addresses a set of related concerns related to stakeholders.

An architecture concern is "a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture" (Rozanski & Woods, 2005). According to Katara and Katz (2007, pp. 247-265), architectural concerns are "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 viewpoints. And also relying on Rozanski and Woods (2005), 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.

2.1.3 Roles in the Sensor Network Life Cycle

As we have just described, different stakeholders are involved in the life-cycles and their roles are closely related to the viewpoint definition. To refine the SN life-cycle definitions, we identify the stakeholders and their roles that are particularly involved during the design phase, which remains a critical phase of those systems.

Zuniga and Dini (2013) 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.

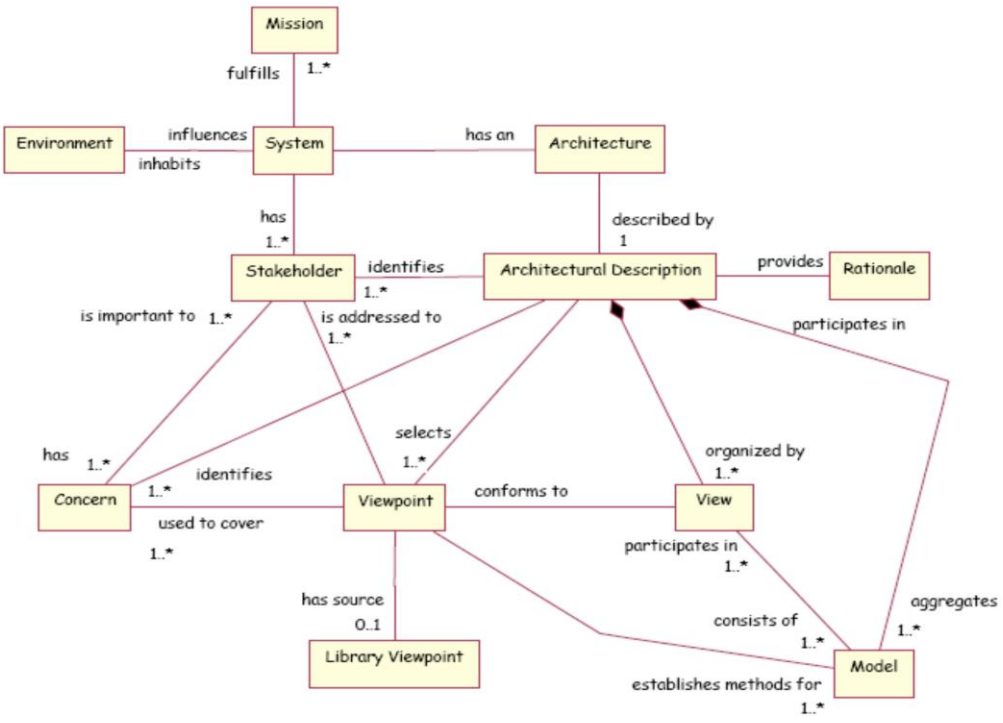
2- Software Designer: define an architecture of the sensor network applications by specifying software components, distribution of actions, events, services such as a service includes functions and procedures, and relationships between components.

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.

Figure 1

Conceptual Model of Architectural Description



Note. Adapted from *ArchiMate® 2.1 Specification*. (n.d.). Retrieved December 20, 2020, from https://pubs.opengroup.org/architecture/archimate2-doc/chap08.html

2.2 Sensor Networks Systems

In order to consider the use of SN systems, we try to know how to use the sensor data to provide a higher-level customer-based service. NEPTUNE Canada, American Global Positioning System (GPS), and European GALILEO System, and Russian GLONASS System are the systems or SoS that deliver continuous customer services. Such systems provide important information on the properties and behavior of complex systems (Vossough & Getta, 2009, pp. 75-90). High-level services, such as continuous observations and weather parameter processing, or continuous observations of moving objects, and many others (Vossough & Getta, 2009, pp. 75-90), are based on a wide range of functions (Mills, 207, pp. 823-834), including data acquisition, tracking, routing, fusion, distribution, storage, and querying. The resulting information may be considered as an unlimited sequence of complex data items from the sensor nodes and ready for analysis and storage (Mills, 207, pp. 823-834) (Vossough & Getta, 2009, pp. 75-90).

For example, NEPTUNE Canada (North East Pacific Time-series Undersea Networked Experiments) allows for real-time study of tectonic plates, fluid movement on the ocean floor, and the effects of climate change on marine ecosystems; It will also help to locate earthquakes accurately and observe the resulting seismic stress. Additionally, it measures salinity, carbon dioxide, and even organism movement in sediments. The multitude of data collected will then be transmitted via optical fiber to Vancouver Island's Port Alberni station, connecting them via a broadband internet connection at Victoria University. NEPTUNE's data archiving and management system also collects information from VENUS (Victoria Experimental Network Under the Sea), a coastal network of underwater observatories that are linked by cables. NEPTUNE Canada deploys a set of connected nodes and instruments that must be anchored to

optical cables on a SN to provide these services. These nodes and instruments could be hydrophones, seismometers, video cameras, and cameras with high resolution. Several other sub-systems, such as database, database replication, and web server, are also deployed and connected to the SN. To fuse the sequence data items that come from different sensors, a Database is deployed. A replication of the database is deployed to have a copy of all the data collected and analyzed. A web server for the diffusion and display of data on the web is deployed.

Such systems are therefore not just a simple SN; they are systems equivalent to enterprises with high-level services delivered by a high number of interconnected heterogeneous components (Mills, 207, pp. 823-834). For example, the Neptune system is not just an SN, it can be considered as an enterprise system because it requires multiple heterogeneous components (software and hardware) with communication constraints to be deployed on the SN to deliver services similar to the above.

These information systems are categorized according to two main types: wireless outdoors and sub water (Wang, 2008). Such systems may be divided into non-acoustic and acoustic types (Iniewski, 2012). A localization and tracking service for moving objects is available in both types, through a set of sensors and appropriate algorithms. These systems adopt two main approaches (Wang, 2008) (Iniewski, 2012):

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 (Caiti et al., 2005, pp. 140-152).

Based on these approaches, high-level services like a 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) (Choi & Lee, 2010, pp. 1457–1465). 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 (Loicq et al., 2017, p. 105632L).

Data fusion algorithms are differentiated and categorized according to the number of receivers (Boukerche et al. 2008, pp. 2838–2849) (Wang, 2008). Relying on Kaplan and Hegarty (2005), mobile object position accuracy increases with the number of receivers. The localization service's efficiency is highly correlated with the increasing number of sensors. There are several types of fusion algorithms for this purpose, which are differentiated by the number of sensors in the network.

We also focused our interest on the fusion algorithms and related services, as they are quite representative services included in the SN and the extended systems. They provide a high-level service in many SNs and are based on an undetermined number of sensors and there are several deployments of elementary algorithm nodes.

2.3 Fusion Algorithms

A Fusion algorithm act as a representative service in a SN to provide a relevant example of a high-level service for the SN. This type of algorithm is necessarily highly distributed by nature based on heterogeneous components (sensor nodes and fusion nodes) and several architectures may be deployed on selected network infrastructure.

There are several fusion algorithms such as: trilateration, triangulation, Bounding-Box, set-membership, and Dive'N'Rise (Caiti et al., 2005, pp. 140-152) (Han et al., 2012, pp. 2026-2061).

Since it is possible to extend the trilateration to more than three sources, it can be adopted while providing the moving objects localization service. It also gives the possibility to increase the precision of a moving object's determined position. To this end, the trilateration algorithm is used by the world's most relevant positioning system, such as the GPS (Wang, 2008). This algorithm combines and integrates information from various sources (sensors), using a data fusion approach architecture (Castanedo, 2013).

However, it is possible to use several Data Fusion Architectures (DFA) while performing the fusion algorithm. This diversity of DFA types creates the possibility to create different models according to each type by SN designers, with difficulty choosing the most convenient one. This designer should select the architecture appropriate for the components required to perform this algorithm. Thus, this designer's decision is completely oriented towards selecting the appropriate architecture for data fusion among several. The properties that can fit with those architectures should be identified to select the appropriate one. Then, to select the most satisfying architecture, the SN designer checks the availability of each property in the different data fusion architectures. We present these properties for this purpose, in the next section.

2.4 Properties for Selecting a Data Fusion Architecture

The SN designer faces several challenges in adopting the data fusion approach and its logical architecture, and the corresponding mapping of the distributed network infrastructure. The following challenges and difficulties are discerned:

1- Various heterogeneous components should be deployed in the SN by adopting a data fusion architecture. Multiple functions are assigned to each component in the SN (Mitrou et al., 2004). This requires a communication infrastructure with varying levels of computational and communication capabilities based on each communication protocol.

2- The number of sensors should increase with the required degree of accuracy in locating objects. This may cause the need to increase the number of other components, such as fusion servers. As a result, the number of heterogeneous components on the SN is dynamic and can be increased at any time without any component constraints.
3- The effects on the network by the sensor failures should be minimized.

Accordingly, to select the appropriate data fusion architecture, the properties required by the SN designer should be detailed. This should offer the ability to add a large number of sensors and fusion servers to SN. This capability increases the accuracy of locating a moving object, or extends the observed area covered. Additionally, relying on Kaplan and Hegarty (2005), it is necessary to deploy more sensors and fusion nodes on the SN to perform and optimize the fusion service. The SN designer therefore requires an architecture that has sufficient flexibility to cover the need to add or remove components with less impact on the application. These modifications must be carried out on the application architecture without calling into question the system architecture and are inevitable over a long-life cycle of the system.

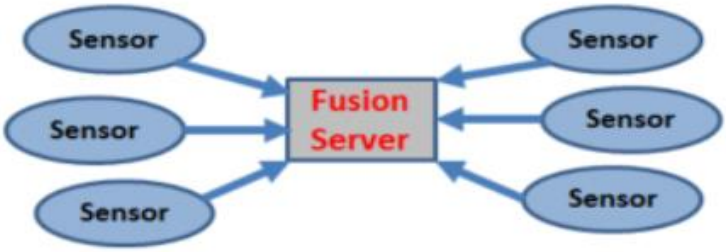
2.5 Data Fusion Architectures

In the SN domain, the information sources are sensors, and the fusion algorithm is often performed in a fusion center that collects this information (El Zoghby, 2014) (Mitchell, 2007) (Liggins et al. 2017), but it can also be decentralized (El Zoghby, 2014). The main possibilities for the categories of architecture are centralized, hierarchical, and distributed (André, 2013). There is a single fusion node within a centralized architecture (Loicq et al., 2017, p. 105632L), as can be seen in Figure 2 (Aoun et al., 2017, pp. 605-6011). Sensors acquire data and then transfer it directly to a single central fusion node. In the hierarchical architecture, as can be seen in figure Figure 3 (Aoun et al., 2017, pp. 605-6011), fusion nodes are classified in a hierarchy

with higher-level nodes processing results from lower-level nodes and may provide some feedback (Loicq et al., 2017, p. 105632L). In a fully distributed architecture, as can be seen in Figure 4 (Aoun et al., 2017, pp. 605-6011), there are several fusion nodes. Each node sends information to the other nodes (Khosla et al., 2017) (Chong & Mori, 2005). There is no predetermined hierarchical relationship so that each fusion node can communicate with any other node.

Figure 2

Centralized Fusion Architecture



Note. Adapted from Aoun, C. G., Lagadec, L., Champeau, J., Moussa, J., & Hanna, E. (2017). A High Abstraction Level Constraint for Object Localization in Marine Observatories. *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 605–611.

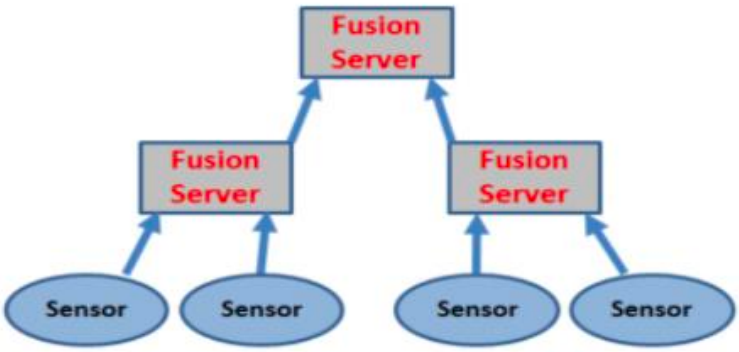
2.6 Data Limits and Comparison Among Different Data Fusion Architectures

The number of sensors and fusion nodes deployed on the network can be changed as required in a fully distributed architecture (André, 2013). The accuracy of the position of moving objects increases when more sensors are added. Consequently, the distributed fusion architecture

allows sensors and fusion nodes to be added to SN as much as is required to enhance the results of the services provided, such as moving object localization.

Figure 3

Hierarchical Fusion Architecture



Note. Adapted from Aoun, C. G., Lagadec, L., Champeau, J., Moussa, J., & Hanna, E. (2017). A High Abstraction Level Constraint for Object Localization in Marine Observatories. *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 605–611.

