A Semantic-Based Framework for Processing Complex Events in Multimedia Sensor Networks.

Chinnapong Angsuchotmetee

UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

Doctoral School Ecole Doctorale Sciences exactes et leurs applications (ED211)

University Department Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour (LIUPPA)

Thesis defended by Chinnapong ANGSUCHOTMETEE

Defended on 22nd December, 2017

In order to become Doctor from Université de Pau et des Pays de l'Adour

Academic Field Computer Science

Thesis Title

A Semantic-Based Framework for Processing Complex Events in Multimedia Sensor Networks

Thesis supervised by	Richard CHBEIR	Supervisor
	Yudith CARDINALE	Co-Supervisor
	Shohei YOKOYAMA	Co-Monitor

Committee members

Referees	Maude MANOUVRIER	PSL Research University - Université Paris-Dauphine
	Ahmed MOSTEFAOUI	Université de Franche-Comté
Examiners	Djamal BENSLIMANE	Claude Bernard University of Lyon 1
	Congduc PHAM	Université de Pau et des Pays de l'Adour
Supervisors	Richard CHBEIR	Université de Pau et des Pays de l'Adour
	Yudith CARDINALE	Universidad Simón Bolívar
	Shohei YOKOYAMA	Shizuoka University

Acknowledgements

First of all, I would like to use this chance to give my thanks to everyone who makes it possible for me to come all the way from Thailand to pursue Ph.D. in France. First, I would like to express my gratitude to Franco-Thai Organization of Thailand who helped me to start it all. Without the help of Franco-Thai organization, it would not possible for me to meet my current advisor, Prof. Richard Chbeir. Furthermore, Franco-Thai organization also helps to arrange the co-financing contract with Prince of Songkla University for me which leads me to have a position at the university as a full-time lecturer after graduating. I am looking forward to making some contribution to the organization later in near future when I am going back to Thailand to return all the support that I got. Next, I would like to pass my gratitude and my regard to everyone from Prince of Songkla University. I am looking forward to working with everyone at the university to return all the support I got during all the period of my studies.

Also, I would like to use this chance to thanks all my advisors, co-advisors and every collaborator I work with throughout my study. First, I would like to thank Prof. Richard Chbeir for accepting me in. Thanks a lot for all the advice you give during all these years. Your standard is not an easy one for me to match up to. However, because of the fact that you have never skipped any small details, I really have to step outside my comfort zone and push myself a lot in order to match up to you. In the end, I learn a lot from you. This will not happen if I were to stay only in my comfort zone working as a developer all the time. I look forward to continuing to work with you on some possible future project. Beside my advisor Prof. Richard Chbeir, I would like to give my big thanks to my co-advisor Prof. Yudith Cardinale and Prof. Shohei Yokoyama. You both contribute a lot to this dissertation. Yudith is always someone which I can always rely on when I need an advice on formulating my idea scientifically. Shohei's advice on the algorithm design, implementation, and practical aspect of the contributions are always correct and on point. I would not be able to finish all of these contributions alone without all the advice from you both.

Next, I would like to pass my thanks to everyone from Université de Pau et des Pays de l'Adour, especially all l'ED211 officers who always there when I need some help and always excuse me when I cannot communicate well in French. Also, I would like to thanks, all of my French teachers from Alliance Francais de Bangkok and CLEREMO of IUT de Bayonne who helps me to learn the French language. I would be able to learn all the language myself without all the help from you all.

Last, but not least, I would like to thanks, everyone whose I met during all my four years in Bayonne and Pau. Thanks a lot to everyone in IUT de Bayonne, all the professors, officers and all laboratory colleagues whose make my four years here pleasant. Thanks for always being so supportive of me. All my friends from LIUPPA and every student I met all the past four years, I will not forget any of you and make sure to remember all the time we spent together. Finally, my special thanks go to everyone from my fencing team, Anglet Olympique Escrime. You all help me to finish this dissertation while keeping myself remain fit both physically and mentally.

Thanks a lot everyone, I promise that this will not be the last time that we meet. I look forward to come back to see everyone and continue to work with you in near future.

A SEMANTIC-BASED FRAMEWORK FOR PROCESSING COMPLEX EVENTS IN MULTIMEDIA SENSOR NET-WORKS

Abstract

The dramatic advancement of low-cost hardware technology, wireless communications, and digital electronics have fostered the development of multifunctional (wireless) Multimedia Sensor Networks (MSNs). Those latter are composed of interconnected devices able to ubiquitously sense multimedia content (video, image, audio, etc.) from the environment. Thanks to their interesting features, MSNs have gained increasing attention in recent years from both academic and industrial sectors and have been adopted in wide range of application domains (such as smart home, smart office, smart city, to mention a few). One of the advantages of adopting MSNs is the fact that data gathered from related sensors contains rich semantic information (in comparison with using solely scalar sensors) which allows to detect complex events and copes better with application domain requirements. However, modeling and detecting events in MSNs remain a difficult task to carry out because translating all gathered multimedia data into events is not straightforward and challenging.

In this thesis, a full-fledged framework for processing complex events in MSNs is proposed to avoid hard-coded algorithms. The framework is called Complex Event Modeling and Detection (CEMiD) framework. Core components of the framework are:

- MSSN-Onto: a newly proposed ontology for modeling MSNs,
- CEMiD-Language: an original language for modeling multimedia sensor networks and events to be detected, and
- GST-CEMiD: a semantic pipelining-based complex event processing engine.

CEMiD framework helps users model their own sensor network infrastructure and events to be detected through CEMiD language. The detection engine of the framework takes all the model provided by users to initiate an event detection pipeline for extracting multimedia data feature, translating semantic information, and interpret into events automatically. Our framework is validated by means of prototyping and simulations. The results show that our framework can properly detect complex multimedia events in a high work-load scenario (with average detection latency for less than one second).

Keywords: multimedia sensor networks, complex event processing, semantic web, ontology

Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour (LIUPPA)

Résumé

Les progrès de la technologie des capteurs, des communications sans fil et de l'Internet des Objets ont favorisé le développement des réseaux de capteurs multimédias. Ces derniers sont composés de capteurs interconnectés capables de fournir de façon omniprésente un suivi fin d'un espace connecté. Grâce à leurs propriétés, les réseaux de capteurs multimédias ont suscité un intérêt croissant ces dernières années des secteurs académiques et industriels et ont été adoptés dans de nombreux domaines d'application (tels que la maison intelligente, le bureau intelligent, ou la ville intelligente). L'un des avantages de l'adoption des réseaux de capteurs multimédias est le fait que les données collectées (vidéos, audios, images, etc.) à partir de capteurs connexes contiennent des informations sémantiques riches (en comparaison avec des capteurs uniquement scalaires) qui permettent de détecter des événements complexes et de mieux gérer les exigences du domaine d'application. Toutefois, la modélisation et la détection des événements dans les reséaux de capteurs multimédias restent une tâche difficile à réaliser, car la traduction de toutes les données multimédias collectées en événements n'est pas simple.
Dans cette thèse, un framework complet pour le traitement des événements complexes dans les réseaux de capteurs multimédias est proposé pour éviter les algorithmes codés en dur et pour permettre une meilleure adaptation aux évolution des besoins d'un domaine d'application. Le Framework est appelé CEMiD et composé de :

- MSSN-Onto : une ontologie nouvellement proposée pour la modélisation des réseaux de capteurs,
- CEMiD-Language : un langage original pour la modélisation des réseaux de capteurs multimédias et des événements à détecter, et
- GST-CEMiD : un moteur de traitement d'événement complexe basé sur un pipeline sémantique.

Le framework CEMiD aide les utilisateurs à modéliser leur propre infrastructure de réseau de capteurs et les événements à détecter via le langage CEMiD. Le moteur de détection du framework prend en entrée le modèle fourni par les utilisateurs pour initier un pipeline de détection d'événements afin d'extraire des données multimédias correspondantes, de traduire des informations sémantiques et de les traduire automatiquement en événements. Notre framework est validé par des prototypes et des simulations. Les résultats montrent que notre framework peut détecter correctement les événements multimédias complexes dans un scénario de charge de travail élevée (avec une latence de détection moyenne inférieure à une seconde).

Mots clés : réseaux de capteurs multimédias, traitement des événements complexes, web sémantique, ontologie

Table of Contents

List of Tables

List of Figures

Introduction

Background

The vertiginous advances in low-cost hardware technology, wireless communications, and digital elec-tronics have fostered the development of multifunctional (wireless) *Multimedia Sensor Networks (MSNs)*. Multimedia sensor networks are networks of interconnected devices that are able to ubiquitously retrieve multimedia content, such as video and audio streams, images, and still scalar sensor data from the envi-ronment [1, 2, 3]. Nowadays, multimedia sensor networks become increasingly popular and important in our everyday life, for monitoring, tracking, and detecting events in different scenarios (e.g., smart homes, smart buildings, smart cities) [4].

The popularity of sensor networks (also multimedia sensor networks) has led to the tremendous amount of data that are produced from sensors each day. This leads to the challenge of how the data from sensor networks should be maintained efficiently such that events can be detected among the gathered data effectively. The existence of different and smart devices within the network makes this challenge more difficult to overcome due to the diversity of sensor types, a large variety of sensor output formats, and the difficulty in modeling multimedia data such that they can always be retrieved efficiently when detecting an event. Yet, approaches for modeling sensor networks, modeling the gathered data, and assisting users for modeling and detecting events are lagging behind.

Adopting multimedia sensor networks or hybrid sensor networks(i.e., a sensor network which consists of both scalar and multimedia sensors) over using solely scalar sensors allows an application to be able to detect more complex events that may not be possible to be detected without using multimedia data. For example, a network of velocity sensors by the roadside allows only the detection of an `Excessive Driving Speed` event. However, enriching the network with surveillance cameras provides the possi-bility of detecting more complex events, such as `Traffic Jam` event or `Car Accident` event. The analysis of surveillance camera footage and velocity sensors can also be used for detecting the plate number of the car that drives faster than a speed limit.

Even though, multimedia sensor networks are extensively used nowadays, problems concerning the difficulty in processing events still persists. The main reasons can be briefly elaborated as follows.

- The lack of a data model for modeling data gathered from multimedia sensor networks, which can help to facilitate event modeling and detection;
- The lack of an approach for modeling and detecting events in multimedia sensor networks;
- The lack of an interface, a tool, or a language to support users in the modeling of complex events that they are interested in.

In an ideal use-case of a multimedia sensor network application, users are needed to be able to model sensor network infrastructure and events that they would like to detect on their own without relying on hard-coding or reimplementation by developers. However, the lack of a suitable data model, a language

for modeling events, and an approach for processing complex events in multimedia sensor networks forces developers to hard-code all the data models and the detection process. To overcome this problem, a suitable data model for modeling multimedia sensor networks, a language which helps users to model their own needs, and an approach for processing events that are designated to multimedia sensor networks are needed to be proposed.

There exist several studies related to processing events in multimedia sensor networks that have focused separately on sensor network modeling [5, 6], multimedia data modeling [7, 8], and complex event processing languages [9, 10]. These studies can address only challenges and requirements which are related to their respective field. However, to process events in multimedia sensor networks effectively, all the requirements for modeling sensor network, modeling multimedia data, and processing complex events, are needed to be addressed comprehensively. Hence, the fact that existing studies focus solely on overcoming challenges and requirements in their respective field make them not to be able to address to fully overcome challenges and requirements on processing events in multimedia sensor networks.

To the best of our knowledge, there is no existing study that can comprehensively address the aforementioned challenges on processing events in multimedia sensor networks. Hence, to overcome the limitations of existing approaches, in this work, we propose *CEMiD*, a full fledge ontological-based framework to support Complex Events ModelIng and Detection in multimedia sensor networks, as well as the modeling of the multimedia sensor networks itself. *CEMiD* mainly relies on three core components: (i) an ontology-based data model, named *MSSN-Onto*, which allows the representation and modeling of the knowledge related to multimedia sensor networks modeling and semantic interoperability; (ii) a *CEMiD* language for defining and describing complex events in multimedia sensor networks; and (iii) a complex event processing engine, which detects events according to predefined data models provided by the *MSSN-Onto* and *CEMiD* language. *CEMiD* framework helps users design their MSN applications, process all gathered sensor readings, and translate them into events, when detected according to predefined models provided by users through the *CEMiD* language.

The ultimate aim of this dissertation is to provide an approach for processing complex events in multimedia sensor networks. Our approach targets multimedia sensor networks modeling and complex event processing through the following objectives:

- To propose a data model for modeling multimedia sensor networks and data gathered from the networks which helps to facilitate complex event modeling and event detection;
- To propose an approach for modeling and detecting events in multimedia sensor networks in such a way that users can freely model events that they would like to detect without relying on developer's hard-coding or re-implementation;
- To propose a full-fledged framework for detecting events which is capable to support different needs of users in different application domains such that, developers can easily reuse the framework in different wide range of applications while keeping the need of reimplementation to be minimal.