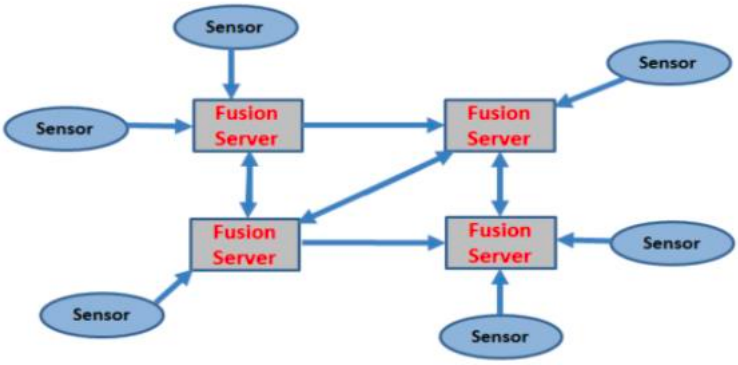
The number of sensors and fusion nodes that are deployed on the network cannot be changed in a centralized architecture as required. This is due to the centralized architecture structure since one single fusion node can be deployed on the SN. Thus, this architecture cannot contribute by increasing the accuracy of object localization.

We can add more sensors and fusion nodes in a hierarchical architecture but that is a complex task. This is due to the required feedback scenario (communication constraint), which should be executed between the fusion node of the sender and the fusion node of the receiver

(Loicq et al., 2017, p. 105632L). As a result, communication constraints increase in parallel with the number of added sensors and fusion nodes.

Figure 4

Distributed Fusion Architecture



Note. Adapted from Aoun, C. G., Lagadec, L., Champeau, J., Moussa, J., & Hanna, E. (2017). A High Abstraction Level Constraint for Object Localization in Marine Observatories. 2017 International Conference on Computational Science and Computational Intelligence (CSCI), 605–611.

Consequently, the property of adding several components to the SN is available in hierarchical and distributed fusion architectures. In the distributed architecture, the fusion nodes can be added at any location on the SN without any hierarchical level of execution. This means that all the nodes can communicate together without any kind of causal execution. As a result, the distributed architecture provides a more flexible approach when taking into account the long time period of the system life cycle.

Based on the analysis and comparison discussed above, the SN designer should take into account the distributed fusion architecture to perform the trilateration fusion algorithm. Thus, the Distributed Architecture is the most generic one associated with our requirements.

In addition, the architecture selected should be compatible with the trilateration fusion algorithm (Liggins et al., 1997, pp. 95-107). According to Alriksson and Lantzer (2007, pp. 5499–5504), this algorithm adopts and uses a distributed fusion architecture.

Making a SN durable in case of individual sensor failures is another major advantage of selecting a distributed architecture. The loss of one sensor in a distributed environment does not affect the entire SN (Khosla et al., 2017) (Chong & Mori, 2005). Thus, there is no component that constitutes a weak point that paralyzes the system (André, 2013).

For the purposes set out above, the distributed fusion architecture is the preferred one among the others. This is also due to the advantages provided by this architecture, such as reliability and energy efficiency (Liggins et al., 1997, pp. 95-107) (Khosla et al., 2017) (Chong & Mori, 2005).

2.7 Requirements for Designing Sensor Networks Systems

Most SN systems adopt distributed fusion architecture. In order to perform the design phase of such systems, the SN designers face a number of challenges. A model should be defined to describe and analyze how to map the service provided (e.g., localization of moving objects) on a SN. This requires the intervention of different SN designers who are experts in different fields of experience. The service provided must be described in a model by a SN designer who is an expert in the business process of that service. In order to perform such a service, a model containing the corresponding software components and relationships must be defined by another SN designer who is an expert in the application process. In order to implement these software

components and relationships, a model that includes an appropriate technology platform is required to be defined by a network infrastructure expert. In addition, as SN designers define models in a complex context, architectural design errors may occur (Li et al., 2010, pp. 386-399). Developed models must therefore support the analysis of architectural constraints that must help designers and avoid errors that would be detected during the next phase of process development.

For this purpose, SN designer should consider two parts when designing such systems: (1) services related to the business domain, such as the location of moving objects; and (2) information systems to support the deployment of these services and provide sufficient flexibility to take into account the long-life cycles of these systems.

To design such information systems, suitable tooling (Hoffmann et al., 2002) has to support the SN designer. According to Rowe et al. (2010), graphical modeling languages could support the design approaches. These approaches and their related tools help in the design and deployment of SN applications (Rowe et al., 2010). These tools provide the SN Designer with the ability to analyze and model complex systems like NEPTUNE Canada (Zheng et al., 2011, pp. 372-387). Thus, SN designer's requirements can be coupled with this tool's requirements. These requirements are identified as:

1- **Requirement 1** Improving Architectural Design: Possibility of preventing architectural errors that may be made by the designer of the SN. These errors can occur when defining the services provided that the adopted SN architecture performs in relation to the communication constraints. For example, architectural errors can be prevented by avoiding the connection of two sensors or the connection between a sensor and a database server without any processing node.

2- **Requirement 2** Multiple Viewpoints: Providing each designer with the ability to work independently from a viewpoint, in order to have their proper model in accordance with their field of experience. The different designers involved collaborate to share the design of the same system. And in order to have a consistent model, these different viewpoints need to correlate. For example, in their independent viewpoints, the domain expert, software designer, and network designer create their models, so they cooperate with each other by interrelating their models to obtain a consistent model. This interrelation can be established through the use of specific relationships to ensure consistency of this unique model.

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

4- **Requirement 4** Heterogeneity Supported: Ability to have different types of components and communication types in the same defined SN model. Since the SN is an information system that consists of heterogeneous devices and communication protocols, it is necessary to have this capability while defining the scenario of the services provided. For

example, the communication protocol between sensors and fusion servers is different than the one between a fusion server and a database server.

5- **Requirement 5** Validation Tools Supported: Ability to validate the SN model in order to detect architectural errors during the design phase and to validate the created model earlier in the development life cycle. For example, using a network simulation tool to evaluate the mapping of service on an SN infrastructure.

2.8 Limits and Comparison Among Different Approaches of Sensor Networks Design

In order to select the appropriate approaches to design and then implement an SN system, we will discuss a comparison of the different approaches to the requirements, previously identified. Our comparison is based on the most relevant approaches regarding our context: SimStudio (Touraille et al., 2011, pp. 229-237), CA-PSCF (Context-Aware Pervasive Service Creation Framework) (Achilleos et al., 2010, pp. 281-296), DSM (Domain-Specific Model) (Vujović et al.), ITSML (Intelligent Transportation Systems Modeling Language) (Alberto Fernández-Isabel & Rubén Fuentes-Fernández, 2015, pp. 14116-14141), and SysWeaver (Rowe et al., 2010). The results are summarized in Figure 5 and discussed in the next sections.

2.8.1 Approaches of Architectural Design Improvement

All the approaches illustrated above provide tools for modeling and validating high-level services, they offer SN concepts. But they did not propose constraints on the concepts themselves nor on the communication between them. Using a design tool that offers such constraints may help the designer to prevent architectural design errors which lead to the improvement of the architectural design.

Consequently, all the discussed approaches help the designer to design SN by using specific components in this context. Adding domain-specific constraints may further help the designer detecting errors at early design phases.

2.8.2 Approaches of Providing Multiple Viewpoints

SysWeaver is the only approach that offers multiple viewpoints while designing an SN model. Different designers, with different domains of experience, are involved in building the model. A disadvantage is that it does not contain advanced and complex network components.

Consequently, SysWeaver cannot be used in our context unless it is extended furthermore in order to include different layers.

2.8.3 Approaches of Offering Concepts Extensibility

All the discussed approaches offer built-in components in order to design an SN. They are all extendable to add new concepts and relationships but doing this will not respect the semantics of newly added concepts and relationships.

2.8.4 Approaches of Supporting Heterogeneity

Components' heterogeneity is supported by all approaches. Each component used to build an SN has a different function and offers different services.

2.8.5 Approaches of Supporting Validation Tools

SimStudio, ITSML, and SysWeaver are the only approaches that support model validation. They enable the SN designer to simulate the created model and detect architectural error prior to the implementation phase.

Figure 5

Comparison Among SN Design Approaches

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	✓	✓	✓

2.9 Discussion

After discussing SN approaches in the previous section, the following can be concluded:

1- Improvements of SN architectural design is possible in all of the approaches presented. In particular, the frameworks are focused on the interpretation of domain concepts that provide efficient support for the designer. We notice, though, that these approaches lack the domain constraints relative to the concepts and relationships of the domain. In the next section, we will look for the relevance of the Model-Driven Engineering approach relative to our context to improve the lack in the actual tooling and to keep the domain concept definition.

2- Multiple Viewpoints are addressed by a single approach of all the approaches presented, the SysWeaver. It provides separate viewpoints with the ability to interrelate these viewpoints according to each domain of experience, in order to have one unique model.

However, this approach focuses mainly on the design of small-scale SN and does not fulfill the necessity to develop complex information systems like NEPTUINE. This is why we study and analyze the Enterprise Architecture frameworks to try to identify a system approach including SN and the necessarily associated IT infrastructure.

3- Extensibility of all the solutions discussed is feasible even though the related tooling is still difficult to extend. In addition, we find that some tools are based on a meta-modeling approach that offers a simple language description to facilitate language extensibility. However, these approaches may be extendable to a certain level. We will apply a Model-Driven Engineering approach in order to meet the requirements.

4- All the discussed approaches support heterogeneity. However, the design tool must also support domain constraints.

5- Not all the approaches include the validation step. Simulators are required in order to simulate created models.

Chapter 3: Model-Driven Engineering

3.1 Model-Driven Engineering Fundamentals

This chapter represents how Model-Driven Engineering can help improve the design of the SN and the associated information system. Based on Parreiras (2012) and Van Den Brand (2008, pp. 8-15), the fundamental concepts of Model-Driven Engineering (MDE) are model, metamodel, and model transformation.

A model is a simplified view of a system. A model's purpose is to explain and enhance the understanding of the system, often at many levels of abstraction. A model selects interesting concepts on viewpoints for a given context and provides a representation of the reality for a dedicated purpose.

As we have previously mentioned, SN designers need to use modeling languages to construct models. Therefore, in the upcoming sections, we will elaborate on the key current modeling languages. Semantics, abstract syntax, and concrete syntax characterize the modeling language. There are several methods of formally defining abstract language syntax (Krahn et al., 2007, pp. 286-300). Metamodels are used in the modeling context to define the abstract syntax of modeling languages.

A metamodel is: "a model that defines the language for expressing a model" (Gašević et al., 2007, pp. 91-105) (Object Management Group). Metamodeling is a common approach that defines the Domain-Specific Modeling Language (DSML) abstract syntax, so the designer can map the classes of a domain analysis directly to the classes of the metamodel. Associations and inheritance of domain classes are also mapped to the language definition. For this purpose, to define Modeling Languages such as UML or DSML, the metamodeling method can be adopted.

A metamodel is not a model of a model and is not a language in itself; it is a model that defines a language, an explicit and concrete description of a language, to describe models.

In the four-layer approach advocated by the OMG standard organization (Model0, Model1, Model2, Model3 in Figure 6), each layer conforms to the upper layer.

Model Transformation (MT) is a set of rules that apply to the elements of a metamodel (Van Den Brand, 2009, pp. 8-15). To construct a target model that conforms to the target metamodel, the transformation engine reads the source model that must conform to the source metamodel, and applies the rules specified in the transformation model.

There are two key types of transformations: Endogenous, where the source and target models conform to the same metamodel such as a UML Model to another UML model; Exogenous, where the source and target models are expressed using different languages.

3.2 Model-Driven Engineering Aspects

Van Der Straeten et al. (2008, pp. 35-47) have identified major aspects in MDE which are:

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.

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 (France & Rumpe, 2007, pp. 37-54). Also, ensuring a correspondence between inconsistent quality aspects in and between viewpoints.

4- Model Validation: verifying and testing the models and the code generated from those models.

5- Model Transformation: 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 the development life cycle.

In the coming sections, we will concentrate on three aspects relevant to SN design: Modeling Languages, Heterogeneity and Quality Model, and Model Transformation.

3.2.1 Modeling Languages

There are two types of modeling languages: those that can be adopted in any domain for general purpose, such as UML; and Domain-Specific Modeling Languages that are used in specific domains. It is difficult to create, using a general-purpose modeling language, a model for a specific domain system due to the complexity of describing the precise meaning of domain concepts and their relations. As a result, general purpose modeling languages, such as UML, are not well suited to cover some of the specifications of the SN designer. However, the Domain-Specific Modeling Language (DSML) is specifically developed for a technical or business domain, typically containing a limited number of concepts, and is used by a limited number of professional and expert users (Van Deursen et al., 2000, pp. 26-36).

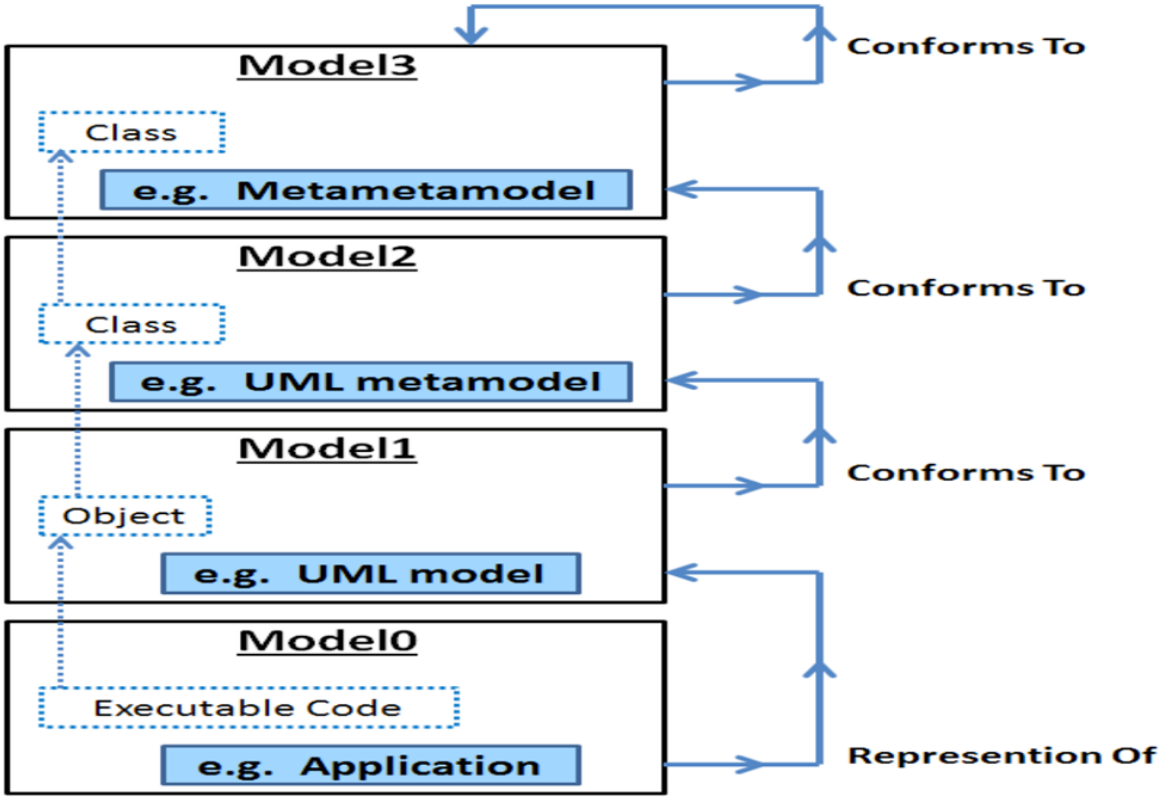
There is a large number of DSMLs with very different abstraction levels. Various studies, including two reported in (Kieburtz et al., 1996, pp. 542-552) and (Gray & Karsay, 2003), ensure that specific languages enable specialists and experts to improve productivity and efficiency in solving problems over the use of General-Purpose Languages. Also, DSML makes it possible to provide unique domain components in abstract syntax, concrete syntax, and modeling language semantics. These components could be ready for use during the design process. DSML supports the work of domain specialists such as designers.

The definition of a language includes several activities that are complementary to each other: (1) defining the abstract syntax of a language, and the corresponding graphical representation of that abstract syntax, which is the concrete syntax; and (2) defining the meaning of a language, the semantics (Krahn et al., 2007, pp. 286-300) (Harel & Rumpe, 2004, pp. 64-72). The description of abstract syntax consists of describing the concepts used in the modeling language. Defining a concrete syntax consists of defining the use of abstract syntax concepts.

In order to define the constraints relative to the metamodel concepts and relationships, the Object Constraint Language (OCL) (Cabot & Teniente, 2007, pp. 179-195) is used 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.

Figure 6

Layered Architecture of MDE



Note. Adapted from *OpenUP - The Best of Two Worlds: Agile, Scrum and RUP*. (n.d.). Retrieved December 21, 2020, from http://www.methodsandtools.com/archive/archive.php?id=69p3

3.2.2 Model Heterogeneity and Quality

The use of viewpoint models in the process of building a complex software design phase becomes a standard fact. The problem is to deal with heterogeneous models and the need for integration at the model level, 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, 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.

3.2.3 Model Transformation

Model Transformation (MT) is a set of rules that defines and controls the transformation process of a single model into a target language, as can be seen in Figure 7. Atlas Transformation Language (ATL) is a model transformation tool that is part of the Eclipse Modeling project (Kurtev et al., 2006, pp. 602-616); it provides ways to produce a set of target models from a set of source models.

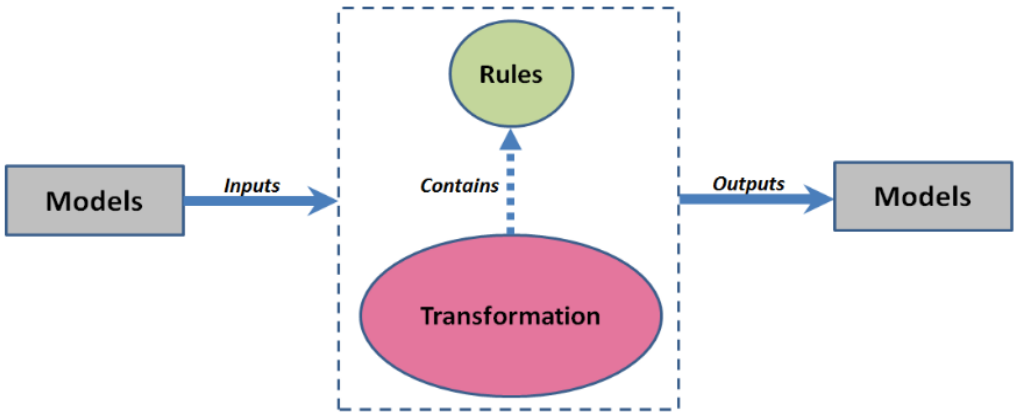
3.3 Separation of Concerns in MDE

As we discussed in the previous section, the system architecture description is based on several concerns and viewpoints. One of the main features of the MDE approach is to provide languages and tooling which defines viewpoints and views on the system. To provide a modeling approach based on the separation of concerns, the modeling framework must take into consideration this definition.

The separation of concerns appears at various points of the system life cycle, and therefore takes on several forms. It can deal with the time separation of each phase, from design to implementation, of the development process. Also, many viewpoints are required to explain different design concerns for each phase of the process.

Figure 7

Model Transformation



Several viewpoints and stakeholders need to be identified to achieve the separation of concern process. The top half of Figure 8 shows the designing, deciding, and informing viewpoints relating to the design, analysis, and dissemination of enterprise architecture. The stakeholders are identified relative to the viewpoint's definition. The main stakeholder is the designer based on our context.

The different levels of abstractions, from details to overview, are illustrated in the bottom half of Figure 8. The integrity and consistency of the system model are maintained by the coherence between the abstraction levels. To define models of each viewpoint, several modeling

languages can be used. However, the use of generic languages such as UML does not provide domain-specific constraints in the design phase unless they are extended.

Due to the complexity of the design of SN, many unique constraints should be respected by SN designers during the design process to prevent architectural errors. These constraints could be implemented into DSML for various levels of abstraction to enhance the consistency of architectural design during the design process. Furthermore, the generated models can be effective at different levels of abstractions by implementing the use of DSMLs during the design process (Van Deursen et al., 2000, pp. 26-36). The use of Domain-Specific Modeling Language (DSML) has advantages for our context to apply separation of concerns and to ensure consistency of the model through domain constraints.

3.4 Model-Driven Engineering Standards and Tools

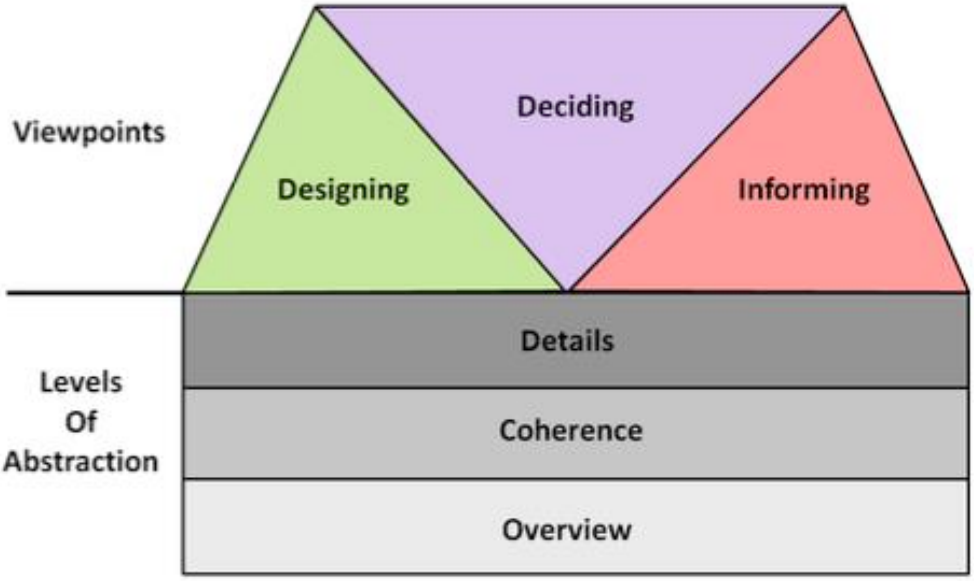
Model-Driven Architecture (MDA) is an approach to software design, development, and implementation led by the Object Management Group (OMG). It provides guidelines for structuring software specifications using a model-centric approach. MDA separates business and application logic from underlying platform technology. MDA is supported by the Unified Modeling Language (UML), the Meta Object Facility (MOF), XML Metadata Interchange (XMI), and the Common Warehouse Metamodel (CWM).

The Meta-Object Facility (MOF) is a standard of the Object Management Group (OMG) for model-driven engineering for describing, defining, and manipulating metamodels (MetaObject Facility | Object Management Group, n.d). There is a range of tools that could be adopted for MDE with various target users' concerns. Based on surveys for MDE tools conducted in (Hussey et al., 2010) and (Pérez-Medina & Dupuy-Chessa, 2007, pp. 84-97), we may consider that Eclipse IDE provides a powerful environment that encourages

modeling/metamodeling activities. The Eclipse Modeling Framework (EMF), which is part of the Eclipse IDE, is a framework developed by the Eclipse Foundation as the foundation of the Modeling Project (The Eclipse Foundation). The framework is a code generation facility for building tools and other applications based on a structured data model.

Figure 8

Enterprise Architecture Viewpoint



Note. Adapted from *ArchiMate® 2.1 Specification*. (n.d.). Retrieved December 20, 2020, from https://pubs.opengroup.org/architecture/archimate2-doc/chap08.html

3.5 Model-Driven Engineering for Sensor Networks

Mostly, SN system development methods concentrate on implementation problems and rarely rely on software engineering methodology that supports the entire life cycle of development. However, several recent research approaches to SN development in (Romer et al., 2002, pp. 59-61) (Masri & Mammeri, 2007, pp. 349-356) (Boonma & Suzuki, 2008, pp. 360-

367) tackle this problem, and most of these approaches focus on modeling applications at different abstraction levels with subsequent code generation as in MDE. MDE can contribute to the SN context by reducing the complexity of the design by enabling designers to model their systems at various levels of abstraction. It also provides designers with automated model transformations to turn abstract models such as XMI files into concrete models such as C++ or Java (Losilla et al., 2007, p. 179-194) (Schmidt, 2006).

For this reason, Losilla et al. (2007) and Schmidt (2006) demonstrated the use of MDE to model the SN life cycle. The model-driven performance engineering framework is also discussed in (Boonma & Suzuki, 2010, pp. 1674-1690). It is also useful for SN designers to use and implement the Model-Driven Engineering (MDE) to model SN.

Based on the discussed MDE concepts, model transformation, and the definition of DSML using the meta-model approach, an existing design tool can be extended by adding new SN concepts and relationships. The extension can be achieved by adding new components and constraints to the initial metamodels, then generating a new design tool that contains the concrete syntax. This concrete syntax includes elements and relationships to be used by SN designers during design time.

MDE offers many advantages when used in the SN development process. Since MDE focuses on creating domain models, Losilla et al. (2007) propose a SN application development technique by building a model for the targeted system using the SN DSML. The metamodel of the modeling language contains all the necessary concepts to build SN applications; this is a benefit since SN designers can use domain-specific concepts and relationships during the design phase of the SN development life cycle. Another advantage offered is a new graphical modeling

editor that enables SN domain experts to graphically explain the structure and behavior of their systems that are built based on the SN metamodel (Losilla et al., 2007, p. 179-194).

The advantages provided by MDE to SN are:

1- Ability to have DSML with specific SN concepts and relationships by adopting the metamodeling approach. Thus, to facilitate the modeling task of the SN designer, these components and relationships can be added into the targeted framework.
2- Diversity of SN elements and connections within the same model from different viewpoints and DSML by adopting the model heterogeneity aspect. This benefit makes it possible to provide a cohesive model that involves interrelationships between the various viewpoints according to 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 to verify the created models.

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

3.6 Discussion

MDE is required to contribute to meeting the following requirements, previously discussed in section “Requirements for Designing Sensor Networks Systems”, based on presented MDE benefits:

1- Improving Architectural Design (Requirement 1): Specific concepts in the IT domain are required to develop an SN model. This domain is too wide and involves a large number of complex concepts. It includes various types of devices for exchanging data between these devices, with different operating systems and communication protocols. For this reason, by following a certain approach that enables the SN designer to reuse some existing specific IT concepts, we can avoid developing such complex concepts. Existing metamodels can be expanded with new SN definitions, relationships, and constraints through the metamodeling approach to the language definition, to describe the syntax of a particular SN DSML. Through invoking the implementation of the constraints, the architectural design mistakes that might be made by the SN designer can be avoided.
2- Extensibility (Requirement 3): The SN designer needs a specific design tool that includes the current specific IT concepts and the new extended SN concepts, relationships, and constraints to model an SN model. Implementing these new SN principles in a design tool is a difficult task. For this reason, to facilitate the implementation of new SN concepts, we can follow a certain approach. The extensibility of the concepts appears via the model transformation method by generating a new design tool automatically. It enables the creation of a new design tool containing the new elements, relationships, and constraints that have been introduced.
3- Heterogeneity Supported (Requirement 4): Through the metamodeling strategy for language definition, we can have various components and types of relationships linked to different contexts and activities. These extended components could be software and hardware.