Contributions

Responding the proposed objectives, the main contributions of this study can be summarized as follows:

- *Multimedia Semantic Sensor Network Ontology(MSSN-Onto)*, an ontology for modeling multimedia sensor networks. This first contribution is the provision of a generic and comprehensive ontology, *MSSN-Onto*, that aims at addressing the challenge concerning the lack of a suitable data model for modeling multimedia sensor networks. It helps to model multimedia sensor networks infrastructure and all the gathered data in such a way that all the modeled data can be plugged to several application domains;

- *CEMiD* language, a language for modeling complex events in a multimedia sensor network. This second contribution is related to addressing the challenge of allowing users to model their own events without relying on developer's hard-coding or reimplementation;
- *CEMiD* Framework, an engine to for processing complex events in multimedia sensor networks. The third contribution is a full-fledged framework which is built on top of the *MSSN-Onto* and the *CEMiD* language. The framework is able to process and detect complex events in multimedia sensor networks in a near real-time manner. The basis of the framework is constructed by a newly proposed pipelined-based multimedia event processing plugins so called the *GST-CEMiD*.

Publications

From the results of the contributions of this study, the following publications were produced and support the material presented in this thesis:

- Chinnapong Angsuchotmetee and Richard Chbeir. Nov 2016. A survey on complex event definition languages in multimedia sensor networks. In Proceedings of the 8th International Conference on Management of Digital EcoSystems (MEDES). ACM, New York, NY, USA, 99-108.
- Chinnapong Angsuchotmetee, Richard Chbeir, Yudith Cardinale and Shohei Yokoyama. Dec 2017. A dynamic event detection framework for multimedia sensor networks. In 23rd Asia-Pacific Conference on Communications (APCC) (*Accepted on August 31, 2017, To be appeared), Perth, Australia,
- Chinnapong Angsuchotmetee, Richard Chbeir, Yudith Cardinale and Shohei Yokoyama. 2017, A Pipelining-based Framework for Processing Events in Multimedia Sensor Networks, ACM SAC 2018, (*Accepted on November 24, 2017, To be appeared)
- Chinnapong Angsuchotmetee, Richard Chbeir, Yudith Cardinale and Shohei Yokoyama. 2017, CEMID: A Semantic Based Framework For Complex Events Modeling and Detection in Multimedia Sensor Networks. submitted to ICDE 2018. (* The notification result is expected to be out by December 22, 2017)
- Chinnapong Angsuchotmetee, Richard Chbeir, and Yudith Cardinale, 2017, A Pipelining-based Framework for Processing Events in Multimedia Sensor Networks, MSSN-Onto: An Ontology-based Approach for Flexible Event Processing in Multimedia Sensor Networks. (* Minor revision is submitted to Elsevier Future Generation Computer System Journal on October 13, 2017)

Structure of the Dissertation

The structure of the dissertation is given as follows:

- Chapter 1 introduces the concepts of multimedia sensor networks and event processing in multime-dia sensor networks. Problems, challenges, and requirements on these areas are identified and given in this section;
- Chapter 2 is dedicated to the literature review. Most relevant and recent studies related to processing events in multimedia sensor networks are discussed, evaluated, and compared with respect to the identified requirements for a generic approach for complex event processing in multimedia sensor networks;

- Chapter 3 introduces the overview architecture of the *CEMiD* framework. All the main components of the framework are described briefly in this chapter. Each of the main components is further described in detail later as a dedicated chapter;
- Chapter 4 describes the *MSSN-Onto* which is one of the main components of the *CEMiD* framework. The *MSSN-Onto* is the core data model of the framework. All the sensor infrastructure information, sensor readings, multimedia data, and events are modeled by using the *MSSN-Onto*;
- Chapter 5 describes the *CEMiD* language. The *CEMiD* language is one of the main components of the *CEMiD* framework and also one of the main contributions of this dissertation. The *CEMiD* language helps users to model all the data models which are necessary for processing events in multimedia sensor networks without having to rely on hard-coding or using a low-level data modeling language which can be difficult for users to use;
- Chapter 6 is dedicated to the details information on how the *CEMiD* framework can be developed. All the algorithms that are used and the implementation details are described in this chapter. This chapter ends with the experiments for validating the performance of the framework;
- Chapter 7 summarizes and concludes all the contributions of this dissertation. This chapter ends with the list of future studies that can be done in the future.

Chapter 1

Multimedia Sensor Network and Complex Event Processing: Preliminaries

This chapter presents a general description of multimedia sensor networks and complex event processing technologies. The most important open research questions from both of these fields are identified and described. At the end of this chapter, we discuss the common points and the needs of applying complex event processing in multimedia sensor networks. For the clarity of the discussion and illustration, we base the discussion in the context of a *Smart Office*, as motivating scenario.

1.1 Multimedia Sensor Networks

Multimedia Sensor Networks (MSNs) emerge from the improvement of sensor capabilities and the intro-duction of several low-cost, high-performance embedded devices, such as BeagleBone[1] or Raspberry Pi[2]. The performance of these devices is powerful enough to encode and stream video or audio data, while their size is considerably small. The overall differences between scalar sensors and multimedia sensors are mainly related to the aspects of device size, power consumption, data type produced, and communication methods. Table 1.1 summarizes the major difference between scalar sensors and multimedia sensors.

Table 1.1 – Comparison between Scalar Sensors and Multimedia Sensors

	Scalar Sensors	Multimedia Sensors
Device Size	Small	Medium
Power Consumption	Low	Medium
Data Type Produced	Numeric, Text	Video, Audio, Image, Numeric, Text
Communication Methods	Infrared, Bluetooth, Short range wireless communication (e.g., Zigbee, RFID)	Ethernet, 802.1X, Cellular (e.g., UMTS, LTE)
Sample Devices	NodeMCU (ESP8266), Blynk Arduino based MCU	Beaglebone, Raspberry Pi, ARMv7, ARMv9

The comparision summarizes the information according to [11, 12, 13]. Regarding the device size, a scalar sensor device is considerably smaller than a multimedia sensor. The difference in device reflects the difference in power consumption rate. Scalar sensors can operate on battery for a considerably long period

[1] http://beagleboard.org/bone

[2] http://www.embeddedpi.com/

(e.g., a month of operation time in a single battery charged). In addition, a multimedia sensor usually contains more complicated circuit can produce complex data types. However, its power consumption rate is considerably higher than a scalar sensor. Hence, a multimedia sensor is rarely operated solely on battery. It is often required to be equipped with a reliable power source all the time. The methods for transferring data between scalar sensors and multimedia sensors are also mostly different. The scalar sensors use a low powered short range communication method, such as infrared, Bluetooth, RFID[3] or Zigbee[4]. Multimedia sensor devices mostly use an Internet Protocol (IP) based communication interface such as Ethernet, Wireless 802.1X, or cellular based communication, such as UMTS[5] or LTE[6].

Nowadays, both scalar sensors and multimedia sensors are available commercially with an inexpensive price. Multiple sensors are also often integrated into a single device, so-called a *smart device*, which can sense and produce multiple types of data. Examples of smart devices include smartphones and smart televisions. Applications which rely on a smart device includes smart home surveillance, smart offices, telemedicine, and smart cities.

1.2 Complex Event Processing

In general, an event can be either an *Atomic Event*, or a *Complex Event*. Definitions of both types of events can be found in the *Event Processing Glossary* report, provided by Real-Time Intelligence and Complex Event Processing community [14]. They are given follows.

- Atomic Event: An atomic event (or a simple event) is an event that is not viewed as summarizing, representing, or denoting a set of other events;
- Complex Event: A complex event is an event that summarizes, represents, or denotes a set of other events.

According to the definitions provided by the glossary, it can be said that a complex event is an abstraction of multiple events, while an atomic event is a smallest unit of a complex event. For example, a *Rainstorm* event can be modeled as a complex event which consists of a *Heavy rain* event, and a *Fast wind speed* event. The *Heavy rain* and *Fast wind speed* events are not an abstraction of other events. Hence, they both are atomic events. With respect to the sensor network context, we can intepret the definition of an atomic event as *an event that can be detected by using a single reading from a sensor*, while a complex event definition can be interpret as *an event that requires multiple sensor readings from one or more sensors to detected*

Complex Event Processing (CEP) is a field of study which focuses on analyzing a pattern of data from multiple sources and detecting meaningful events from gathered data [15]. Figure 1.1 illustrates an engine for processing complex events.

In short, a complex event processing engine assumes that inputs of the engine are streams of atomic events. The engine works by applying predefined event patterns (given as a parameter or defined previously) by users over streams of atomic events. The result of the complex event processing engine is the streams of (complex) events that match the predefined event patterns.

A complex event pattern of the complex event processing engine is constructed by modeling a complex event as a spatiotemporal pattern of multiple atomic event [15]. For example, *one hour meeting occurs at*

[3] Radio-frequency identification (RFID) is the use of a small device tag which can emit radio wave for sending data.

[4] Zigbee is a commercial name of an IEEE 802.15.4-based communication method which helps low-powered sensor node to be able to use Internet Protocol (IP) based network

[5] Universal Mobile Telecommunications System (UMTS) is a third generation mobile cellular system which allows mobile devices to access data over IP.

[6] Long-Term Evolution (LTE) is a fourth generation mobile cellular system which allows mobile devices to access data over IP with higher speed than UMTS

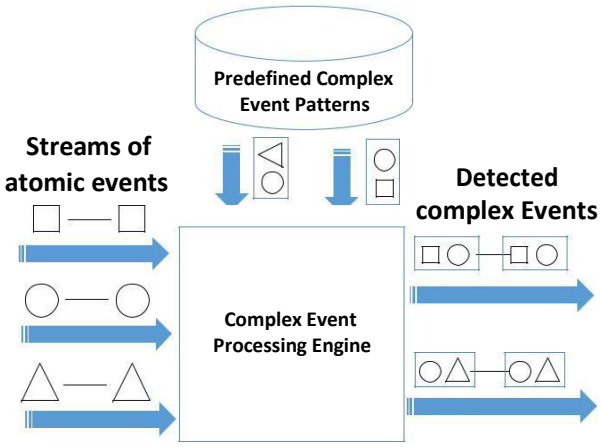


Figure 1.1 – Complex Event Processing (CEP) Engine

Room A, can be modeled as a pattern of: an atomic event *people are in Room A* is detected simultaneously with multiple atomic events *people are speaking*, and they occur continuously for one hour within *Room A*. Expressing such a pattern can be difficult for end users. Hence, a common practice for most of the complex event processing engines is to propose a high-level language for users to define their own desirable event patterns [16, 17, 18].

Since the last decade, complex event processing has received an extensive attention from both academic and industrial communities. Lots of complex event processing engines and languages have been proposed [15, 19]. Complex event processing is mainly adopted in business monitoring and analysis field. However, the usage of a complex event processing engine in a sensor network context, especially in multimedia sensor networks, has not been extensively addressed.

1.3 Multimedia Sensor Networks and Complex Event Processing: Motivating Scenario

So far, we discussed the background of multimedia sensor networks and complex event processing. In this section, we depict the need of applying complex event processing in a smart office equipped with a multimedia sensor network.. The smart office is designed to monitor, support, and facilitate activities of users within the office. To do so, different types of sensors are installed within the office to gather data. Figure 1.2 shows a sample smart office with four types of sensors installed: (i) video camera, (ii) computer (for sensing a list of processes that are executed on each computer), (iii) light sensor, and (iv) temperature sensor.

Detecting solely atomic events may be sufficient in several use-cases. However, the need to detect complex events is a must in various situations, especially when having a gap between installed sensor network and events to be detected (quality of sensings, missing data, etc.) and when having an evolving set of needs (since needs can evolve with time and can be expressed by different people involved in an application domain).

Table 1.2 shows some complex events to be detected in order to cope with the needs of an *employee* and a *manager* of that office. Events that are related to employees are mostly related to the condition of their working room whether the light is too bright or too dark for their activities, or whether the room

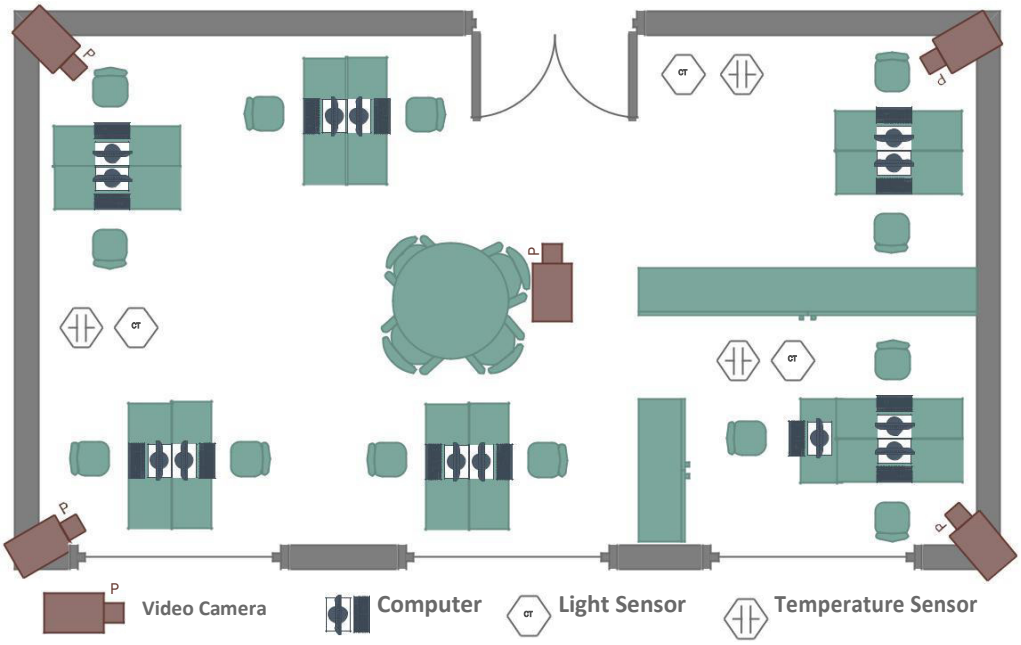


Figure 1.2 – Smart Office with Multimedia Sensors Installed

temperature is too hot to work comfortably. Managers are more concerned about surveying the office to detect whether the situation is currently normal or an unexpected incident is occurring (e.g., someone tries to hack an office system). As shown in Table 1.2, each event requires the combination of several atomic readings with some video detection techniques (to detect if people are authorized stuff or not). This clearly shows a simple multimedia sensor network scenario where data gathered from multimedia and scalar sensors are needed to be combined so to detect an event.