In conclusion, the implementation of new extended concepts and constraints strengthens architectural design and enforces constraints. This prevents SN designers from making errors. Therefore, the metamodeling approach for language definition deals with the first requirement (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 a model from several heterogeneous models (different viewpoints). This is to ensure the ability for the SN designer to model complex systems and satisfy the third and fourth requirements (Extensibility and Heterogeneity Supported).

Choosing the correct Modeling Language specific to our SN domain is one of the key options for implementing the MDE approach. As far as the application domain is concerned, general-purpose languages are too far away from our specific concepts. Thus, a DSML with the appropriate concepts and its associated tools may be the most relevant language. However, to cover all the functions of SN systems, DSML development, and its tooling is a huge challenge.

Therefore, to optimize the development life cycle, we try to find a DSML that is close to our context. To conform to our SN domain with precision, we may apply a metamodel specialization. The next chapter introduces modeling languages and their tooling relevant to information systems based on the network infrastructure to overcome this constraint.

Chapter 4: System Architecture Modeling

4.1 Modeling Context

Different experts should be involved in the modeling process to cover the sensor integration in the IT system when designing a SN system. In such systems, the sensors are linked to dedicated algorithms and IT infrastructure to provide users of SN with high-level services.

Experts with different domains of experience, such as business or technical, are required to address different viewpoints when designing an SN distributed system. The problem of the modeling process is, therefore, to generate the appropriate models according to the necessary points of view for distributed network infrastructure applications. Therefore, it is essential to model many perspectives to identify and select the mapping of the software application on given network architecture.

Designers need a collection of standardized and domain-specific concepts, provided mainly by frameworks, to construct viewpoints. They also need design tools that can define a model using various viewpoints to handle the second requirement (Multiple Viewpoints), as previously mentioned in section “Requirements for Designing Sensor Networks Systems”. This can be provided by Enterprise Architecture (EA) frameworks that are based on domain-specific concerns. Each viewpoint is defined by its concepts and then aggregated by a framework in a modeling language. To this end, to choose the most suitable one, we are interested in presenting and researching the current EA frameworks.

4.2 Enterprise Architecture Types

Enterprise Architecture is: "The organizing logic for business and IT infrastructure reflecting the integration and standardization requirements of the firm’s operating model" (Ross, 2006). The EA model, therefore, is about splitting a model into many interrelated models, such

as business and IT models. Each model is a set of related elements and describes the activities and actions of a specific domain of experience.

According to ISO 15704, there are two types of EA: (1) EA dealing with the design of a system, called System Architecture; and (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 that target at structuring the required concepts and tasks to design and build complex systems such as SN. According to a survey of EA in (Chen et al., 2008, pp. 647-659), Enterprise-Reference Projects are the most adopted and used to build such systems. For this purpose, we present some of the Enterprise Architecture Frameworks in the next section.

4.3 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 (Cameron & McMillan, 2013, pp. 60-71). The framework is built to support the communication between the different stakeholders with different domains of experience, within the same model, by providing specific relations (Cameron & McMillan, 2013, pp. 60-71). This framework also enables a wide range of domains to be represented, it suits the problem of our SN modeling process by (Chiprianov, 2012): (1) generating relevant models according to the different fields of experience that are divided into different viewpoints; and (2) providing the ability to connect these models using specific relationships. The Five major EA Frameworks that can provide the features listed above are the Zachman Framework, the Open Group Architecture Framework (TOGAF), the Federal

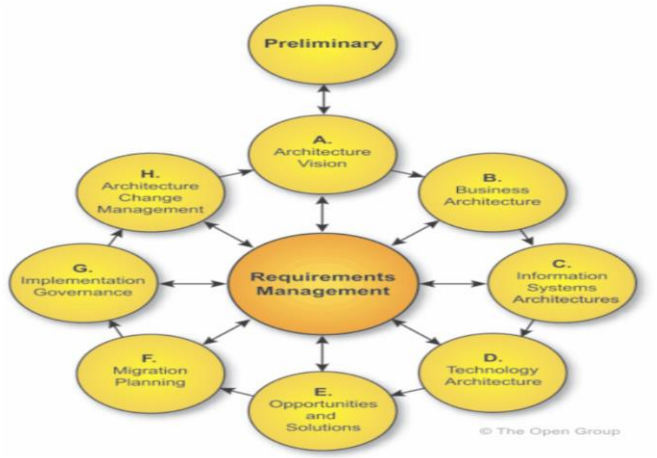
Enterprise Architecture Framework (FEAF), the Department of Defense Architecture Framework, and Gartner Framework (Cameron & McMillan, 2013, pp. 60-71).

Therefore, based on previous studies in (Chiprianov, 2012) (Cameron & McMillan, 2013, pp. 60-71) (Fatolahi & Shams, 2006, pp. 133-143), TOGAF is the most useful framework for (1) building a model from various viewpoints; (2) interrelating business and technical viewpoints; and (3) detailing the technical viewpoint as it is necessary to build complex systems.

The Open Group Architecture Framework (TOGAF) describes a method called the Architecture Development Method (ADM) to design an enterprise architecture as part of its core. Figure 9 illustrates all the phases of ADM in sequential order from planning to implementation. Also, as can be seen in Figure 9, ADM is an iterative process, which satisfies the first requirement of the SN Design (Improving Architectural Design) for detecting and minimizing architectural design errors during the design phase.

Figure 9

Architecture Development Method



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

4.4 Domain-Specific Concepts in Enterprise Architecture Frameworks

Designers need modeling languages to construct models. UML and SysML are existing modeling languages to design efficient general-purpose applications without offering domain-specific concepts. Therefore, to create domain-specific models, designers should use modeling languages that contain specific components related to this domain.

We are interested in defining models for a specific domain, which is the SN, according to our context. Therefore, designers need dedicated SN modeling languages to define such models when using adopted frameworks such as TOGAF. This is due to the complexities of having unique SN concepts in the general-purpose modeling languages. Such concepts are the initial step of Domain-Specific Modeling Language development. Also, the adopted framework recommends the use of DSML to fulfill the requirements of the modeling task in a particular domain.

An SN DSML should be defined accordingly, based on our SN context. This definition should take the needs of the SN designer into account. These needs are concepts, relationships, and constraints in the SN domain. The designer also needs to define a SN model from different viewpoints according to different layers. Thus, building such models is a challenging task. Therefore, existing metamodels can be extended by adding new concepts. This extension enables SN designers to reuse these concepts in the modeling task.

A key question can therefore be asked here: What are the existing metamodels that can be extended to define an SN DSML that meets the above requirements? The ideal answer to this question is to find an existing metamodel that incorporates concepts of IT and SN that allows the designer to create a model of SN from various viewpoints.

We distinguish several SN metamodels such as SensorML (Chandrasekaran et al., 2014), ThingML (Fleurey, 2011, pp. 349-363), Deep-Sea Observatory metamodel (Champeau et al., 2009, pp. 1-6), Heterogeneous Sensor Web Node MetaModel (Chen et al., 2014, pp. 222-237), Wearable Markup Language (Fortino et al., 2014), SUM MetaModel (Burger, 2014), and GINPEX Sensor MetaModel (Hauck, 2014). 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 and ArchiMate rely on EA. Therefore, we elaborate on TOGAF and ArchiMate, in the next sections.

4.5 Enterprise Architecture Modeling Languages and Metamodels

EA metamodels are the abstract syntax of EA modeling languages. Therefore, the use of the EA Modeling Language is needed to model a complex system by adopting the EA Framework. EA Modeling Language is a conceptual or logical representation of EA with a high abstraction level.

ArchiMate and TOGAF are EA Modeling Languages based on concepts identified by EA Frameworks such as TOGAF (Noran, 2003, pp. 163-183). These EA Modeling Languages are defined by EA metamodels that define the concepts, relationships, and constraints necessary to construct models.

4.5.1 ArchiMate

ArchiMate breaks down the system design into three layers: business layer, application layer, and technology layer. It ensures interoperability between these layers. These layers are described as follow:

1- Business layer: defines the actions, functions, and the exchange between these two.
2- Application layer: defines the way of performing the actions and functions provided in the upper 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.

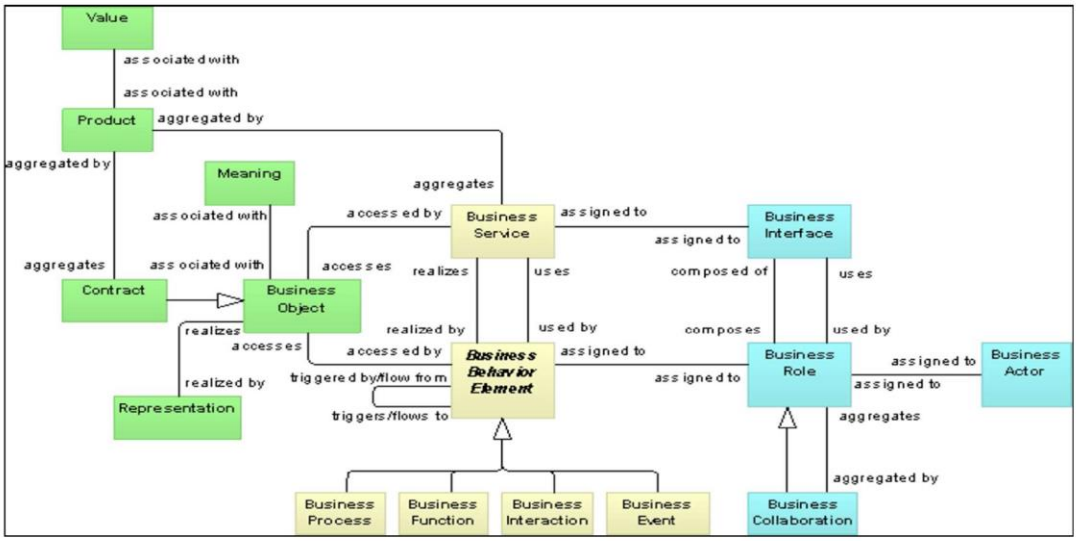
A metamodel defines each layer of ArchiMate. According to (Pérez-Medina & Dupuy-Chessa, 2007, pp. 84-97), a metamodel defines a language for describing a specific domain of interest. Specific relationships interrelate the three different metamodels (The Open Group ArchiMate(R)). Figures 10, 11, and 12 represent respectively the business, application, and technology layers of ArchiMate. These metamodels are the abstract syntax of ArchiMate.

A textual or graphical user interface represents the concrete syntax of ArchiMate. Figures 13 and 14 represent the graphical interface related to the abstract concepts of the business layer of ArchiMate. For example, the Business Object entity in the concrete syntax, as can be seen in Figure 13, represents the Business Object in the abstract syntax, as can be seen in Figure 10. The semantics describes the meaning of each concept in the concrete syntax.

ArchiMate determines functioning relationships between two contiguous layers (Noran, 2003, pp. 163-183) (Quartel et al., 2009, pp. 3-13). Figure 15 represents the relationship between the business and application layers. Figure 16 represents the relationship between the application and technology layers. According to our previously presented modeling context, relevant models can be created by relying on ArchiMate metamodel. The interoperability between ArchiMate layers allows communication between different created models by exchanging information.

Figure 10

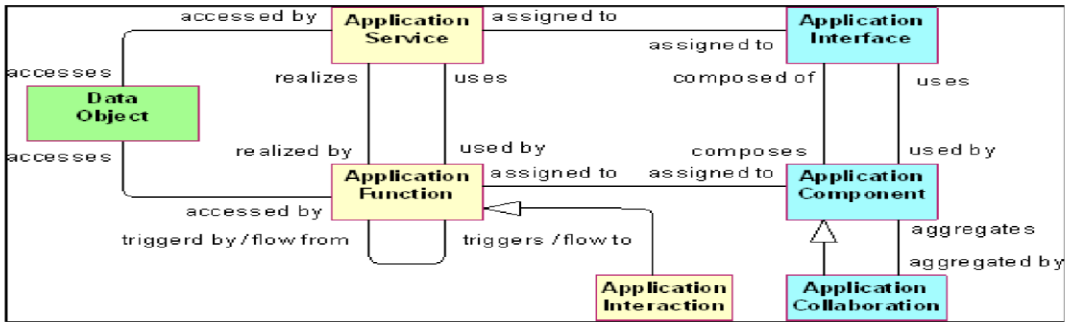
ArchiMate Business Layer Metamodel



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

Figure 11

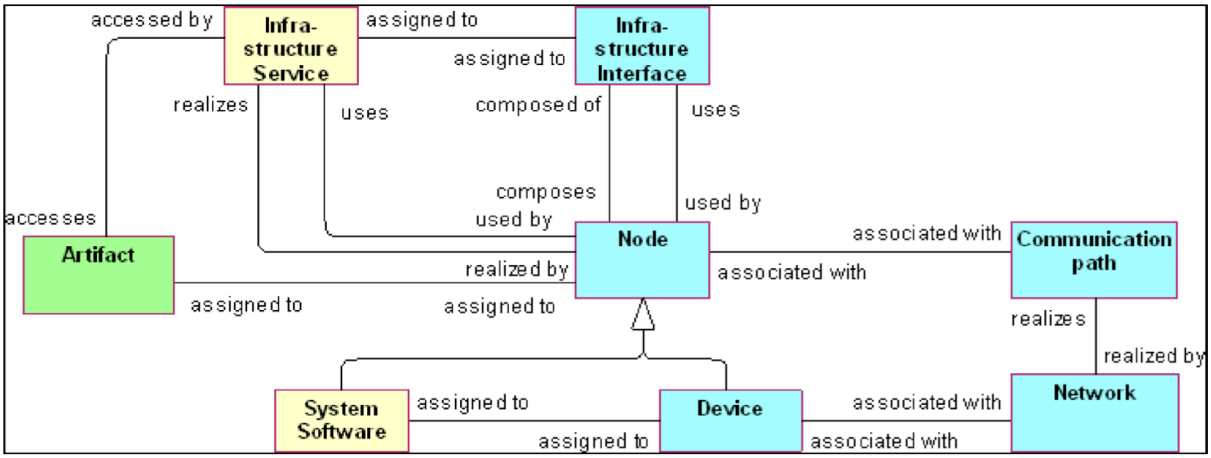
ArchiMate Application Layer Metamodel



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

Figure 12

ArchiMate Technology Layer Metamodel



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

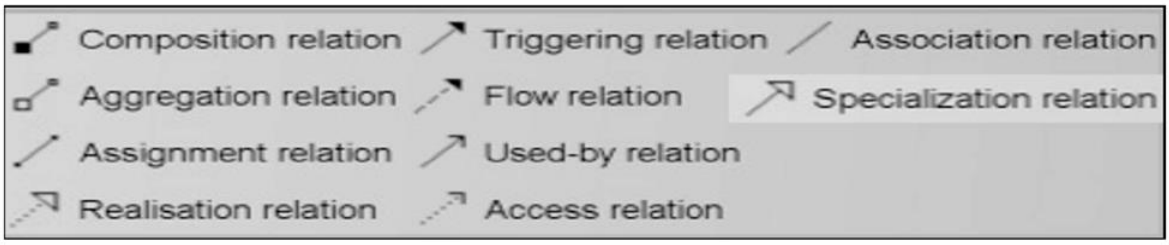
Figure 13

ArchiMate Business Layer Concrete Syntax Components

Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

Figure 14

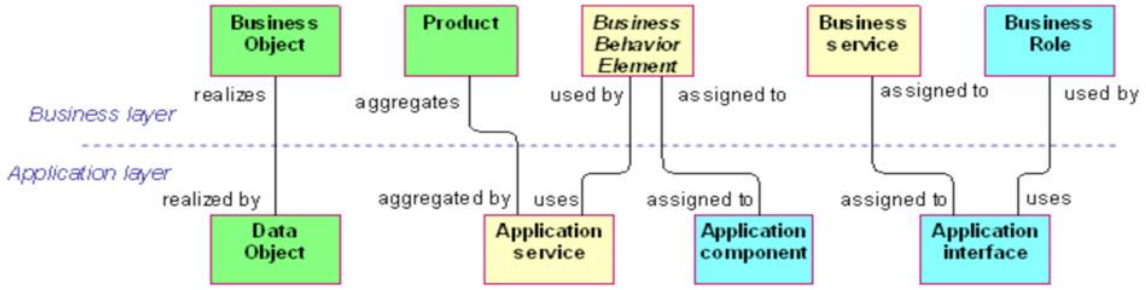
ArchiMate Business Layer Concrete Syntax Relationships



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

Figure 15

ArchiMate Business-Application Alignment



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

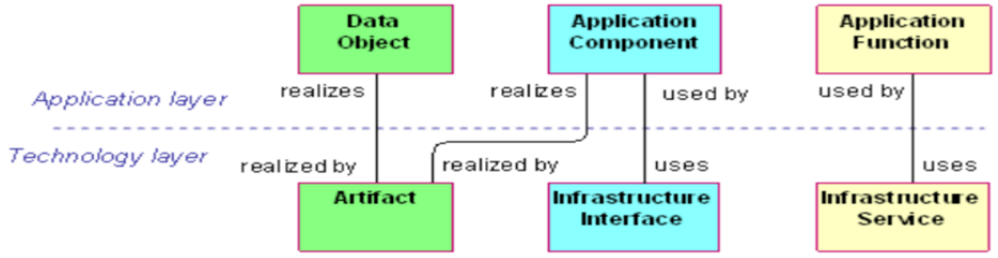
4.5.2 Togaf

TOGAF, as illustrated in Figure 17, decomposes the system design into three layers: the business architecture layer, the information system layer, and the technology architecture layer. It ensures interoperability between different layers (The Open Group ArchiMate(R)). The layers of TOGAF are defined by a metamodel, where the latter defines by itself, a language for describing

a specific domain of interest (Pérez-Medina & Dupuy-Chessa, 2007, pp. 84-97). The metamodels are interrelated by specific relationships (The Open Group ArchiMate(R)).

Figure 16

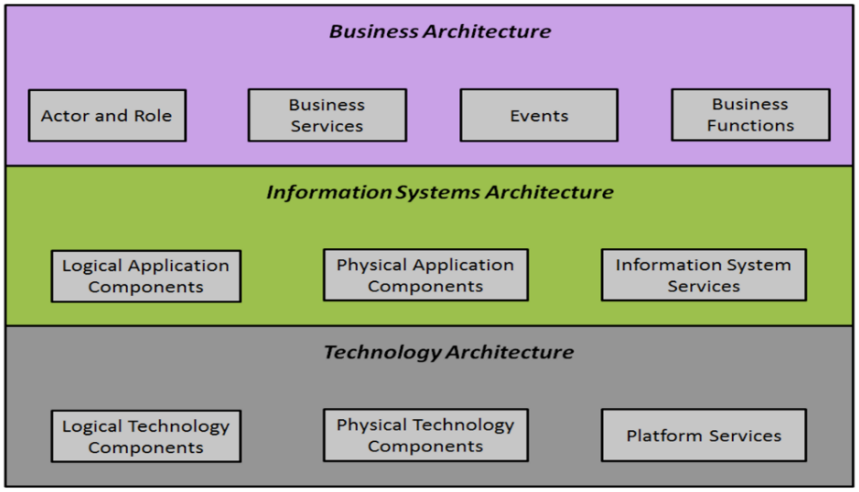
ArchiMate Application-Technology Alignment



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

Figure 17

Layers of TOGAF Metamodel



Note. Adapted from *The Open Group ArchiMate(R) 1.0 Technical Standard*. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

4.6 Requirements for Selecting the Enterprise Architecture Metamodel

To define an SN DSML, we have to expand the metamodel of one of the previously discussed EA modeling languages, ArchiMate and TOGAF. For this reason, we are interested in identifying the requirements to select the appropriate metamodel:

1- **Requirement 1** Several Viewpoints: Each designer should be able to work independently on a model according to his area of experience to address different viewpoints.

2- **Requirement 2** Separate Logical and Physical Views: Software designers should be able to define logical models in a separate view to describe logical components. Network designers should be able to define physical models in another view to describe physical components. Several designers, with different domains of experience, should work on different views.

3- **Requirement 3** Consistency Supported: Each created model should be able to interoperate with other models. The modeling language should provide communication between different layers to generate one consistent model from different viewpoints.

4- **Requirement 4** Specific IT Components: A network designer should be able to use built-in IT components from the generated design tool. For example, the designer can use devices such as clients or servers, components, and relationships to connect devices using specific relationships such as communication path.

4.7 Comparison Among Enterprise Architecture Metamodels

We discuss the comparison between ArchiMate and TOGAF based on the requirements discussed in the previous section:

1- **Requirement 1:** Both metamodels take into consideration several viewpoints when creating multiple models according to different domains of experience.
2- **Requirement 2**: ArchiMate deals with the separation of logical and physical views within the same viewpoint; this is due to the three separate layers that are provided by ArchiMate: business, application, and technology. The application layer contains only logical components, while the technology layer only contains physical components. However, in TOGAF, both application architecture and technology architecture contain logical and physical application components at the same time.
3- **Requirement 3**: ArchiMate and TOGAF allow the interoperation between different models. They provide the ability to have one consistent model that contains components and relationships from different viewpoints at different layers.
4- **Requirement 4** Specific IT Components are available in both TOGAF and ArchiMate metamodels.

Figure 18 illustrates the result of the comparison between ArchiMate and TOGAF. Also, both metamodels are almost similar. Thus, ArchiMate and TOGAF share similar concepts so they can be used together, as can be seen in Figure 19. Furthermore, ArchiMate provides a concrete syntax with a graphical user interface. For this purpose, ArchiMate metamodel can be extended by adding new SN concepts, relationships, and constraints.

Accordingly, to build SN, the designers can adopt TOGAF as a framework and ArchiMate as a modeling language that relies on EA. However, we distinguish several frameworks and design tools that are dedicated to building SN. Therefore, to make 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.

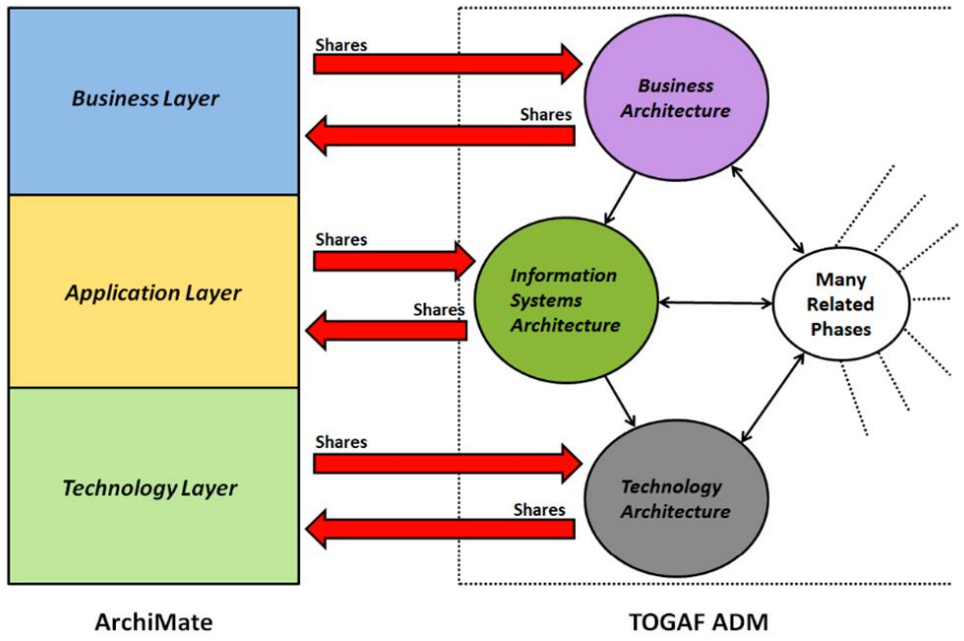
Figure 18

Comparison Between ArchiMate and TOGAF

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

Figure 19

Compatibility Between TOGAF ADM and ArchiMate



4.8 Enterprise Architecture Frameworks and Design Tools for Sensor Networks

A framework is a set of functions and libraries to model applications from different domains. Several frameworks provide design tools. Design tools enable the designers to create analysis and design models of the system to be built and ensure consistency between models. We differentiate design tools by their provided features. Thus, the choice of the design framework is a difficult task.

Some studies and recent researches have focused on SN frameworks. These frameworks provide graphical interfaces for heterogeneous sensors and actuators and ease their deployment and management (Grabis & Kirikiva, 2011). Examples of these frameworks are Global Sensor Networks (GSN), Sensor Web Enablement (SWE) (Funk et al., 2011), SENSEI (recent European research project) (Luo, 2013). The mentioned frameworks are not useful for our context since the designer requires to adopt a framework and design tool that includes a DSML that contains IT and SN concepts that enable the designer to build an SN model from different viewpoints.

Several frameworks are proposed to support the management of enterprise IT by describing the systems from the IT domain, using an EA modeling language (Meyer et al., 2011, pp. 167-177) such as the Enterprise Architecture Framework, TOGAF. These IT frameworks address a wide range of domains and technologies as they allow different stakeholders to describe a system according to different domains of experience. Thus, each stakeholder creates his proper model according to his viewpoint. Therefore, TOGAF and ArchiMate can be adopted by the designers to build SN.

4.9 Discussion

With regard to the previously presented EA features for SN, the advantages of ArchiMate, and the need to incorporate new SN concepts into the EA Framework, the extension of ArchiMate is

a suitable contribution to defining the Domain-Specific Modeling Language (DSML) for SN. The DSML should contain new SN concepts and constraints that are inherited from the original ArchiMate metamodel.

EA provides the below advantages for SN:

1- Achieving the right balance between IT efficiency and activities at a high abstract level. It helps SN designers to build their models, such as any data fusion architecture or network topology, as it ensures the needs of the models generated for an integrated IT strategy. SN designers are capable of modeling any complex system without thinking about the availability of IT components.
2- Reducing the deployment uncertainty of the SN model. This is due to the existence of various layers and stakeholders, where each stakeholder is an expert in his area. And because of the interoperability between different layers to provide a single consistent model.

Relying on the identified EA advantages, EA is expected to contribute toward satisfying the second requirement (Multiple Viewpoints). EA provides the ability for the SN designers to create several models according to their viewpoints and domain of experience. Also, it provides the ability to interrelate different models to have one overall and consistent model.

In conclusion, we investigated the use of MDE approach for sensor networks with an Enterprise Architecture Framework to address all the requirements for designing SNs systems. To improve the development life cycle of SN, we state that we need to extend an existing modeling language by adding new concepts.