In order to be able to cope with complex event modeling and detection, challenges in both hardware aspect and event processing aspect are needed to be addressed properly. The challenges that are related to hardware aspect are given as follows:

- Communication Method Optimization: Sensor devices are mostly working on a wireless commu-nication medium. Hence, it is susceptible to the delay, loss, and error when coping with multimedia data [11]. This challenge is related to how to optimize the communication method to reduce the communication latency, loss, and errors;

- Power Consumption Optimization: Encoding and decoding multimedia data consume a consid-erable amount of processing power. This affects directly the power consumption rate [11]. This challenge is related to a suitable method for multimedia sensors to encode and decode multimedia data under an energy constraint environment.

In short, it is needed to be ensured that multimedia sensors within the network can effectively transfer all the gathered data effectively, while communication error and power consumption rate are kept minimal. Overcoming these challenges means that event processing engine will always have a credible data (i.e., data which contains minimal latency and error) for processing events.

Table 1.2 – Complex events and their atomic events in smart energy management and smart office surveillance applications

Complex Event Name	Atomic Event(s) Used				Constraint
	Video Camera	Computer	Light Sensor	Temperature	
- User Role: Employee					
Excessive Working Light	A known person is detected	Working	Too bright	-	-
Excessive Videoconferencing Light	A known person is detected	Videoconferencing	Too bright		-
Overpowered Heater	A known person is detected	-	-	Too high	During office hour
- User Role: Manager					
Hacking	An unknown face is detected	Working	-	-	Outside working hour
Normal Situation	More than one person are detected	Working	-	-	During office hour

Regarding the event processing aspect, there two main components that are required for developing an effective complex event processing engine for multimedia sensor networks. First, a language is needed to allow users to define events to be detected. Such a language needs to provide features for:(i) modeling multimedia sensor networks infrastructure (e.g., defining a list of sensors, capabilities of each sensor, their installation locations), (ii) modeling atomic events, and (iii) modeling complex events. Second, a processing engine which helps users handle the raw input stream, decode the data, and translating them into an event is also required. Such an engine is needed to be able to detect events in at least a near real-time manner as some events can lose their significance when detected a bit late. We summarize explicitly all the requirements regarding the event processing aspect as follows.

- MSNs Modeling: In order to provide a satisfactory event definition, multimedia sensor networks modeling needs to be considered. It needs to be generic (can be applied to various applications) and able to include: (i) the list of related sensors, (ii) their locations (e.g., installation location, coverage area), and (iii) their capacities (e.g., hardware specification, output format, accuracy);

- Multimedia Sensor Data Modeling: An atomic event is always related to a sensor reading. Hence, the ease and the effectiveness of defining an atomic event rely on how sensor data is indexed. The existence of multimedia sensor readings in multimedia sensor networks needs to rely on appropriate and rich multimedia data design and index, in order to cope with atomic and complex events definition process later on;

- Event Modeling: Users need to model both *atomic* and *complex* events on their own. Hence, a high-expressive language with a set of dedicated operators for constructing a complex event pattern is needed in order to support at least both event spatial and temporal facets.

- Near Real-time Event Detection: Events that are time-sensitive (i.e., an event that loses its significance if it is detected too late) must be detected and reported in a near real-time manner, such as `Hacking` or `Overpowered Heater`. Thus, the system needs to react to the occurrence of these events on time;

- Event Detection from Historical Data: Some events are not required to be detected in a near real-time manner. In this case, the system needs to store and report them later in response to the historical data query from users.

- Application Domain Interoperability: All the data model, event model, and event detection method are needed to be able to accommodate the change in an application domain. Ideally, users must be able to define, modify, extend, or change an application domain on their own.

In this dissertation, we choose to focus on the challenges on the event processing aspect. This means that the main objective of this dissertation is to propose a complex event processing engine which is designated to multimedia sensor networks. The challenges regarding sensor communication method optimization and power consumption optimization are out of the scope of this dissertation.

To the best of our knowledge, none of the existing studies can fully address the requirements in the event processing aspects as given in this section. A new approach that can handle them is a must. Next chapter surveys related studies before introducing our proposal.

Chapter 2

Related Studies on Complex Event Processing in Multimedia Sensor Networks

Existing studies that are related to complex event processing in multimedia sensor networks can be categorized into three main topics: (i) sensor network modeling; (ii) multimedia data modeling and retrieval; and (iii) complex event processing engine. We select significant studies in these three main topics to discuss in this chapter. This chapter begins with a section dedicated to related studies on sensor network modeling. Another section follows to it describe related studies on multimedia data modeling and retrieval. The section after is dedicated to describing related studies on complex event processing engine. This chapter ends with a comparative analysis of every selected study against requirements as identified in the previous chapter.

2.1 Sensor Network Modeling

The approaches for modeling sensor networks can be categorized into three groups: (i) relation database approach, (ii) graph database approach, and (iii) ontology-based approach. They are described as follows.

2.1.1 Relational Database Approach

The studies belonging to this approach choose to model sensor networks using relational databases. Sensor network infrastructure and data produced from the network are modeled and stored as relation tuples in a relational database with a predefined schema. Significant studies which adopt this approach include [20, 21, 22]. Their details are described as follows.

2.1.1.1 Fjording the stream: an architecture for queries over streaming sensor data [20]

This study was proposed in 2002. It is one of the earliest approaches that aimed at proposing a middleware for modeling a sensor network using a relational database. This study proposes a framework named *Fjording* designated to use as a middleware for modeling a sensor network infrastructure, and helps developers to gather data collected from a sensor network. The framework relies on a lightweight relational

database system, so-called *TinyDB*, as its backend. The benefit of adopting TinyDB lies in the fact that it consumes low computation resources. Hence, this allows the Fjording framework to be deployed in a resource-constrained environment, while it can still provide users or developers a capability to query sensor data using a conventional SQL expression.

The main limitation of the Fjording framework is the hard-coding schema nature of the framework. This restricts the framework to hardly interoperate with another system which does not use the Fjording framework. The design of the framework also does not take a multimedia sensor network into account. Hence, its capability for supporting multimedia sensor networks is also limited.

2.1.1.2 IrisNet: an architecture for a worldwide sensor web [21]

This study was proposed in 2003. The IrisNET is a framework which allows developers to connect multiple sensors to form a sensor network over the Internet. Hence, this makes possible for IrisNET developers to create a worldwide level sensor network. The backend of modeling sensor network infrastructure and the gathered data is a relational database. Each application which adopts the IrisNET needs to propose a database schema on their own for modeling sensor network infrastructure and gathered data.

The IrisNet is one of the earliest middlewares of a sensor network application which is capable to support a worldwide level architecture. The limitation of the IrisNet is related to fact that each of an application connected to the IrisNet needs to propose a schema for modeling a sensor network on its own. Hence, each IrisNET application may not be able to interoperate with each other if they use different database schemas.

2.1.1.3 A distributed cloud-based cyberinfrastructure framework for integrated bridge monitoring [22]

This study has been recently proposed in 2017. The aim of this study is to propose a cloud-based infrastructure framework for monitoring sensors which are installed for monitoring bridges in a road surveillance system. The system relies on a distributed database system so-called *Cassandra*[1] of Apache Foundation. The data model of this study is hard-coded into five relational tables which are Sensor, SensorData, ImageData, Geometry, and FELine. Sensor and SensorData tables are used for modeling sensors and produced data. ImageData table is used for storing images recorded from sensors. Geometry and FELine are tables which contains data specific to bridge monitoring application.

The significant point of this study is that it is one of the most recent studies which adopts distributed relational database to propose a framework for modeling multimedia sensor network (in this case, a bridge monitoring sensor network). However, this study focuses solely on bridge monitoring application. Hence, it cannot be used for modeling different types of sensors, and events from gathered data. It also cannot be interoperated with other application as the database schema of this study is fixed to a bridge monitoring application.

2.1.2 Graph Database Approach

Graph database refers to a database which chooses to model and store data as a graph. This means that all data within a graph database is modeled as a set of interconnected nodes in which a node in a graph represents a data, while an edge represents a relation between a pair of nodes. Significant studies which adopt graph database approach for modeling data in sensor networks include [23, 24, 25]. Their details are given as follows.

[1] http://cassandra.apache.org/

2.1.2.1 The G* graph database: efficiently managing large distributed dynamic graphs [23]

This study was proposed in 2014. It proposes an architecture of a graph database system for managing large distributed data collected from sensor networks and social networks. The system is named $G*$. G* models data as a nested graph in which a graph is modeled as a collection of vertices and edges. Each vertex can be either a data node or a sub-graph. The main contributions of $G*$ are mainly on the data model, system architecture for indexing and handling large distributed graphs, and a language for querying data from stored graphs.

The main advantage of $G*$ lies in the scalability of the system on managing series of large graphs and the performance on querying for data. $G*$ also offers a language for users to query for data from the graph. However, the design of $G*$ is neither designated for multimedia data modeling nor sensor data modeling. Its design is also not designated for complex event modeling and retrieving. This limits the capabilities of $G*$ on supporting complex event processing for multimedia sensor networks.

2.1.2.2 Integration of Geo-Sensor Feeds and Event Consumer Services for Real-Time Representa-tion of Iot Nodes [24]

This study was proposed in 2016. The main objective of this study is to propose an architecture for integrating a large amount of data provided from Geo-Sensor[2] feeds. This study chooses to adopt a well-known graph database implementation so-called *Neo4J*[3] for modeling all the data. Th main contribution of this study is a web-based RESTFul service architecture for retrieving a geo-sensor feeds (modeled according to the GeoJSON[4] standard), re-modeled as a graph and stored into *Neo4J* repository.

The advantage of this study lies in the usage of *Neo4J* technology. It allows the study to adopt the Neo4J proprietary language to query for data directly without the need of proposing any new language. The limitation of the study lies in the fact that it is limited to modeling data coded according to GeoJSON standard only. Hence, it cannot be used in the case which sensors within the network are heterogeneous.

2.1.2.3 Big Data Model Simulation on a Graph Database for Surveillance in Wireless Multimedia Sensor Networks [25]

This study was proposed in 2017. This study proposes a data model for modeling multimedia gathered from wireless multimedia sensor networks (WMSNs). The This study proposes three levels data model for modeling multimedia sensor networks. The first level of the data model is used for modeling a linked graph between sensors and the data that they produced. The second level is used for modeling a gateway (i.e., a base station for collecting sensor data and interpreting semantic information) as a linked graph between a gateway and sensors. The final level is used for modeling a graph of interconnected gateways. The data model proposed in this study was tested by means of experiments. Several graph databases implementations and relation database implementations are adopted to compare the performance of the data model under the different implementations. Tested graph database backends used are OrientDB[5] and Neo4J. Tested relational database backend used is MySQL.

The main advantage of this study lies in the fact that the data model is modeled designated for handling data within wireless multimedia sensor networks. However, this study has not yet focused on modeling complex events in multimedia sensor networks. Hence, querying for data or events currently rely solely on the language provided by the backend (e.g, Neo4J language for Neo4J case, or SQL language for MySQL case).

---

[2] Geo-Sensor refers to a sensor which provides spatial information (e.g., GPS co-ordinate, a geometry of a covered spatial area) as an observation result

[3] https://neo4j.com/

[4] http://geojson.org/

[5] https://www.orientdb.com

2.1.3 Ontology-based Approach

The limitations of the relational database approaches for modeling a sensor network is related to mostly its fix database schema nature. Hence, it cannot be used for combining data from heterogeneous sources (i.e., integrating data from sensor networks with different infrastructures). To overcome this limitation, several ontologies for modeling sensor networks have been proposed and used in several domains such as smart cities [26], smart buildings [27], microgrids [28]. Some ontologies in previous studies are also proposed to be used as an abstraction layer that can model any sensor network regardless of the technologies used [1, 6, 29, 30]. Several studies propose generic ontologies for sensor networks such as OntoSensor [6] and SensorML [1], or domain-specific ontologies such as Marine Meta Data Interoperability (MMI) for an oceanic sensor network [30].

In 2011, the World Wide Web Consortium (W3C), introduced the *Semantic Sensor Network (SSN)* ontology [29] as an attempt to propose a generic ontology for a sensor network. All existing ontologies for modeling sensor networks have been reviewed and analyzed toward proposing the *SSN* ontology. Hence, we choose to discuss only the *SSN* ontology in detail in this sub-section as all the features for modeling sensor networks of other major ontologies are already reviewed, analyzed, and integrated into the *SSN* ontology. The detail of the *SSN* ontology is given as follows.

2.1.3.1 Semantic Sensor Network (SSN) Ontology [29] Main

concepts of the *SSN* ontology are depicted in Figure 2.1.