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

5.1 ArchiMO Definition

Some of the existing ArchiMate concepts and relationships can be used while constructing a SN model for MO. These concepts are not domain-specific to meet the Marine Observatory (MO) requirements in the design process. This is because not all ArchiMate components can be used to construct models for specific domains, such as MO. For this purpose, researchers in the discipline of computing have extended ArchiMate metamodel to generate a new design tool, ArchiMO, while taking into consideration specific Marine Observatory components (Aoun et al., 2015). Like any other DSML, ArchiMO involves an abstract syntax, a concrete syntax, and semantics (Cho et al., 2012, pp. 22-28).

5.1.1 Selected ArchiMate Concepts and Relationships

To build a consistent model that reflects a real description of detecting underwater objects, MO designers require software and hardware concepts to be used in the design phase. These concepts are elaborated below according to the ArchiMate business and application layers.

5.1.1.1 Business Layer. To define a MO model in the ArchiMate business layer, the domain expert requires behavioral concepts, structural concepts, and relationships. The role of structural concepts is to perform behavioral concepts by using structural and dynamic relationships while taking into consideration MO constraints. The structural concepts are Smart Sensors (SS) and Data Fusion Servers (DFS). The behavioral concepts are Algorithm Selection (AS), Data Transmission (DT), Data Acquisition (DA), and Object Localization Algorithm (OLA). The structural relationships are Assignment, Association, and Used By. The dynamic relationship is triggering (Aoun et al., 2015).

Since MO constraints should be applied when defining models, the predefined ArchiMate concepts and relationships are not enough. Some predefined structural and behavioral concepts can be used, such as business actors and business functions. However, these concepts do not have the required MO constraints that should be applied when defining the model. For this reason, the business actor and business function are extended to include MO constraints (Aoun et al., 2015).

5.1.1.2 Application Layer. To define a MO model in the ArchiMate application layer, the software designers require behavioral concepts, structural concepts, and relationships. The structural concepts are Smart Sensor Systems (SSS), and Fusion Systems (FS) that are mapped to the Smart Sensors (SS) and Data Fusion Servers (DFS) from the business layer respectively. The behavioral concepts are Manage Resources (MR), Coordinates Storage Handling (CSH), Compute Coordinates (CC), Transmit Localization Data (TCD), Inform Server (IS), Voice Streaming, and Video Streaming.

The discussed concepts are not available in ArchiMate application layer. Some predefined concepts can be used, such as the application component. However, these concepts do not have the required MO constraints that should be applied when defining the model. For this reason, the application component and application function are extended to include MO constraints (Aoun et al., 2015).

5.1.2 ArchiMO Metamodel

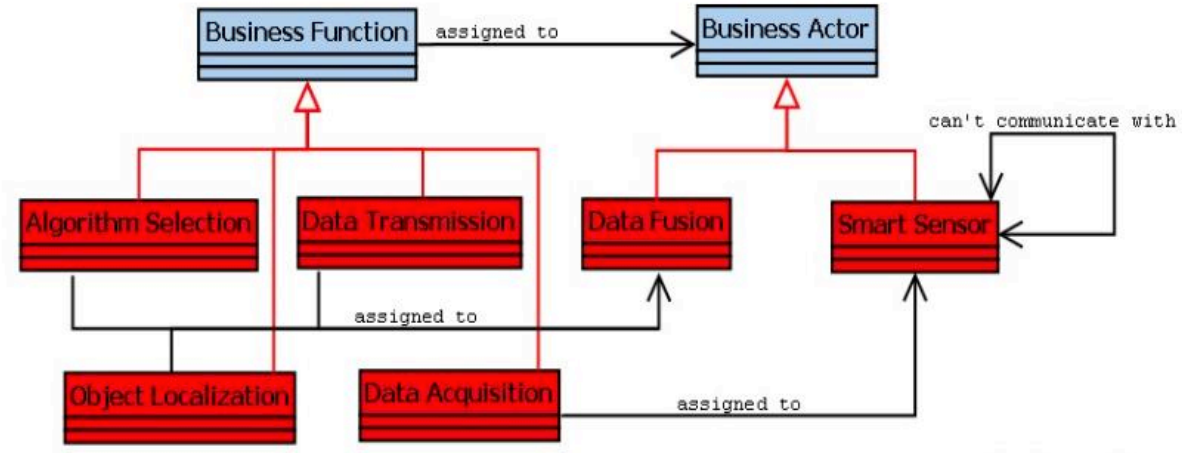
As previously discussed, ArchiMate is composed of business, application, and technology layers. As per our scope, we are interested in extending the business and application layers only. The technology layer of ArchiMate is extended by (Chiprianov, 2012) to define a DSML for IP Multimedia Subsystems (IMS) in the telecommunication domain. This DSML can

be used for different types of applications within various domains including the MO (Alloush, 2016). For example, IMS allows the exchange of messages between terminals such as smart sensors. Therefore, IMS metamodel can be used to model the deployment of our application on technical infrastructure.

Figures 20 and 21 (Aoun et al., 2015) represent part of the ArchiMate business layer and application layer metamodels. The concepts in white are those predefined by ArchiMate, while the concepts in green are those related to the MO domain. Figure 22 (Aoun et al., 2015) illustrates the communication constraint between two sensors because two sensors cannot communicate together.

Figure 20

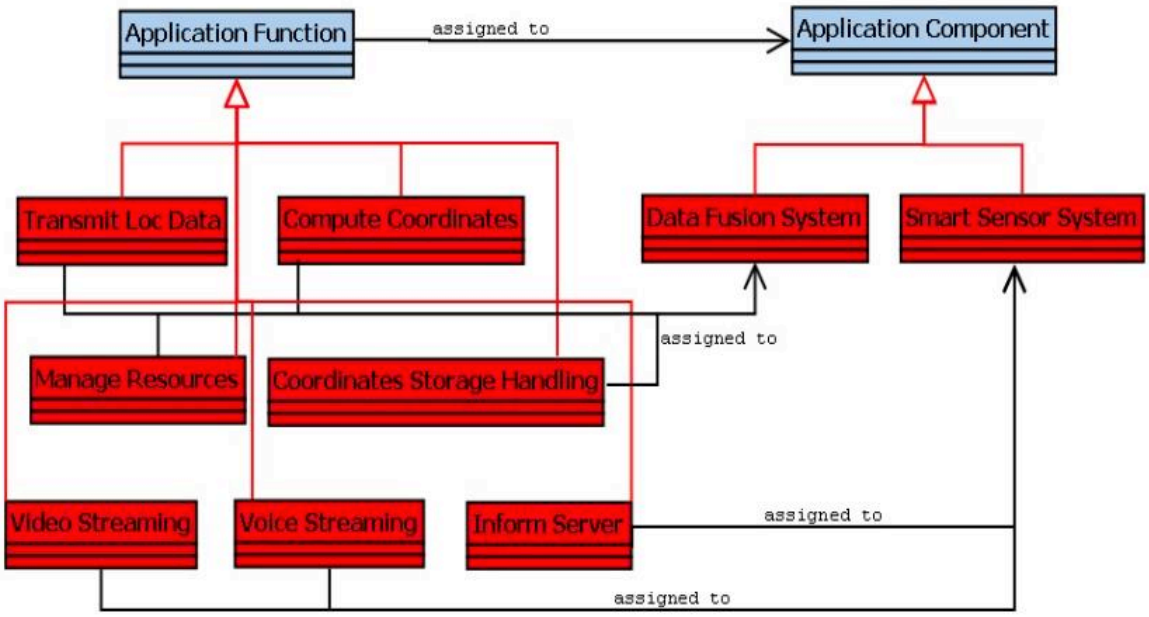
ArchiMate Extended Business Layer



Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

Figure 21

ArchiMate Extended Application Layer



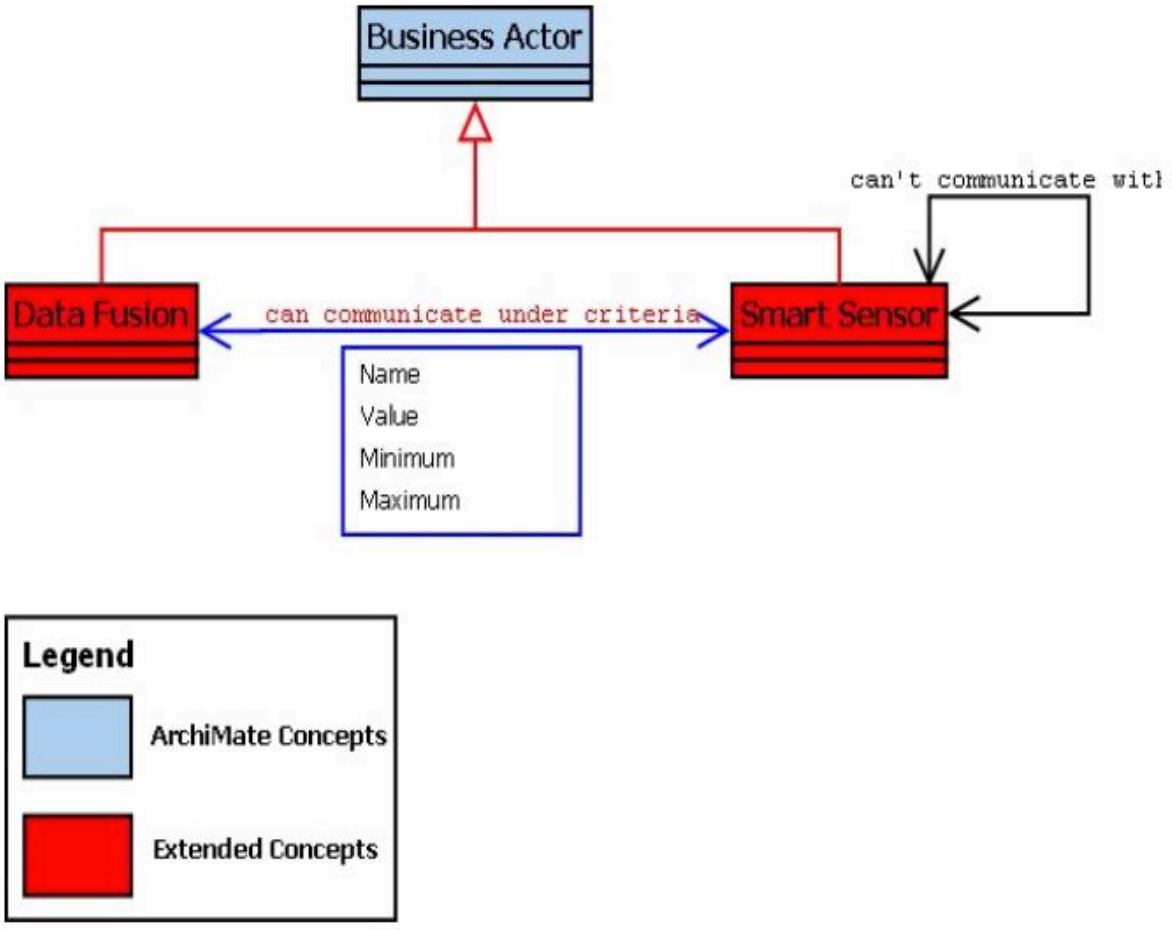
Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

5.1.2.1 Concepts. ArchiMO is composed of business and applications layers. The metamodels of these layers are described as follow (Aoun et al., 2015):

1- Business Layer (Figure 20): The Business Actor of ArchiMate is extended with two new concepts, Smart Sensor and Data Fusion. The Smart Sensor is responsible for Data Acquisition. The Data Fusion is responsible for: (1) Algorithm Selection, to select the proper algorithm in case there are several algorithms with different functions; (2) Data Transmission, to transmit data between different Data Fusion components; and (3) Object Localization, to call the localization algorithm.

Figure 22

Communication Constraint Between Smart Sensor-Data Fusion



Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

2- Application Layer (Figure 21): The Application Component of ArchiMate is extended with two new concepts, Smart Sensor System and Data Fusion System. The Smart Sensor System is responsible for: (1) Inform Server, to inform the fusion server about the detection of an object; (2) Voice Streaming, in case the Smart Sensors are hydrophones; and (3) Video Streaming, in case the Smart Sensors are underwater cameras. The Data

Fusion System is responsible for: (1) Manage Resources, to manage the resources needed for the algorithm execution; (2) Coordinates Storage Handling, to store the coordinates correlated with time. The DFS handles the storage of the coordinates received by each Smart Sensor; (3) Compute Coordinates, to compute the position by using the stored coordinates received by each Smart Sensor; and (4) Transmit Localization Data, to exchange information between different fusion servers.

5.1.2.2 Relationships. There are various types of relationships provided by ArchiMate, such as association and assignment. Assignment relationships are used to connect structural and functional elements. For example, function1 should be performed by node1 in case function1 is assigned to node1.

ArchiMate predefined relationships cannot satisfy the MO domain since they do not impose relevant constraints. For this reason, Aoun et al. (2015) extended ArchiMate relationships by adding new constraints. A new constraint is imposed on the association relationship for the smart sensor. For example, a smart sensor can only be associated to a data fusion, and two smart sensors cannot be associated to each other, as can be seen in Figure 22 (Aoun et al., 2015). Also, the assignment relationship is also extended. For example, a smart sensor can only be assigned to data acquisition, and a data fusion can only be assigned to algorithm selection, data transmission, and object localization functions. In addition, Aoun et al. (2015) introduced a new logical relationship to connect a smart sensor to a data fusion.

5.1.3 ArchiMO Design Tool

5.1.3.1 ArchiMO Concrete Syntax. A concrete syntax should be defined for every new concept defined in the abstract syntax. ArchiMO design tool contains the concrete syntax associated with the extended concepts and relationships. The right part of Figure 23 (Aoun et al.,

2015) illustrates the business concrete syntax of ArchiMO, and the left part illustrates the application concrete syntax. In addition, the relation between a smart sensor and data fusion is illustrated is Figure 24 (Aoun et al., 2015).

5.1.3.2 Constraints Implementation. As illustrated in Figure 25 (Aoun et al., 2015), all the extended constraints are checked while designing the model. Moreover, the relationship constraint between the smart sensor and data fusion is also checked, as can be seen in Figure 26 (Aoun et al., 2015). This relationship reflects the marine cable. The designer should enter a valid length for the cable.

The design of the application layer is automatically generated from the design of business layer. For example, when a smart sensor is connected to a data fusion in the business layer, the corresponding elements such as smart sensor system and inform server function are automatically reflected in the application layer.

5.2 Smart Sensor Specification

In terms of the specifications of the components, it should be noted that each underwater moving object generates its frequency tag. In fact, according to the National Oceanic and Atmospheric Administration (NOAA), aquatic acoustics are comprised of a wide range of frequencies with variations between and within species (Bittle & Duncan, 2013). Since MO projects are intended to monitor different underwater objects, this constraint should be taken into account when using and installing smart sensors in such projects (Nielsen et al., 2019). Failure to do so results in monitoring the incorrect object which could lead to substantial costs in both time and resources.

To prevent these losses in the deployment phase, we propose to extend ArchiMO business layer metamodel by adding a new constraint to the minimum and maximum frequencies at which a smart sensor operates.

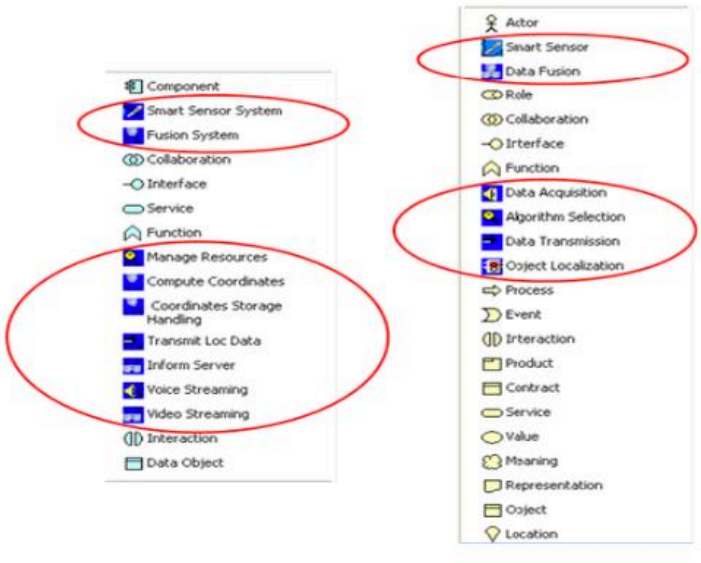
5.2.1 Smart Sensor Constraint Implementation

We extend ArchiMO metamodel by adding the frequency range constraint as a property of the smart sensor component, which answers our second research question. Therefore, as illustrated in Figure 27, a range of frequencies should be specified whenever the designer add a smart sensor component while building the model. Different smart sensors can operate at different ranges of frequencies. Figure 27 illustrates an example of the required frequencies to detect a dolphin; the smart sensor should be configured to receive frequencies between 0.2 and 170 KHz.

Accordingly, the proposed constraint reduces the possibility of deploying an inappropriate smart sensor.

Figure 23

Business and Application Layers Palette



Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

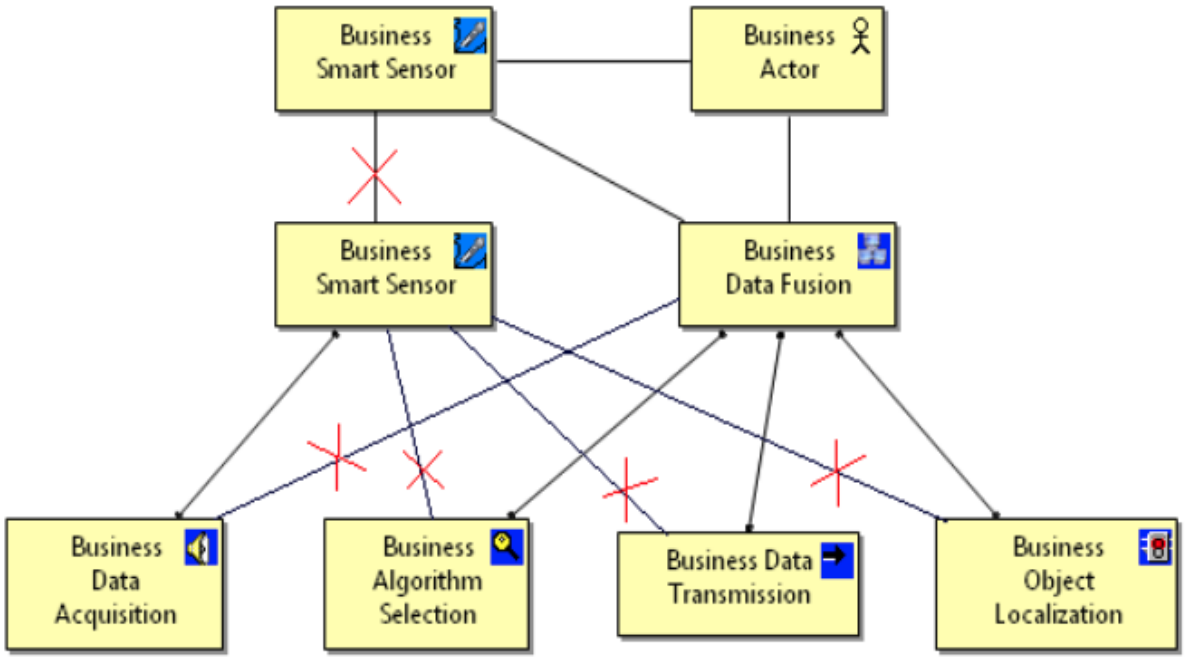
Figure 24

Extended Relationship in Palette

Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

Figure 25

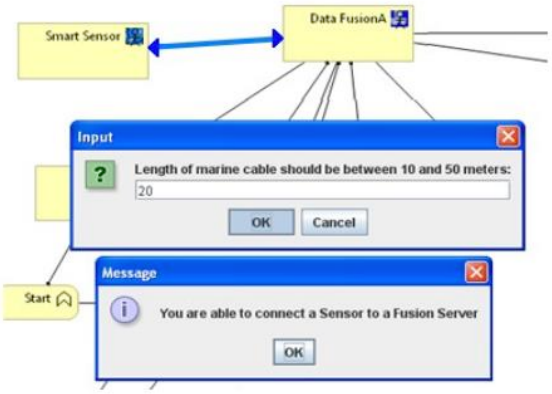
Association and Assignment Relationships



Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

Figure 26

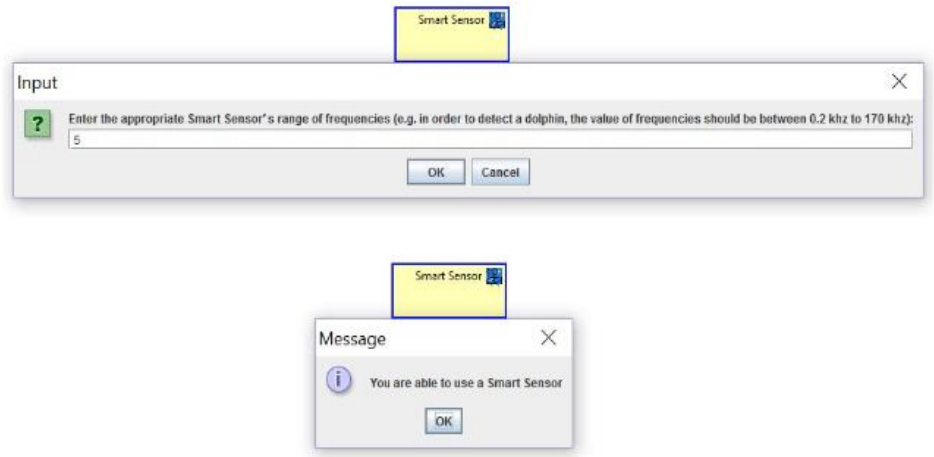
Smart Sensor – Data Fusion Relationship



Note. Adopted from Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

Figure 27

Smart Sensor Frequency



Note. Adapted from Aoun, C. G., Lagadec, L., Champeau, J., Moussa, J., & Hanna, E. (2017). A High Abstraction Level Constraint for Object Localization in Marine Observatories. *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 605–611.

Conclusion

Answering the Research Questions

Underwater sensor networks are complex systems that 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 sensor network development and deployment phases.

We have defined ArchiMO, a design tool introduced by researchers, which is used to build sensor networks models for MO. This tool is a Domain-Specific Modeling Language that extends ArchiMate, an Enterprise Architecture Modeling Language. ArchiMO extends the business and application layers of ArchiMate. It helps the designers to prevent architectural design errors that can be made during design time.

The research questions were as below:

1- How to prevent errors at design time by adding a new environmental constraint?

2- How to implement the proposed constraint in an existing design tool?

To answer the first research question, we proposed a new domain-specific constraint that deals with the minimum and maximum frequencies at which a smart sensor operates. And to answer the second research question, we implemented this constraint in an existing metamodel that suits our context. At each time the designer uses a smart sensor from ArchiMO, he should specify its frequency. By extending a Domain-Specific Modeling Language for MO, designers are able to prevent architectural design errors that can be made during design time.

By using a Domain-Specific Modeling Language, with specific domain constraint such as the sensor frequency, designers are able to prevent architectural design errors that can be made during the design phase. It also supports designers to build models from different viewpoints. In

addition, it ensures consistency between different models since all ArchiMate layers are interrelated.

By extending ArchiMO design tool with a new environmental constraint, the designers are able to build more consistent models using the extended constraint by specifying the frequency supported by each sensor.

Recommendations and Future Work

Enterprise Architecture Modeling Languages offer many advantages when designing sensor networks. In our case, it was used to design models related to the marine observatory sub-domain. But it can also be used in other sub-domains under the sensor network domain or even to design other complex systems related to other domains. The metamodel of these modeling languages can be extended and new design tools can be generated since they are based on the Model-Driven Engineering methodology.

We are looking forward to add new environmental constraints, by the help of domain experts, to the design tool. All these constraints should be taken into consideration while designing a sensor network for MO. Design constraints can support designers and facilitate the validation process. Also, we will try to generalize the extended concepts and constraints to cover more domains.

References

Achilleos, A., Yang, K., & Georgalas, N. (2010). Context modelling and a context-aware framework for pervasive service creation: A model-driven approach. *Pervasive and Mobile Computing*, *6*(2), 281–296.

Ahmed, A., Ali, J., Raza, A., & Abbas, G. (2006). Wired vs wireless deployment support for wireless sensor networks. *TENCON 2006-2006 IEEE Region 10 Conference*, 1–3.

Alloush, I. (2016). *A design and verification framework for telecommunication services* [PhD Thesis].

Alriksson, P., & Rantzer, A. (2007). Experimental evaluation of a distributed Kalman filter algorithm. *2007 46th IEEE Conference on Decision and Control*, 5499–5504.

André, C. (2013). *Approche crédibiliste pour la fusion multi capteurs décentralisée* [PhD Thesis]. Paris 11.

Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A modeling approach for marine observatory. *Sensors & Transducers*, *185*(2), 129.

Aoun, C. G., Lagadec, L., Champeau, J., Moussa, J., & Hanna, E. (2017). A High Abstraction Level Constraint for Object Localization in Marine Observatories. *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 605–611.

ArchiMate® 2.1 Specification. (n.d.). Retrieved December 20, 2020, from https://pubs.opengroup.org/architecture/archimate2-doc/chap08.html

Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., & Valls, M. (2005). Sensor networks and distributed CSP: Communication, computation and complexity. *Artificial Intelligence*, *161*(1–2), 117–147.

Bittle, M., & Duncan, A. (2013). A review of current marine mammal detection and classification algorithms for use in automated passive acoustic monitoring. *Proceedings of Acoustics*, *2013*.

Boonma, P., & Suzuki, J. (2010). Moppet: A model-driven performance engineering framework for wireless sensor networks. *The Computer Journal*, *53*(10), 1674–1690.

Boonma, P., & Suzuki, J. (2008). Middleware support for pluggable non-functional properties in wireless sensor networks. *2008 IEEE Congress on Services-Part I*, 360–367.

Boukerche, A., Oliveira, H. A., Nakamura, E. F., & Loureiro, A. A. (2008). Vehicular ad hoc networks: A new challenge for localization-based systems. *Computer Communications*, *31*(12), 2838–2849.

Burger, E. (2014). *Flexible views for view-based model-driven development* (Vol. 15). KIT Scientific Publishing.

Cabot, J., & Teniente, E. (2007). Transformation techniques for OCL constraints. *Science of Computer Programming*, *68*(3), 179–195.

Caiti, A., Garulli, A., Livide, F., & Prattichizzo, D. (2005). Localization of autonomous underwater vehicles by floating acoustic buoys: A set-membership approach. *IEEE Journal of Oceanic Engineering*, *30*(1), 140–152.