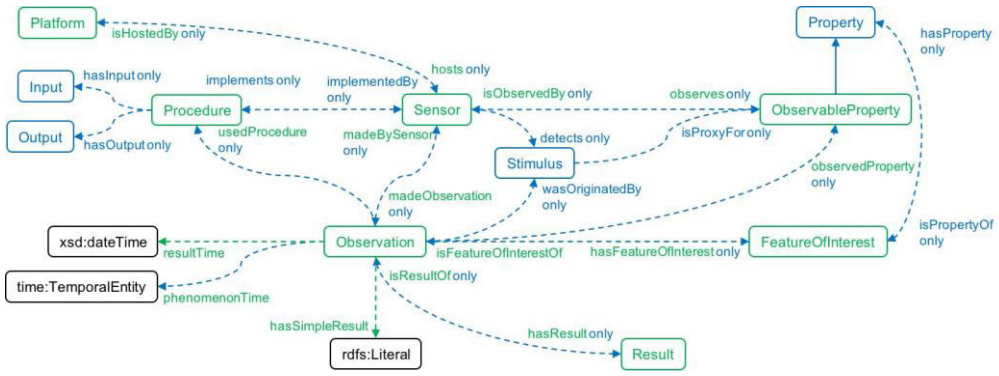


Figure 2.1 – Semantic Sensor Network *SSN* Ontology

Briefly, the *SSN* ontology models a sensor as an entity which produces an observation according to a feature of interest. For example, a temperature sensor can be modeled as a sensor which has `Temperature` as its `FeatureOfInterest`, and has a numerical value as its `Result`. It is noted that not every concept of the *SSN* ontology is given in Figure 2.1 for the brevity of the discussion. The latest version of the ontology is published on 7th September 2017. The detail of the ontology can be found online[6]. More details about *SSN* ontology and its usage will be provided the later chapter as the *SSN* ontology is a part of our approach.

The *SSN* ontology has been adopted by researchers and industrialists in several major projects such as SENSEI [31] and SmartProducts[7]. The *SSN* ontology is generic and complete when it comes to a scalar sensor network. There are several domain-specific ontologies that are developed by extending the SSN

[6] http://www.w3.org/TR/vocab-ssn/

[7] http://www.smartproducts-project.eu

ontology to take the advantage of its completeness for modeling sensor networks such as [32] and [33]. However, they do not consider any multimedia sensor aspect and thus remain limited in that context.

Briefly, the *SOSA/SSN* ontology, the observation perspective, describes a sensor network through seven main modules which cover different aspects of a sensor network. Sensor descriptions and capabilities are defined using the following modules:

- `Observation/Actuation/Sampling` module: contains the fundamental concepts for model-ing a sensor network, which are `Sensor`, `Stimulus`, `Observation`, `ObservableProperty`, and `FeatureofInterest`. In short, a sensor is modeled using the `Sensor` concept. A `Sensor` observes a `FeatureOfInterest` from a `Stimulus` and produces a result as an `Observation`. A possible kind of value or feature that a sensor can observe is modeled us-ing `ObservableProperty`; This module also contains concepts that are related to *Actuator* and *Sampling* perspectives. However, they are given in Figure 4.1 as they are not related to the *Observation* perspective;
- `Feature` module: helps to model all properties and features that are used within a sensor network;
- `System` module: is used for modeling a sensor network as a system which is composed of multiple sensors. A system can also be divided into multiple sub-systems by using an object property `hasSubSystem`. A system or a sub-system itself can also be a sensor;
- `Procedure` module: helps to model a sensing method of a sensor. Such a method has an input information described by the `Input` concept, and has an output information describes by the `Output` concept.

The operating restriction of a sensor network is described by:

- `SystemProperty` module: is used for modeling properties and operating constraints of each sensor within a sensor network and also the network itself. In short, the capabilities of a sensor (e.g., accuracy, precision) are modeled by using the `SystemProperty` concept. Survival constraints and operating constraints are modeled by using `SurvivalRange` and `OperatingRange` concepts;
- `Condition` module: is used for modeling a sensing constraint of a sensor as a constraint block. For example, a wind speed sensor can be attached to a constraint block to indicate that it can only capture the data when the wind speed is between 10-60m/s.

The deployment related information is described by the following modules:

- `Deployment` module: in the case where the sensor is a device, it is expected to be deployed at some specific location. This module is used for describing such information and related process, or constraint on deploying a given sensor device.

Finally, the observation result is described by using the `Result` module, which contains only a single concept, the `Result`. It is used for modeling a value of the `Observation`. The format of the value is coded according to the `FeatureOfInterest`. When the result is a simple value, (e.g., a single numerical data or a short text), the value can be attached directly to the `Observation` concept directly without using the `Result` concept too.

2.2 Multimedia Data Modeling and Retrieval

The method for modeling and retrieving multimedia data semantically (i.e., modeling multimedia data by means of its content meaning) has been the subject of interest since the appearance of digital multimedia

objects and multimedia encoding techniques (e.g., WAV, MP3, MP4, AVI). The main objective of these studies is to be able to model multimedia data in such a way that users can search for their desirable multimedia content semantically (e.g., searching for an image of a *red car* out of an image corpus). The common method that studies in this group used is to model the multimedia data by means of its low-level features. Major studies in this field can be categorized into two groups according to their choice of data model which are (i) Relation Database Oriented Multimedia Data Modeling and Retrieval, (ii) Linked Data-Oriented Multimedia Data Modeling and Retrieval and (iii) Ontology-Based Multimedia Data Modeling and Retrieval. The details are given in dedicated sub-sections as follows.

2.2.1 Relation Database Oriented Multimedia Data Modeling and Retrieval

Studies in this group choose to use a traditional relational database for modeling multimedia data. Low-level features are extracted, modeled and stored in a table within a relational database. The multimedia retrieval is done by mapping important keywords and low-level features within a database. For example, a *red car* keyword can be mapped to an image which contains an object with a specific kind of contour and the color descriptor of the content within the contour contains mostly red color. The studies in this group propose languages for users to model the mapping process as described. The languages can be categorized into two groups (i) predicate style and (ii) SQL Style Language. Major languages are the CVQL [34] and [35] detailed as follows.

2.2.1.1 CVQL [34]

CVQL is developed in a content-based video retrieval over video database. The syntax is given as follows.

```
{range; predicate; target}
```

The `range` is used for specifying the search space within a multimedia database. The `predicate` is a function for evaluating multimedia content that is stored within the `range`. The `target` is used for defining the target multimedia type that the user wants to retrieve. The multimedia type can be videos, shots or frames. The example statement is as follows.

```
{ smart_office_db;
  AP(unknown_face) [face_features !=
  <known_face_features>]; frames }
```

The CVQL predicate above is used for retrieving an unknown face within a smart home multimedia database. The predicate part is defined using the `AP` function which is a built-in function of the CVQL. This function is used for retrieving an absolute spatial position of a given subject. In this case, the statement tries to search for an absolute position of an unknown face. The evaluation criteria of an unknown face are to determine whether a set of facial features matches with any face within a database of the known faces or not.

The CVQL syntax is simple and intuitive. Hence, users can use the language to model and retrieve multimedia data easily. It also offers several built-in multimedia feature comparison functions for users to use for working with multimedia features of video data.

The main limitation of the CVQL is that it is binded to a video retrieval application. Thus, it is questionable whether can it be extended to support audio or image retrieval. The lack of complex event operators also limits this language to be able to define events that involve video data.

2.2.1.2 SVQL [35]

Similar to CVQL, SVQL is also developed in a context of video retrieval application. SVQL uses SQL style syntax to retrieve concerned contents within a video. SVQL extends SQL by proposing both content-based and spatiotemporal filter operators into SQL language. These operators are (i) feature specification operators, (ii) structure specification operators, and (iii) spatiotemporal specification operators. They are not given here in detail for the brevity of the discussion. The sample statement is given as follows.

```
SELECT m.video FROM MOVIE m
WHERE face1:OBJECT WITH m.video
AND (face1 IS "FACE")
AND (NOT(TEXTURE(face1, "knownface.bmp") > 0.75))
    --a <feature-specification>
```

The SVQL statement above is used for querying for a video file which contains a *known face*. The variable `m` is defined as a type `MOVIE`. The `m.video` is a video event within a movie that is returned from the query. The `face1:OBJECT WITH m.video` means that we define a variable `face1` as an object that is a part of a concern video event. The constraint `(face1 is "FACE")` is used for filtering only a face object out of all other objects within a video. The two final line is a feature called `Feature Specification` of SVQL. It filters for a frame within a video that has low-level features that are matched with a given constraint. In this case, the statement tries to filter only for a face that its texture match (75% similarity) with a known face that as given in `knownface.bmp`.

SVQL offers a language style that is similar to the SQL. Thus, users with SQL knowledge can learn SVQL with a short learning curve. Operators in SVQL are also fairly complete for both spatiotemporal and content-based video retrieval. Similar to CVQL, however, SVQL is also limited to video retrieval. Thus, it is questionable whether can it be used with audio or image data.

2.2.2 Linked Data-Oriented Multimedia Data Modeling and Retrieval

A linked data, similar to a graph database approach, refers to a method of modeling data as interlinked nodes in which each node describes data and edges between each node describes a relation between nodes. To date, a study which proposes a language for retrieving multimedia data in a linked data scenario which is SPARQL-MM [36]. The detail is given as follows.

2.2.2.1 SPARQL-MM [36]

SPARQL-MM is a language for retrieving multimedia data in a linked multimedia data scenario. SPARQL-MM assumes that multimedia data is modeled by using Resource Description Framework (RDF)[8]. In RDF, a data is modeled as a triple modeled as `<s,p, o>` which `s` and `o` are used for modeling resources or data, while `p` is used for modeling a relation between `s` and `o`.

The author of SPARQL-MM assumes that all the multimedia are modeled by decomposing multimedia data into smaller segments in which each segment is annotated with low-level features. To retrieve multimedia data, SPARQL-MM extends SPARQL language[9] and proposes spatiotemporal operators. he extension covers video and image. However, audio operators are not proposed. The example of SPARQL-MM statement is given as follows.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mm: <http://linkedmultimedia.org/sparql-mm/ns/1.0.0/function#>
```

[8] https://www.w3.org/RDF/

[9] SPARQL is a language for querying data from RDF triplestore.

```
SELECT ?t1 ?t2 WHERE {
    ?f1 rdfs:label ?t1.
    ?f2 rdfs:label ?t2.

    FILTER mm:rightBeside(?f1,?f2)
    FILTER mm:temporalOverlaps (?f1, ?f2)
} ORDER BY ?t1 ?t2
```

As seen in the example statement, the syntax of the SPARQL-MM is exactly with SPARQL. The extension which is proposed by SPARQL-MM is a set of functions proposed within `mm:` prefixed names-pace. The example statement uses two functions: *rightBeside* and *temporalOverlaps*. The *rightBeside* is a spatial function for filtering that two media segments are needed to be located right beside each other. The *temporalOverlaps* is used for filtering that time duration of two media segments are needed to be overlapped with each other. This means that the example segment as given above queries for two multimedia segments which located spatially right beside each other and happens during the same temporal period.

The current version of SPARQL-MM does not contain audio related operators. The lack of both complex event processing operators and the content-based retrieval aspect represent also some weakness to be considered.

2.2.3 Ontology Based Multimedia Data Modeling and Retrieval

Another important issue in multimedia sensor networks is the capability of understanding multimedia content in a machine interpretable manner. To achieve that, ontology is usually a preferable method that several previous works adopted. Adopting an ontology usually assumes that the SPARQL language is used for retrieving multimedia data in these studies. The ontology that is used in these studies are either specific to an application domain [37, 38] or generic to an application domain [39, 7, 40, 8, 41]. The application domain specific ontologies are not discussed in this section as we deemed them to be too limited to contribute to the development of a generic complex event processing for multimedia sensor networks.

The studies that propose a generic ontology for modeling multimedia data developed their ontologies by using the MPEG7 Standard [42]. The MPEG-7 is an XML-based multimedia content description interface that provides a set of low-level multimedia features (e.g., track length, frame rate, color, texture) that are classified under visual descriptors, audio descriptors, and descriptions schemes. The introduction of MPEG-7 helps to standardize the method for indexing and representing multimedia data. It also helps to separate the multimedia data representation model from a specific application domain. Hence, the interoperability (syntactic and semantic) issue among multimedia data can be overcome by adopting the same standard for indexing multimedia data. However, the weakness of the MPEG-7 standard lies in the fact that the MPEG-7 is an XML-based standard. It is not compatible to interoperate with other application domains that are modeled within the semantic web by using an ontology. To overcome this issue, several studies try to propose an MPEG-7 based ontology to make it possible to model and interoperate with the semantic. Main MPEG-7 ontologies are described as follows.

2.2.3.1 Hunter's MPEG-7 Ontology [39]

Hunter's MPEG-7 Ontology was proposed in 2001 [39]. Hunter proposes the ontology by manually translate the MPEG-7 standard into an ontology by using RDF. The latest version of Hunter's ontology is written in OWL. Hunter's MPEG-7 ontology is covered up with visual descriptors of MPEG7, while audio descriptors are not yet covered. Hunter's MPEG7 ontology was used in several digital libraries projects.

2.2.3.2 DS-MIRF Ontology [7]

DS-MIRF, also known as Tsinaraki's MPEG-7 Ontology, was proposed in 2004 by Tsinaraki [7]. The DS-MIRF ontology is written in OWL. This ontology covers MPEG-7 standard video and audio descriptors of MPEG-7 only partially. The usability of the DS-MIRF is demonstrated in a soccer match video database and Formula-1 video database retrieval applications. The DS-MIRF is used practically in several digital libraries and e-learning projects.

2.2.3.3 Rhizomik Ontology [40]

Rhizomik Ontology was proposed in 2005 by the company named Rhizomik Technologies [40]. Rhizomik uses an automatic XML to RDF conversion for translating MPEG-7 schema into RDF format. Rhizomik ontology is available in OWL. It fully covers all MPEG-7 descriptors. Rhizomik ontology was used in digital right management project and several e-business projects that are developed by Rhizomik Technologies Company.

2.2.3.4 Core Ontology for MultiMedia (COMM) [8]

COMM is an MPEG7 ontology proposed by Arndt in 2007 [8]. The ontology was developed by manually re-engineered MPEG7 schema into ontology in OWL. It covers MPEG7 up to visual descriptor, while audio descriptors are only partially supported. COMM is used in several multimedia analysis and multimedia annotation projects.

2.2.3.5 Ontology for Media Resource Annotation (MA-ONT) [41]

MA-ONT is an ontology propose by W3C as an attempt to generalize a data model for annotating multimedia content [41]. Unlike other MPEG7 ontology, MA-ONT chooses to support only the meta-data information of MPEG7. No audio or visual descriptor is supported. Instead, this ontology is designed to work with metadata of multimedia instead such as EXIF, Dublin Core, ID3, etc.

2.3 Complex Event Processing Engine

The significance of the Complex Event Processing (CEP) Engine on processing complex events in a multimedia sensor network is mentioned in the previous chapter. Previous studies in this field are mostly focused on a stream database context. Studies in this field are widely adopted in business activity monitoring and market data analysis application [15, 19]. Most of these studies propose a language for users to model events. The CEP engine takes the predefined event model provided by users and detect events accordingly.

The main components of a CEP language are (i) event operators and (ii) language syntax. The syntax of CEP languages in previous studies may be different. However, most of them share the similar set of complex event operators. The sub-section as follows describe the common set of operators that are shared in most of CEP languages. This sub-section follows by another dedicated sub-sections for describing the syntax and styles of CEP languages.

2.3.1 Complex Event Operators

In the context of a multimedia sensor network, operators that are necessary for modeling a complex event must support both spatial relation modeling and temporal relation modeling. However, due to the fact that previous studies are not developed under this context, the operators that they propose are solely related to temporal relation modeling only because the spatial relation is not considered as a crucial information

to model in a stream database or a business activity monitoring context. The choice of temporal event operators that are often adopted in previous studies are taken from the Allen's Temporal Algebra [43]. The Allen's Temporal Algebra proposes a set of operators for describing temporal relations between events. The list of operators that are proposed by Allen's Temporal Algebra is depicted in Figure 2.2.

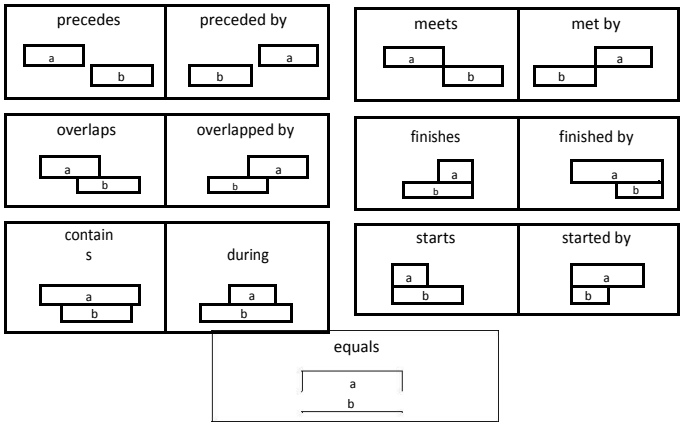


Figure 2.2 – Temporal Operators according to Allen's Temporal Algebra

The Allen's Temporal Algebra proposes thirteen temporal operators which consists of six pairs of direct/inverse operators (e.g., *a* precedes *b* and *a* preceded by *b*) and one *equals* operator. All of these operators can be effectively used for describing a relation between a pair of events in a historical data querying case (i.e., a case which every event is assumed to be terminated before detecting event relations). However, in a near real-time case, not every operator within the Allen's Temporal Algebra can be used due to the fact that some operator requires that an event is needed to be terminated first before such a relation can be detected. For example, it is not possible to detect that *a contains b* or *a finishes b* unless both events are properly terminated. However, it is possible to detect *a precedes b* without waiting for the event *b* to be terminated. For this reason, it can be seen that operators *finishes*, *finished by*, *contains*, *during* and *equals* cannot be used in a near real-time event detection case while, *precedes*, *preceded by*, *meets*, *met by*, *overlaps* and *overlapped by* can be used in a real real-time event detection case.

Some CEP language in previous studies chooses to adopt all the operators from the Allen's Interval Algebra, while some of them choose to adopt only some operators. The choice of operator name in some study is also different. The variation between each CEP language comes from the difference in the choice of language styles and usage context. The details of the different language styles of CEP languages are given in the next sub-sections.

2.3.2 Syntaxes and Styles of CEP languages

According to [19], The CEP language styles can be categorized into 3 categories. They are: (i) Event-Condition-Action(ECA) based language, (ii) SQL/SPARQL based Languages, and (iii) Logic Languages. The details of each language style are given as follows.

2.3.2.1 Event-Condition-Action (ECA) based CEP language

This group of languages chooses to follow the Event-Condition-Action (ECA) approach for designing their language. In short, ECA style languages allow users to define an event name, an event pattern (as a set of

conditions) and how to act when such an event is detected. Languages in this group are commonly used in an active database system. In this section, we choose to emphasis on three selected languages which are Snoop[44], CeDR[45] and SaSE. [46]. Snoop is one of the earliest languages in this group that has been developed. CeDR and SaSE are selected from their popularity as they have been cited often. Their details are as follows.

Snoop: An expressive event specification language for active databases[44]

Snoop was developed to use for detecting complex events within an active database. Snoop language syntax is given as follows:

```
On    ( <Event name> )
Condition <Filtering Constraint>
Action <Action name>
```

Snoop syntax consists of 3 parts. The `On` part is used for defining an event structure. The `Condition` part is used for defining a filtering constraint. The `Action` is used for indicating an action required when an event is detected. Snoop proposes temporal operators and periodic operators. Snoop does not adopt every temporal operator of Allen's temporal algebra. Only operators for expressing sequence (e.g., precedes/preceded by, overlaps/overlapped by) are adopted. Snoop provides a language with a simple and intuitive syntax.

Complex Event Detection and Response (CeDR) Language [45]

CeDR was developed under a context of a business event monitoring system. However, this language is also generic enough to be used in other application domain. The basic syntax of the CeDR is given as follows:

```
EVENT  <Event Name>
EVENT_PATTERN  <Event Pattern>
WHERE <Filtering constraint>
```

The `EVENT` part is used for defining the name of an event. An event structure is given in the `EVENT_PATTERN` part. The `WHERE` part is used for specifying filtering constraints. CeDR has all basic composition operators for defining event pattern. Beside these operators, CeDR also proposes an event lifetime operator and a detection window operator for adding temporal constraints.

SaSE [46]

SaSE was developed under the context of complex event processing over sensor data streams. SaSE was also demonstrated in a stock management application by using RFID tracker. The basic syntax of SASE is as follows:

```
EVENT <Event Pattern>
WHERE <Filtering Constraints>
WITHIN <Time Sliding Window>
RETURN <Output data>
```

The event structure is defined in the "EVENT" part. The "WHERE" part is used for defining a constraint. The "WITHIN" is used for specifying a time sliding window size (e.g., an event must happen within 5 minutes window time). The "RETURN" part is used for specifying which data field to show as an output. SaSE is recognized from its usage simplicity and the proof of a successful practical usage.

2.3.2.2 SQL/SPARQL Based Languages

This group of languages is developed to use with a continuous data stream management system. Such a system uses either a relational database management system or a linked data management system (i.e., RDF, OWL triplestore). CEP languages in this category are developed by extending either SQL or SPARQL. SQL is used in the case that a system is based on a relational database. SPARQL is used in a linked data case. Languages in this categories that are extended from the SQL include (CQL)[47], StreamSQL[48], and PIPES[49]. Languages that are extended from the SPARQL include C-SPARQL[50], SPARQL-ST[51], and SPARQLstream [52].

In this subsection, we choose to emphasize on two papers, the ESPER [17], and the EP-SPARQL [53]. The ESPER is selected as it is currently the most popular tool for CEP and it offers a SQL style language for defining events. The EP-SPARQL is selected as it is the most recent SPARQL based language that is designated for processing complex events.

ESPER[17]

ESPER is developed as an engine for detecting complex events in a relation stream database. ESPER is majorly used in business monitoring application. However, it is generic enough to use in wide range of application. A language that is proposed in ESPER is called ESPER Event Processing Language(ESPER EPL). The ESPER EPL adopts a syntax style from the SQL. Hence, it contains all the conventional SQL operators (e.g. SELECT, INSERT, UPDATE, CREATE or DELETE). The full syntax is not given in detail here for the sake of brevity. However, it can be found online in the official website[10].

ESPER engine and ESPER EPL are expressive as they contain all necessary event operators. The learning curve for ESPER-EPL can be quite fast if users are already known the SQL as the syntax of both of them are similar. However, even though the learning curve of ESPER EPL can be short for users that are already known the SQL, writing complex events definition in this language can be difficult as it has to be written as a nested query.

EP-SPARQL[53]

EP-SPARQL is developed to address the lack of a language for processing events in a linked data stream context.To the best of our knowledge, it is currently the most recent SPARQL style language that is designated for complex event processing in linked data streams. EP-SPARQL syntax is similar to SPARQL. EP-SPARQL addresses the issue of the lack of event structure operators in previous papers such as C-SPARQL[50] or SPARQL-ST[51]. The operators that are newly proposed in EP-SPARQL are "SEQ", "OPTIONALSEQ", "EQUALS" and "OPTIONALEQUALS". SEQ is a sequence operator. OPTION-ALSEQ is a temporal sensitive version of SEQ. EQUALS is a composition operator. OPTIONALEQUALS is a temporal sensitive version of EQUALS.

EP-SPARQL is addressed the necessity of temporal operators for constructing complex events in SPARQL. Users that are familiar with SPARQL language can also learn EP-SPARQL fast as the syntax of both of them are similar. However, defining complex events in EP-SPARQL can be complicated due to the difficulty that a nested query is also required to do so.

2.3.3 Logic Languages

Logic languages are a group of languages that choose to define events as a logic style formula (i.e. First order logic). Languages in this categories include Prova[54], RuleML[55], and XChangeEQ[9]. XChangeEQ is one of few languages that is inspired by an event calculus. Thus, it has a strong semantic foundation. ETALIS [10] is one of the most recent work and it is proved to be working well by under its demonstration example. Their details are discussed in subsections as follows

[10] http://www.espertech.com/esper/

XChangeEQ[9]

XChangeEQ is developed under the context of complex event processing in XML databases. The syntax of XChangeEQ is given as follows.

```
DETECT  <Event Name> { Required Parameters }
ON      <Event Structure>
WHERE   <Filtering condition>
END
```

In short, the syntax of XChangeEQ consists of three major parts and one query termination keyword. There are "DETECT, "ON" and "WHERE" for defining events, while "END" is used for terminating a query. The "DETECT" part is for defining a name of an event and necessaries function and variable. XChangeEQ is recognized as an expressive language for defining complex events in XML databases.

ETALIS[10]

ETALIS is developed as a generic complex event processing engine for continuous data streams. It can be used in wide range of applications such as stock trading, logistic or traffic control. The Syntax of ETALIS is written in a Fortran style function. Operators that are proposed in ETALIS are SEQ, AND, PAR, OR, DURING, STARTS, EQUALS, FINISHES and MEET. They are not described here for the brevity of our discussion. The reader can refer to [10] for more information concerning these operators

ETALIS provide a wide range of operators. ETALIS also prove to be usable in a pratical application. ETALIS is also available one of the few languages that publish its source code as an opensource project online[11]. The weakness of ETALIS is the fact that it adopts Fortran style language which can be difficult for end-users to use.

2.4 Comparative Analysis

So far, we described previous studies that are related to proposing a complex event processing engine for a multimedia sensor network. In this section, each of selected studies are being compared and analyzed against the requirements described in Chapter 1. The result of the comparative study is given in Table 2.1.

The MSNs Modeling and Event Modeling requirements within Table 2.1 are elaborated more into sub-requirements. The sub-requirements of MSNs Modeling requirement are related to (i) modeling list of sensors, (ii) modeling location maps, and (iii) modeling sensor capabilities. The sub-requirements of Event Modeling requirement are related to atomic event modeling and complex event modeling. The complex event modeling requirement is elaborated furthered into two sub-requirements related to spatial operators and temporal operators for modeling complex events. It can be seen from the table that there is no existing study that can completely address all the requirements that we give in Chapter 1. The studies on sensor network modeling which adopt a relational database [20, 21, 22] can be used for modeling multimedia sensor network infrastructure well. However, their hard-coded database schematic nature makes them not able to interoperate with several application domains efficiently. The study in [22] can be used for modeling images gathered from a bridge surveillance camera. However, the method to do so is considerably too limit to a bridge surveillance camera sensor. Hence, we still leave the multimedia sensor data modeling support to be unmarked as it cannot support the modeling of different types of multimedia sensors. The studies which adopt a graph database [23, 24, 25] are also have similar limitation as the schematic for modeling nodes and edges are also hard-coded. On the other hand, the *SSN* ontology can be used for modeling multimedia sensor networks, and can interoperate with external application domain. None of the studies, which are related to sensor network modeling, can satisfy the rest of other requirements.

[11] https://code.google.com/archive/p/etalis/

Table 2.1 – Comparative Analysis of Previous Studies Against the Requirements on processing complex events in a multimedia sensor network

Related Study	Requirement									
	MSNs Modeling			Multimedia Sensor Data Modeling	Event Modeling			Near Real-Time Event Detection	Event Detection in Historical Data	Application Domain Interoperability
	List of sensors	Location information	Sensor capabilities		Atomic Event Modeling	Complex Event Modeling				
						Spatial Operators	Temporal Operators			
Sensor Network Modeling										
FJording	X	X	X	-	-	-	-	-	-	-
IrisNET	X	X	X	-	-	-	-	-	-	-
Bridge Monitoring	X	X	X	-	-	-	-	-	-	-
G*	X	X	X	-	-	-	-	-	-	-
GeoSensor	X	X	X	-	-	-	-	-	-	-
WMSN Model	X	X	X	-	-	-	-	-	-	-
SSN-Onto	X	X	X	-	-	-	-	-	-	X
Multimedia Data Modeling and Retrieval										
CVQL	-	-	-	X	X	X	-	-	X	-
SVQL	-	-	-	X	X	X	X	-	X	-
SPARQL-MM	-	-	-	X	X	X	X	-	X	-
Hunter's Ontology	-	-	-	X	X	-	-	-	X	X
DS-MIRF Ontology	-	-	-	X	X	-	-	-	X	X
Rhizomik's Ontology	-	-	-	X	X	-	-	-	X	X
COMM Ontology	-	-	-	X	X	-	-	-	X	X
MA-Ont	-	-	-	X	-	-	-	-	-	X
Complex Event Processing Engine										
Snoop	-	-	-	-	X	-	X	X	X	X
CeDR	-	-	-	-	X	-	X	X	X	X
SaSE	-	-	-	-	X	-	X	X	X	X
ESPER	-	-	-	-	X	-	X	X	X	X
EP-SPARQL	-	-	-	-	X	-	X	X	X	X
XChangeEQ	-	-	-	-	X	-	X	X	X	X
ETALIS	-	-	-	-	X	-	X	X	X	X

Every study in multimedia data modeling and retrieval field can be used for modeling multimedia sensor data as they are capable of modeling multimedia data itself. However, modeling sensor networks or multimedia sensor networks is considered as out of the scope of these studies. All of these studies, except the *MA-Ont*, can be used for modeling an atomic event and detecting historical events. However, they are not designed for near real-time detection case. Only CVQL, SVQL and SPARQL-MM can be used for modeling complex events as they propose a language for doing so. However, CVQL does not contain any temporal operator because their language syntax is only a simple predicate based style and cannot be used for representing a complex temporal relation pattern. The SVQL and SPARQL-MM can support both spatial and temporal relations modeling. The *MA-Ont* is the only study in multimedia data modeling and retrieval that can support multimedia sensor data modeling and application domain interoperability requirement. The reason comes from the context of the development of this ontology which focuses only on modeling multimedia metadata.

Every study in complex event processing engine group can support most of the requirements except multimedia sensor networks Modeling, Multimedia Sensor Data Modeling, and Spatial Relation Modeling. The reason comes from the fact that these studies are designed to process complex events regardless the choice of an application domain. Hence, they can overcome every requirement that is related to event processing but, they fail to satisfy the requirements that are specific to the multimedia sensor networks context. It is noted that none of these studies propose any spatial relation operators for modeling spatial relations since they are designed for a stream database or a business monitoring application.

From this comparative analysis, we conclude that none of the existing studies can fully address the identified requirements on complex events modeling and detection in multimedia sensor networks. Hence, in this work, we propose an ontology-based framework aimed to reach all these requirements.

Chapter 3

CEMiD Framework: Overview

To satisfy requirements on processing complex events in multimedia sensor networks (as identified in Chapter 1), we propose a framework so-called *CEMiD* to support Complex Events Modelling and Detection in multimedia sensor networks. This chapter is dedicated to the description of the general idea and the architecture of *CEMiD*. The details of the framework are later elaborated in Chapter 4, 5, and 6, where each of framework component is described.

3.1 Architecture Overview of the CEMiD Framework

The main objective of the *CEMiD* framework is to allow users in different roles to be able to freely define events according to their needs, while keeping low-level architecture to be the same regardless users' role or application domain.

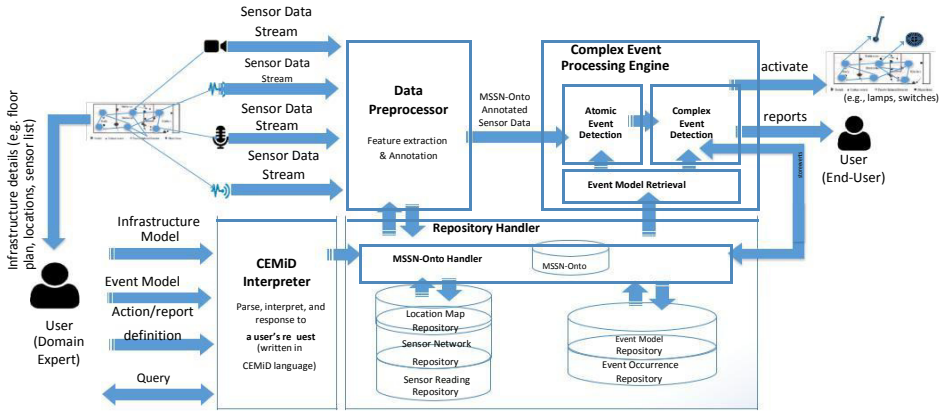


Figure 3.1 – Architecture of the *CEMiD* framework

As shown in Figure 3.1, the architecture of the *CEMiD* framework is outlined in four main modules: (i) Repository (ii) CEMiD Intrepreter (iii) Data Preprocessor, and (iv) Complex Event Processing Engine. Their details are described as follows.

3.1.1 Repository

The Repository is defined to store all sensed data. component in which all data is stored. The five types of data are stored in their respective repositories:

- Location Map: is used to store the description of the floor plan of the deployment site of the sensor networks;
- Sensor Network: stores essential information concerning sensor network infrastructure. The data that is needed to be modeled includes: the list of sensors within the network, capacities of each sensor, and the location that each sensor is deployed in. Capacities of each sensor should be modeled according to its hardware specifications. An example of capacities includes output format types, accuracy, precision, drift, or battery capacities. The location that each sensor can be installed into should be related to a predefined location map;
- Sensor Reading: All the raw sensor readings from the framework are processed and stored within the sensor reading repository. All the sensor readings are modeled in such a way that related data can be retrieved and translated easily during the event detection, and query processing;
- Event Condition: stores conditions that the framework can use for detecting events. These condi-tions are modeled by users;
- Event Occurrence: stores actual occurrences of events according to predefined event conditions provided by users.

All the data within the *Repository* is modeled and stored by using an ontology for modeling multimedia sensor networks. Due to the lack of a suitable ontology for modeling multimedia sensor networks, we propose a new ontology so called the *Multimedia Semantic Sensor Network Ontology (MSSN-Onto)*. The *MSSN-Onto* extends the *SSN* ontology to provide full-fledged capabilities for modeling data for each of the repositories within this module. The ontology handler of the Repository module is used to cope with all the *MSSN-Onto* related operations which include data modeling, data storing and querying. The detail of the *MSSN-Onto* is described in Chapter 4.

3.1.2 CEMiD Interpreter

CEMiD Intepreter serves as a high-level interface to help users to interact with the framework. Our framework proposes a language so called *CEMiD* language for modeling their needs. The functionalities of CEMiD language are given as follows.

- Data and Event Modeling for Multimedia Sensor Networks: The *CEMiD* language can be used for modeling all the data which users need to model and store within the *Repository*. The data which can be modeled by the *CEMiD* language are: (i) a location map, (ii) a sensor network infrastructure, (iii) an event condition.
- Reports and Action Modeling: The *CEMiD* language can also be used for modeling reports and actions that the framework needs to be carried out when a certain event is detected;
- Historical Event Querying: Users can also use the *CEMiD* language to fetch events from the gathered historical data;
- Compact and Intuitive Syntax: *CEMiD* language is inspired from SQL/SPARQL language. The syntax is designed in such a way that users with a background in SQL and SPARQL use the language with minimal learning curve.

All the statements provided by users in *CEMiD* language cannot be used by the framework directly. It is required to be translated into the form that is understandable by the framework. Due to the fact that our framework is an ontological-based framework, we choose to translate *CEMiD* language statements into SPARQL queries. Hence, all the translated queries can be used directly by all of the repositories in the *Repository* of the framework. The translation process from *CEMiD* language to SPARQL is done by using the *MSSN-Onto* Handler module. The details of *CEMiD* language are described later in Chapter 5.

3.1.3 Data Preprocessor

This component is responsible for preprocessing and annotating incoming sensor data streams by using *MSSN-Onto*. This module operates automatically in response to every incoming raw sensor reading. In the case that an incoming raw sensor reading is from a scalar sensor, a raw data value can be annotated by *MSSN-Onto* directly. However, in a multimedia sensor case, all multimedia sensor readings need to be decoded, having their low-level features extracted and annotated before being modeled by *MSSN-Onto*. The features are extracted according to the MPEG7 recommendation. Outputs of this module are simultaneously pushed to the complex event processing engine module and stored into the sensor reading repository.

3.1.4 Complex Event Processing Engine

The Complex Event Processing Engine is responsible for detecting events as defined by the users through the *CEMiD* language. The detection process is launched automatically in response to every incoming sensor readings. Our engine is capable of detecting events in a near real-time manner. The engine relies on a pipelining-based mechanism which allows the framework to integrate all the multimedia data processing, atomic event detection, and complex event detection tasks. The internal mechanism for handling all the event detection pipelines within *CEMiD* framework is called the *GST-CEMiD*. The details of the *GST-CEMiD* and all its related algorithms for processing events are described in Chapter 6.

3.2 Discussion

So far, we provided an overview of the architecture of the *CEMiD* framework. The *CEMiD* framework proposes four main modules to handle all the tasks that are related to processing complex events in multimedia sensor networks. These modules are developed on the top of three elements of the framework which are (i) *MSSN-Onto*, (ii) *CEMiD* Language, and (iii) Complex Event Processing Engine. The comparative analysis on how they can address requirements on complex event processing in multimedia sensor networks (see Section 1.3) is given in Table 3.1.

The *MSSN-Onto* is designed to support MSNs modeling. It is also capable of modeling both atomic and complex events. Moreover, it can be used in different kinds of application domains. Hence, we mark *X* for every requirement except the near real-time event detection requirement and event detection from historical data requirement because both of them are more related to the complex event processing and detection than to data modeling.

The *CEMiD* language is designed to support MSNs modeling, event modeling, near real-time event detection, and application domain interoperability. Hence, we mark *X* for all these supported requirements. The only requirement that is not supported by the *CEMiD* language is the *Event Modeling* requirement because it is done automatically through the data preprocessor using the *MSSN-Onto* as a basis.

The *Complex Event Processing Engine* is designed for processing near real-time event detection, event detection from historical data and, application domain interoperability. Hence, all these three requirements are marked *X* within the table. The other requirements are unmarked (i.e., unsupported) as they are more related to data modeling related requirements than complex event processing related tasks.

Table 3.1 – Comparative Analysis of CEMiD Framework Features Against Requirements on Complex Event Processing in MSNs

Requirements	MSSN-Onto	CEMiD Language	Complex Event Processing Engine
MSNs Modeling			
list of sensors	X	X	-
location maps	X	X	-
sensor capabilities	X	X	-
Multimedia Sensor Data Modeling	X	-	-
Event Modeling			
atomic event modeling	X	X	-
complex event modeling			
- spatial operators	X	X	-
- temporal operators	X	X	-
Near Real-Time Event Detection	-	X	X
Event Detection in Historical Data	-	X	X
Application Domain Interoperability	X	X	X

We dedicate the three following chapters for describing the details of each of the core elements of the framework.

Chapter 4

MSSN-Onto: An Ontology for Modeling Multimedia Sensor Networks

As we stated in the previous chapter, our *CEMiD* framework bases its functionalities on a core ontology. In this chapter, we present our ontological data model for *CEMiD*, called the *Multimedia Semantic Sensor Network Ontology (MSSN-Onto)*. *MSSN-Onto* is designed not only to save our framework, but also for modeling multimedia sensor networks itself in general without relying on the framework. This means that sensor network developers can also reuse the *MSSN-Onto* in their applications without adopting our whole framework.

Recall from Chapter 2, there is no existing ontology which can be used for modeling multimedia sensor networks yet. There exists the *SSN* ontology [29] which proposes by W3C to be used as a generic ontology for modeling sensor networks. However, its capability for supporting multimedia sensor network modeling is limited. Hence, in order to avoid re-developing *MSSN-Onto* from scratch, we choose to develop our *MSSN-Onto* by extending from *SSN* ontology. Our *MSSN-Onto* extends the *SSN* ontology by proposing several new concepts to support multimedia sensor network modelings. This chapter begins with a section dedicated to the analysis of missing features of the *SSN* ontology for supporting multimedia sensor networks modeling. Afterward, we present the detailed description of our *MSSN-Onto*. Then, we recommend instructions on how *MSSN-Onto* can be aligned with an application domain without relying on the *CEMiD* framework. This chapter ends with a section for validating the capacities of *MSSN-Onto* for modeling multimedia sensor networks.

4.1 Analysis of the missing features of the SSN ontology

The first step in developing *MSSN-Onto* is to analyze the existing features of the *SSN* ontology and identify the missing features which prevents the it to fully support multimedia sensor network modeling. We briefly summarize the *SSN* ontology before addressing the missing features and introducing our extensions. According to the latest published version[1], the *SSN* ontology is modularized into *SOSA* (Sensor, Observation, Sample, and Actuator), a light-weight, yet self-contained core ontology for modeling a sensor network, and *SSN* ontology, which imports the *SOSA* ontology and proposes more concepts to describe a sensor network with more details. In what follows, we refer to the full *SSN* ontology as *SOSA/SSN* ontology, while the term *SSN* ontology is used when we refer to concepts of the *SSN* ontology

[1] Latest published version on 19 Octoberr 2017: https://www.w3.org/TR/vocab-ssn/

which does not exist within the *SOSA* ontology.

The *SOSA/SSN* ontology can model sensor networks using three different kinds of modeling perspectives, depending on the user choice: (i) the *Observation* perspective, which models a sensor network as a set of devices that can produce sensor readings; (ii) the *Actuator* perspective, which is used for modeling a network of actuators[2]; and (iii) the *Sampling* perspective, which models a sensor network as a set of devices which can produce data periodically. We choose to extend the *SOSA/SSN* ontology into *MSSN-Onto* from the *Observation* perspective. Our choice is related to the fact that a multimedia sensor is not always an actuator nor a device which periodically produces data. The *Observation* perspective of the *SOSA/SSN* ontology is given in Figure 4.1.

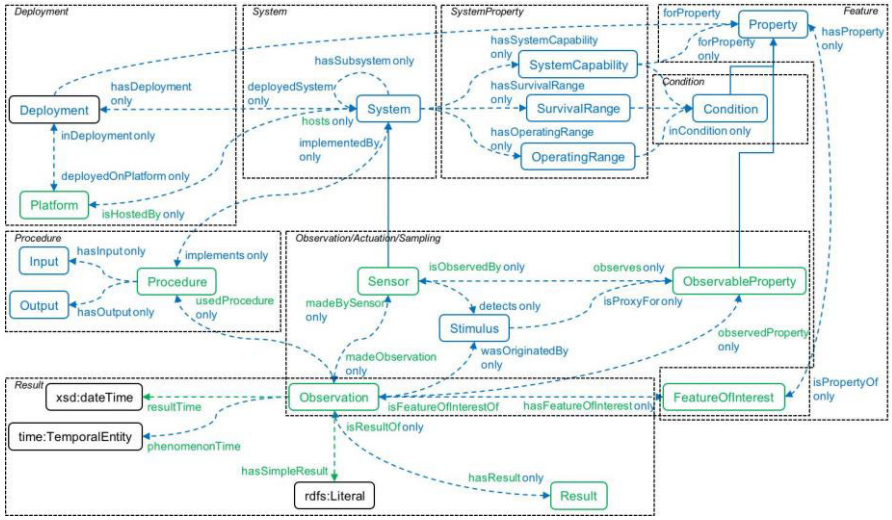


Figure 4.1 – The *SOSA/SSN* Ontology (Observation Perspective)

Briefly, the *SOSA/SSN* ontology describes a sensor network through seven main modules which cover different aspects of a sensor network. Sensor descriptions and capabilities are defined using the following modules:

- `Observation/Actuation/Sampling` module: contains the fundamental concepts for model-ing a sensor network, which are `Sensor`, `Stimulus`, `Observation`, `ObservableProperty`, and `FeatureofInterest`. In short, a sensor is modeled using the `Sensor` concept. A `Sensor` observes a `FeatureOfInterest` from a `Stimulus` and produces a result as an `Observation`. A possible kind of value or feature that a sensor can observe is modeled using `ObservableProperty`; This module also contains concepts that are related to *Actuator* and *Sampling* perspectives. However, these concepts are not depicted in Figure 4.1 as they are not related to the *Observation* perspective;

- `Feature` module: helps to model all properties and features that are used within a sensor network. The properties and features can either related to sensor capabilities (e.g., output format, precision, accuracy) or features that a given sensor can observe (e.g., temperature, brightness);

- `System` module: helpts to model a sensor network as a system which is composed of multiple sensors. A system can also be divided into multiple sub-systems by using an object property `hasSubSystem`. A system or a sub-system itself can also be a sensor;

[2] An actuator is a device which reacts to a certain event (e.g., a lamp reacts when a motion sensor detects a movement).

- `Procedure` module: helps to model a sensing method of a sensor. Such a method has an input information described by the `Input` concept, and has an output information describes by the `Output` concept.

The operating restriction of a sensor network is described by:

- `SystemProperty` module: helps to model properties and operating constraints of each sensor within a sensor network and also the network itself. In short, the capabilities of a sensor (e.g., accuracy, precision) are modeled using the `SystemProperty` concept. Survival constraints and operating constraints are modeled using `SurvivalRange` and `OperatingRange` concepts;

- `Condition` module: helps to model a sensing constraint of a sensor as a constraint block. For example, a wind speed sensor can be attached to a constraint block to indicate that it can only capture data when the wind speed is between 10-60m/s;

The deployment related information is described by the following modules:

- `Deployment` module: A sensor is expected to be deployed at some specific location. This module is used for describing such information and related process, and constraints on deploying a given sensor device.

Finally, the observation result is described using the `Result` module, which contains only a single concept, the `Result`. It is used for modeling a value of the `Observation`. The format of the value is coded according to the `FeatureOfInterest`. When the result is a simple value (e.g., a single numerical data or a short text), the value can be attached directly to the `Observation` concept without using the `Result` concept.

Our analysis suggests that the missing *SOSA/SSN* ontology features are as follows:

- Floor Plan/Coverage Area Modeling: The *SOSA/SSN* ontology has *Deployment* module for modeling a deployment location of a sensor. However, the comprehensive information of the deployment site (such as a complete floor plan, possible coverage areas of sensors, or a relation between locations, cannot yet be modeled within the *SOSA/SSN* ontology);

- Multimedia Sensor Modeling: The *Sensor* concept of the *SOSA/SSN* ontology can be used for modeling a sensor in general regardless if it is a scalar or a multimedia sensor. However, the capabilities of multimedia sensor (such as output data type, encoding method, mobility property) are usually different from a scalar sensor. Hence, it is preferable to propose more concepts for modeling a multimedia sensor and capabilities of a multimedia sensor;

- Multimedia Sensor Data Modeling: The *Result* of the *SOSA/SSN* ontology is used for modeling a sensor reading from a sensor. The ontology specifies that a sensor reading should be modeled either as an RDF literal or by using the concept *Result*. This can be used for modeling a simple scalar value. However, it cannot be used for modeling a multimedia data as a sensor reading semantically (i.e., modeling a multimedia data according to its low-level features). Hence, new concepts for modeling multimedia sensor data is needed to be proposed;

- Atomic/Complex Event Modeling: The atomic and complex event modeling is not within the scope of the *SOSA/SSN* ontology development. Hence, these features are not yet supported within the *SOSA/SSN* ontology. New concepts for modeling atomic and complex events are needed as well.

The main reason why these following features are not included within the *SOSA/SSN* ontology lies in the scope of the ontology definition. The *SOSA/SSN* ontology considers multimedia sensor and multimedia data as application specific requirements. Hence, these missing features are not included within the core

SOSA/SSN ontology. The *SOSA/SSN* ontology document suggests that any missing features of the ontology itself can be proposed by aligning the ontology with an external application specific ontology. However, this can limit the generality of the ontology as the aligned external ontology itself is already limited to a specific application domain. In this work, we propose *MSSN-Onto* which extends the *SOSA/SSN* ontology to address all the missing features of the *SOSA/SSN* ontology for modeling multimedia sensor networks. The details of the ontology are described in the following section.

4.2 Multimedia Semantic Sensor Network Ontology (MSSN-Onto)

MSSN-Onto is developed by extending the *SOSA/SSN* ontology, adding several new concepts, and aligning some part of the ontology with external ontologies.

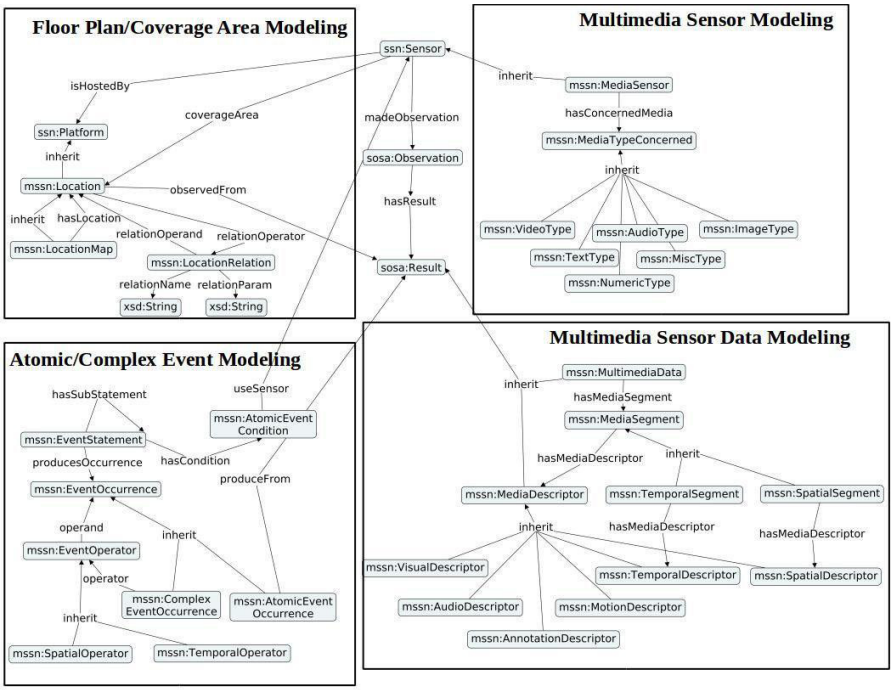


Figure 4.2 – Multimedia Sensor Network Ontology (*MSSN-Onto*)

Our *MSSN-Onto* ontology is depicted in Figure 4.2. In what follows (also in Figure 4.2), every ontology concept is denoted in the format `<prefix>:<concept_name>`, where `prefix` is a an abbreviation name of the ontology, and `<concept_name>` is the name of the concept. For example, `ssn:System` refers to the `System` concept within the *SSN* ontology, while `sosa:Sensor` refers to the `Sensor` concept of the *SOSA* ontology. The list of ontology prefixes that we use in this chapter are as follows.

- *sosa:* is a prefix for the *Sensor-Observation-Sampling-Actuator* ontology (a minimal version of the *SSN* ontology proposed by W3C);
- *ssn:* is a prefix for the *Semantic Sensor Network* ontology;
- *mssn:* is a prefix for the *Multimedia Semantic Sensor Network Ontology* (our proposal);

- *app:* is a prefix for an application specific ontology such as a smart meeting room or a smart office ontology (for illustration purpose only).

The extensions that we have done to the *SOSA/SSN* ontology to propose *MSSN-Onto* can be categorized into four main extensions according to the missing features as given in the previous section. They are described as follows.

4.2.1 Floor plan/coverage area modeling extension

This extension allows *MSSN-Onto* to be able to model a floor plan of the sensor network deployment site, and all potential coverage areas that sensors within the network can cover. To do so, we extend the `sosa:Platform` concept into `mssn:Location` and `mssn:LocationMap`. Briefly, a `mssn:Location` is used for modeling a location which a sensor within the network can either be deployed into or can be observed (i.e., a coverage area). A `mssn:LocationMap` is used for modeling a floor plan as a map of multiple locations. The relation between each location within the `mssn:LocationMap` is expressed using a `mssn:LocationRelation`. *MSSN-Onto* comes with a predefined set of location relations which can be categorized into three categories[3]:

- Topological Relations: This group of relations identifies the topological structure between pairs of locations. Nine predefined topological relations are provided: `equals`, `disjoint`, `intersects`, `touches`, `contains`, `covers`, `covered by`, `within`, `overlaps`, and `crosses`.
- Distance Relations: This group of relations indicates the distance information between each pair of locations. The predefined distance relations in our framework are: `closeTo` and `farFrom`.
- Directional Relations: This group of relations indicate the direction from one location to another. The predefined directional relations are: `leftOf`, `rightOf`, `opposite`, `above`, `below`.

Formal definitions of `mssn:Location` and `mssn:LocationMap` concepts are given in Def. 1 and Def.2. It is to be noted that our data model is an ontology. Hence, Internationalized Resource Identifier[4] (IRI) is used as an identifier to every concepts and instances within the ontology as it is the currently the most adopted approach for modeling data within an ontology. Therefore, Def. 1, Def.2 and all other definitions given in this chapter includes at least one *iri* component for modeling its IRI identifier.

Definition 1 Location: *A location, denoted as l, is a 3-tuple for describing a location, defined as* $l =< iri, n, p >$*, where:*

- *iri: is an identifier to a location l given in IRI format;*
- *n : is a location name given in a plain text format;*
- *p : is a a value that is associated with the location n (e.g., GPS coordinate, room name, floor number). The format of pi depends on the application domain;*

Definition 2 Location Map*: A location map, denoted as LM, is a directed graph, defined as* $LM =< iri, V, E, F >$*, where as:*

- *iri: is an identifier to a location map LM given in IRI format;*

[3] each set can be extended as of the needs of the applications

[4] https://tools.ietf.org/html/rfc3987

- *V : is a set of vertices representing locations, such that* $V = \{l_1, l_2, ..., l_n\}$, $\forall l_i \in V$, $l_i = < iri_i , n_i , p_i >$ *(Def. 1);*
- *E: is a set of edges that establishes relations between two vertices,* v_i *and* v_j *;*
- *F:V xV ⇒ R: is k-dimensional relation, indicating maximum k relations between two vertices, such that:*

  $< v_i , v_j > \in E \Leftrightarrow F(v_i , v_j) = \bigcup_{m=1}^{k} \{r_{ijm} \mid r_{ijm}$ *is a text describing the relation name between* $v_i , v_j \in V\}$

In short, the `mssn:LocationMap` concept is used for modeling location information as a directed graph with named edges. Each node within the graph represents a location. Edges between each pair of locations represent relations between them. Relations can be multi-dimensional modeled using multiple edges. Users can freely define an edge name when describing a location relation.

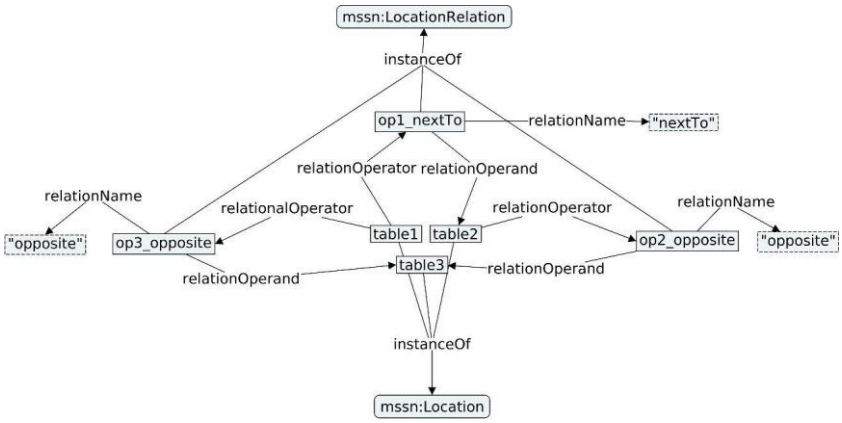


Figure 4.3 – Illustration on location map modeling by using the *MSSN-Onto*

Figure 4.3 depicts how to use *MSSN-Onto* to model a simple location map with three locations. In this case, we model a location map for modeling an office room with three tables. Two tables are located next to each other, while another table is located opposite to both tables. In this case, we models three instances of `mssn:Location` which are `table1`, `table2` and , `table3` for modeling tables within the office. To model relations between them, three instances of `mssn:LocationRelation` are created. `op1_nextTo` is used for modeling that `table1` is next to `table2`. `op2_opposite` is used for modeling that `table2` is opposite to `table3`. `op3_opposite` is used for modeling that `table1` is opposite to `table3`.

4.2.2 Multimedia sensor modeling extension

In *SOSA/SSN* ontology, the *sosa:Sensor* is proposed for modeling a sensor. It is intended to be used as a generic concept for modeling every sensor within a sensor network application. However, in the multimedia sensor networks context, one generic concept for all kind of sensors may not be sufficient as a sensor in this context can be categorized as either a scalar sensor or a multimedia sensor. Therefore, for the sake of clarity on defining a sensor type, another concept, the `mssn:MediaSensor`, is proposed within *MSSN-Onto* (see Figure 4.2). Users of *MSSN-Onto* should instantiate a scalar sensor through the `sosa:Sensor`, while a multimedia sensor should be instantiated from the `mssn:MediaSensor`. Both of these concepts can be formalized in one unified formal definition as given in Def. 3.

Definition 3 Sensor: *A sensor, denoted as s, is an entity that can sense data (scalar or multimedia). It is defined as a 7-tuple, s =< iri, plt, M, m, FI , i, CP, CA > where:*

- *iri: is an identifier to a sensor s given in IRI format;*
- *plt: is the platform where a sensor is deployed to, which is a location within a predefined location map, plt ∈ LM.V (Def. 2);*
- *M: is a set of media types that s concerns. M = {m_1, m_2, m_3, ..., m_n} | m_i ∈ {V ideo, Audio, I mage,-T ext, N umeric};*
- *m: is an IRI to a sensing method that a sensor s is used;*
- *FI: is the set of features of interest that s can observe. FI = {fi_1, fi_2, fi_3, ..., fi_n}, ∀fi_i ∈ FI , fi_i is a name of a feature (e.g., temperature, brightness, motion, face);*
- *i: is the stimulus that a sensor s observes before triggering its data capturing function;*
- *CP: is the set of measurement capabilities such that CP = {cp_1, cp_2, cp_3, ..., cp_n} cp_i ∈ CP, cp_i is a sensing capability of s such as accuracy, resolution, latency;*
- *CA: is the set of coverage areas of s, modeled as CA = {l_1, l_2, l_3, ..., l_n}, ∀l_i ∈ CA, l_i ∈ LM.V (Def. 2);*

The `mssn:MediaSensor` is used as a generic class for modeling a multimedia sensor. An instance of `mssn:MediaSensor` can have one or more `mssn:MediaTypeConcerned` for modeling types of media that a given multimedia sensor can produce. Types of data that a multimedia sensor can pro-duce are *video, audio, image, text, numeric* and *miscellaneous*. They are modeled by using concepts `mssn:VideoType, mssn:AudioType, mssn:ImageType, mssn:TextType, mssn:Nu-mericType` and `mssn:MiscType`. Note that the `mssn:MiscType` type is used in the case that a pro-duced multimedia type is neither audio, image, text, nor numeric (e.g., a PDF file). The *MSSN-Onto* comes with four predefined sensor types: `mssn:VideoSensor`, `mssn:AudioSensor`, `mssn:AudioVid-eoSensor` and `mssn:ImageSensor`. They are depicted in Figure 4.4

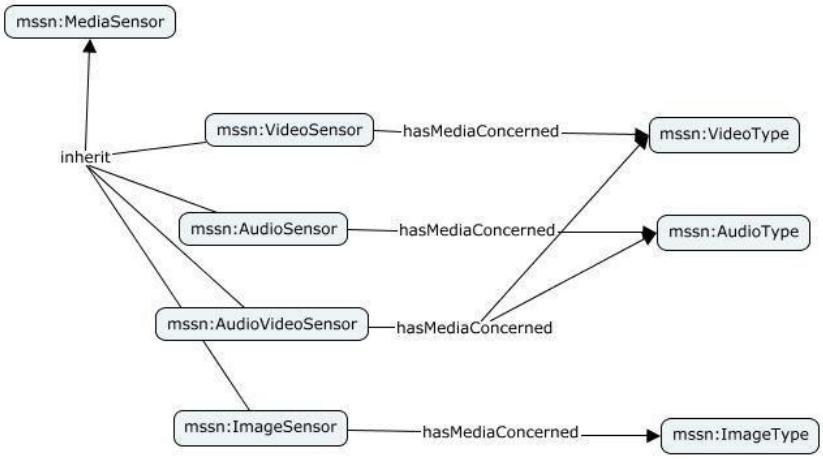


Figure 4.4 – Predefined Sensor Type in *MSSN-Onto*

`mssn:VideoSensor` is a concept that is used as a generic concept for every video sensor such as a video camera, a webcam or an embedded camera module. This kind of sensor produces a stream of video as an output. Hence, it has `mssn:VideoType` as its concerned media type. `mssn:AudioSensor`, `mssn:AudioVideoSensor` and `mssn:ImageSensor` are also proposed to be used as a generic concept for modeling a sensor of their respective type. It is noted that `mssn:AudioVideoSensor` is designed to be used for modeling a sensor that can produce both audio and video. Hence, it has both `mssn:VideoType` and `mssn:AudioType` as its media concerned. The illustration on how we can instantiate sensors from *MSSN-Onto* is given in Figure 4.5.

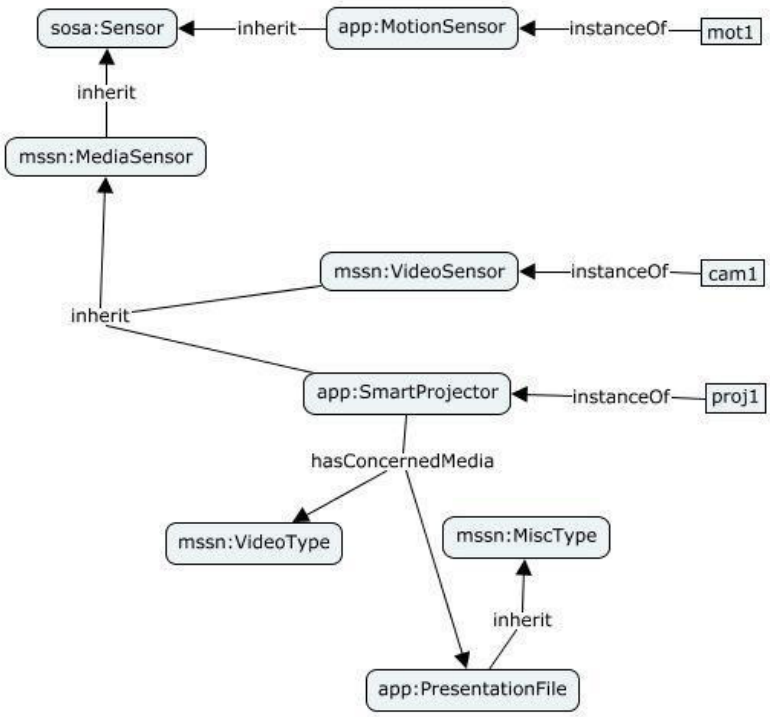


Figure 4.5 – Illustration on instantiating sensors in the *MSSN-Onto*

Figure 4.5 depicts the instantiation of three sensors, a motion sensor, a video camera and a smart projector. An instance `mot1` is instantiated from a concept `app:MotionSensor`, an application specific concept designated for modeling a motion sensor. This sensor is a scalar sensor so, `app:MotionSensor` is inherited from `sosa:Sensor`. Another instance, `vid1` is instantiated from `mssn:VideoSensor` for modeling a video camera. The last instance, `proj1`, in Figure 4.5 illustrates how a special kind of multimedia sensor can be modeled. In this case, we choose to model a smart projector sensor. This sensor is a device which can capture a video footage of the presentation and record the presen-tation file that is used during the presentation. The concept `app:SmartProject` is proposed by inheriting from `mssn:MediaSensor` to indicate that it is an application specific multimedia sensor type. It has `mssn:VideoType` and `app:PresentationFile` as its concerned media type. The `app:PresentationFile` is proposed by inheriting from `mssn:MiscType` as a presentation file is neither a video, an audio, an image, a text or a numerical data.

After all the sensors (scalar or multimedia) are properly modeled, users need to group multiple sensors together to form a sensor network, modeled using `ssn:System` concept. Formal definitions for the

`ssn:System` concept is given in Def. 4.

Definition 4 System: *A system is an aggregation of multiple sensors. It is defined as sys =< iri, plt, SS > where as:*

- *iri: is an identifier to a system sys given in IRI format;*
- *plt: is the platform where sys is deployed to, which is a physical location where plt ∈ LM.V (Def. 2);*
- *SS: is the set of sensors, SS = {s_1, s_2, s_3, ..., s_n}, ∀s_i ∈ SS, s_i is a sensor (Def. 3).*

It is noted that the original *SOSA/SSN* ontology models a sensor network only as a collection of sensors represented by `ssn:System`. Additional information (e.g., topology, connection method) is not used when modeling a sensor network. We also choose to follow the same method for modeling a sensor network in *MSSN-Onto*. Nevertheless, we can extend the definition of `ssn:System` in the future to model more sensor network related information.

4.2.3 Multimedia sensor data modeling extension

The *SOSA/SSN* models every incoming sensor reading by using an `sosa:Observation` concept. An `sosa:Observation` contains information regarding (i) which sensor is used, (ii) an observation result value, (iii) which feature, object or phenomenon that the result value is describing, (iv) a related coverage area (v) a time of the observation, and (vi) a duration that the observation is covered up to. The formal definition of an `sosa:Observation` concept is given as follows.

Definition 5 Observation: *An observation, denoted as o, is the information that a sensor produces for every triggering of its sensing method, It is defined as a 7-tuples, o =< iri, s, f i, r, ca, t, t' >, where:*

- *iri: is an identifier to an observation o given in IRI format;*
- *s: is the sensor that is used for producing o (Def. 3);*
- *f i: is the feature of interest of o, f i ∈ s.FI;*
- *r: is an outcome of the observation o encoded in a format according to f i;*
- *ca: is the coverage area of o and ca ∈ s.CA;*
- *t: is the timestamp of the observation o;*
- *t' : is the duration that observation o (can be empty with instant observation).*

In a scalar sensor data case, a sensor reading value can be attached to an `sosa:