Cameron, B. H., & McMillan, E. (2013). Analyzing the current trends in enterprise architecture frameworks. *Journal of Enterprise Architecture*, *9*(1), 60–71.

Castanedo, F. (2013). A review of data fusion techniques. *The Scientific World Journal*, *2013*.

Chandrasekaran, S., Choi, E., Abawajy, J. H., & Natarajan, R. (2014). Sensor Grid Middleware Metamodeling and Analysis. *International Journal of Distributed Sensor Networks*, *10*(4), 805708.

Chen, D., Doumeingts, G., & Vernadat, F. (2008). Architectures for enterprise integration and interoperability: Past, present and future. *Computers in Industry*, *59*(7), 647–659.

Chen, N., Wang, K., Xiao, C., & Gong, J. (2014). A heterogeneous sensor web node meta-model for the management of a flood monitoring system. *Environmental Modelling & Software*, *54*, 222–237.

Chiprianov, V. (2012). *Collaborative construction of telecommunications services. An enterprise architecture and model driven engineering method* [PhD Thesis].

Cho, H., Gray, J., & Syriani, E. (2012). Creating visual domain-specific modeling languages from end-user demonstration. *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, 22–28.

Choi, B.-S., & Lee, J.-J. (2010). Sensor network-based localization algorithm using fusion sensor-agent for indoor service robot. *IEEE Transactions on Consumer Electronics*, *56*(3), 1457–1465.

Chong, C.-Y., & Mori, S. (2005). Distributed fusion and communication management for target identification. *2005 7th International Conference on Information Fusion*, *2*, 8-pp.

Cuzzocrea, A. (2009). *Intelligent techniques for warehousing and mining sensor network data*. IGI Global.

El Zoghby, N. (2014). *Fusion distribuée de données échangées dans un réseau de véhicules* [PhD Thesis]. Université de Technologie de Compiègne.

Erol, M., Vieira, L. F. M., & Gerla, M. (2007). AUV-aided localization for underwater sensor networks. *International Conference on Wireless Algorithms, Systems and Applications (WASA 2007)*, 44–54.

Fatolahi, A., & Shams, F. (2006). An investigation into applying UML to the Zachman framework. *Information Systems Frontiers*, *8*(2), 133–143.

Fernández-Isabel, A., & Fuentes-Fernández, R. (2015). Analysis of intelligent transportation systems using model-driven simulations. *Sensors*, *15*(6), 14116–14141.

Fleurey, F., Morin, B., Solberg, A., & Barais, O. (2011). MDE to manage communications with and between resource-constrained systems. *International Conference on Model Driven Engineering Languages and Systems*, 349–363.

Fortino, G., Di Fatta, G., Li, W., Ochoa, S. F., Cuzzocrea, A., & Pathan, M. (2014). *Internet and Distributed Computing Systems: 7th International Conference, IDCS 2014, Calabria, Italy, September 22-24, 2014, Proceedings* (Vol. 8729). Springer.

France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. *Future of Software Engineering (FOSE'07)*, 37–54.

Funk, A., Busemann, C., Kuka, C., Boll, S., & Nicklas, D. (2011). Open sensor platforms: The sensor web enablement framework and beyond. *MMS 2011: Mobile Und Ubiquitäre Informationssysteme-Proceedings Zur 6. Konferenz Mobile Und Ubiquitäre Informationssysteme (MMS 2011)*.

Gašević, D., Kaviani, N., & Hatala, M. (2007). On metamodeling in megamodels. *International Conference on Model Driven Engineering Languages and Systems*, 91–105.

Gordon, D., Beigl, M., & Neumann, M. A. (2010). dinam: A wireless sensor network concept and platform for rapid development. *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, 57–60.

Grabis, J., & Kirikova, M. (2011). Perspectives in business informatics research. *10th International Conference, BIR 2011*.

Gray, J., & Karsai, G. (2003). An examination of DSLs for concisely representing model traversals and transformations. *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of The*, 10-pp.

Gronback, R. (n.d.). *Eclipse Modeling Project | The Eclipse Foundation*. Retrieved December 21, 2020, from http://www.eclipse.org/modeling/emf/

Han, G., Jiang, J., Shu, L., Xu, Y., & Wang, F. (2012). Localization algorithms of underwater wireless sensor networks: A survey. *Sensors*, *12*(2), 2026–2061.

Harel, D., & Rumpe, B. (2004). Meaningful modeling: What's the semantics of" semantics"? *Computer*, *37*(10), 64–72.

Hauck, M. (2014). *Automated Experiments for Deriving Performance-relevant Properties of Software Execution Environments* (Vol. 13). KIT Scientific Publishing.

Heidemann, J., Li, Y., Syed, A., Wills, J., & Ye, W. (2005). Underwater sensor networking: Research challenges and potential applications. *Proceedings of the Technical Report ISI-TR-2005-603, USC/Information Sciences Institute*.

Hoffmann, A., Meyr, H., & Leupers, R. (2002). *Architecture exploration for embedded processors with LISA*. Springer.

https://themeforest.net/user/dan_fisher. (n.d.). *OMG | Object Management Group*. Retrieved December 21, 2020, from https://www.omg.org/

Hussey, K., Selic, B., & McClean, T. (2010). *An extended survey of open source model-based engineering tools*. Revision E.

Iniewski, K. (2012). *Optical, Acoustic, Magnetic, and Mechanical Sensor Technologies*. CRC Press.

Jamshidi, M. (2017). *Systems of systems engineering: Principles and applications*. CRC press.

Kaplan, E., & Hegarty, C. (2005). *Understanding GPS: Principles and applications*. Artech house.

Katara, M., & Katz, S. (2007). A concern architecture view for aspect-oriented software design. *Software & Systems Modeling*, *6*(3), 247–265.

Khosla, D., Guillochon, J., & Choe, H. (2017). *Distributed Fusion and Tracking in Multi-Sensor Systems*.

Kieburtz, R. B., McKinney, L., Bell, J. M., Hook, J., Kotov, A., Lewis, J., Oliva, D. P., Sheard, T., Smith, I., & Walton, L. (1996). A software engineering experiment in software component generation. *Proceedings of IEEE 18th International Conference on Software Engineering*, 542–552.

Krahn, H., Rumpe, B., & Völkel, S. (2007). Integrated definition of abstract and concrete syntax for textual languages. *International Conference on Model Driven Engineering Languages and Systems*, 286–300.

Kurtev, I., Bézivin, J., Jouault, F., & Valduriez, P. (2006). Model-based DSL frameworks. *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, 602–616.

Lauterbach, C., Glaser, R., Savio, D., Schnell, M., Weber, W., Kornely, S., & Stöhr, A. (2004). A self-organizing and fault-tolerant wired peer-to-peer sensor network for textile applications. *International Workshop on Engineering Self-Organising Applications*, 256–266.

Lee, J., Kim, J., & Serpedin, E. (2008). Clock offset estimation in wireless sensor networks using bootstrap bias correction. *International Conference on Wireless Algorithms, Systems, and Applications*, 322–329.

Li, X.-Y., Wang, Y., & Wang, Y. (2010). Complexity of data collection, aggregation, and selection for wireless sensor networks. *IEEE Transactions on Computers*, *60*(3), 386–399.

Liggins II, M., Hall, D., & Llinas, J. (2017). *Handbook of multisensor data fusion: Theory and practice*. CRC press.

Liggins, M. E., Chong, C.-Y., Kadar, I., Alford, M. G., Vannicola, V., & Thomopoulos, S. (1997). Distributed fusion architectures and algorithms for target tracking. *Proceedings of the IEEE*, *85*(1), 95–107.

Loicq, J., Clermont, L., Dierckx, W., Van Achteren, T., & Stockman, Y. (2017). A 100-m ground resolution global daily coverage earth observation mission. *International Conference on Space Optics—ICSO 2014*, *10563*, 105632L.

Losilla, F., Vicente-Chicote, C., Álvarez, B., Iborra, A., & Sánchez, P. (2007). Wireless sensor network application development: An architecture-centric mde approach. *European Conference on Software Architecture*, 179–194.

Luo, Z. (2013). *Technological solutions for modern logistics and supply chain management*. IGI Global.

Masri, W., & Mammeri, Z. (2007). Middleware for wireless sensor networks: A comparative analysis. *2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007)*, 349–356.

MetaObject Facility | Object Management Group. (n.d.). Retrieved December 21, 2020, from https://www.omg.org/mof/

Meyer, M., Helfert, M., & O'Brien, C. (2011). An analysis of enterprise architecture maturity frameworks. *International Conference on Business Informatics Research*, 167–177.

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

Mitchell, H. B. (2007). *Multi-sensor data fusion: An introduction*. Springer Science & Business Media.

Mitrou, N., Kontovasilis, K., Rouskas, G., Iliadis, I., & Merakos, L. (2004). *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* (Vol. 3042). Springer Science & Business Media.

Moradi, M., Rezazadeh, J., & Ismail, A. S. (2012). A reverse localization scheme for underwater acoustic sensor networks. *Sensors*, *12*(4), 4352–4380.

Nielsen, J. L., Arrizabalaga, H., Fragoso, N., Hobday, A., Lutcavage, M., & Sibert, J. (2009). *Tagging and tracking of marine animals with electronic devices* (Vol. 9). Springer Science & Business Media.

Noran, O. (2003). An analysis of the Zachman framework for enterprise architecture from the GERAM perspective. *Annual Reviews in Control*, *27*(2), 163–183.

OpenUP - The Best of Two Worlds: Agile, Scrum and RUP. (n.d.). Retrieved December 21, 2020, from http://www.methodsandtools.com/archive/archive.php?id=69p3

Parreiras, F. S. (2012). *Semantic Web and model-driven engineering*. John Wiley & Sons.

Pérez-Medina, J.-L., & Dupuy-Chessa, S. (2007). A survey of model driven engineering tools for user interface design. *International Workshop on Task Models and Diagrams for User Interface Design*, 84–97.

Quartel, D., Engelsman, W., Jonkers, H., & Van Sinderen, M. (2009). A goal-oriented requirement modelling language for enterprise architecture. *2009 IEEE International Enterprise Distributed Object Computing Conference*, 3–13.

Reed, B. L.-K. (2015). *Controller design for underwater vehicle systems with communication constraints* [PhD Thesis]. Massachusetts Institute of Technology.

Römer, K., Kasten, O., & Mattern, F. (2002). Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, *6*(4), 59–61.

Ross, J. (2006). Enterprise Architecture: Driving Business Benefits from IT. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.920666

Rowe, A. G., Bhatia, G., & Rajkumar, R. (2010). *A model-based design approach for wireless sensor-actuator networks*.

Rozanski, N., & Woods, E. (2005). Applying viewpoints and views to software architecture. *Open University White Paper*.

Schmidt, D. C. (2006). Model-driven engineering. *Computer-IEEE Computer Society-*, *39*(2), 25.

Schneider, J.-P., Champeau, J., & Kerjean, D. (2011). *Domain-specific modelling applied to inteegration of smart sensors into an information system*.

Srivastava, N. (2010). Challenges of next-generation wireless sensor networks and its impact on society. *ArXiv Preprint ArXiv:1002.4680*.

Sun, X., Huang, S. C.-H., & Li, M. (2012). Lower bounds on data collection time in sensor networks. *International Conference on Wireless Algorithms, Systems, and Applications*, 120–131.

The Open Group ArchiMate(R) 1.0 Technical Standard. (n.d.). Retrieved December 22, 2020, from https://pubs.opengroup.org/architecture/archimate-doc/ts_archimate/

Touraille, L., Traoré, M. K., & Hill, D. R. (2011). A model-driven software environment for modeling, simulation and analysis of complex systems. *SpringSim (TMS-DEVS)*, 229–237.

van den Brand, M. G. J. (2008). Model-driven engineering meets generic language technology. *International Conference on Software Language Engineering*, 8–15.

Van Der Straeten, R., Mens, T., & Van Baelen, S. (2008). Challenges in model-driven software engineering. *International Conference on Model Driven Engineering Languages and Systems*, 35–47.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, *35*(6), 26–36.

Vossough, E., & Getta, J. R. (2009). Micro implementation of join operation at clustering nodes of heterogenous sensor networks. *International United Information Systems Conference*, 75–90.

Vujović, V., Maksimović, M., & Perišić, B. (n.d.). *A DSM for a Modeling RESTful Sensor Web Network*.

Wang, C. (2008). *Localization and its applications in self-configurable wireless networks*. University of Louisiana at Lafayette.

Yang, J., Zhang, C., Li, X., Huang, Y., Fu, S., & Acevedo, M. (2008). An environmental monitoring system with integrated wired and wireless sensors. *International Conference on Wireless Algorithms, Systems, and Applications*, 224–236.

Zein, O. K., Champeau, J., Kerjean, D., & Auffret, Y. (2009a). Smart sensor metamodel for deep sea observatory. *OCEANS 2009-EUROPE*, 1–6.

Zein, O. K., Champeau, J., Kerjean, D., & Auffret, Y. (2009b). Smart sensor metamodel for deep sea observatory. *OCEANS 2009-EUROPE*, 1–6.

Zheng, M., Sun, J., Liu, Y., Dong, J. S., & Gu, Y. (2011). Towards a model checker for nesc and wireless sensor networks. *International Conference on Formal Engineering Methods*, 372–387.

Zuniga, M., & Dini, G. (2013). *Sensor Systems and Software: 4th International ICST Conference, S-Cube 2013, Lucca, Italy, June 11-12, 2013, Revised Selected Papers* (Vol. 122). Springer.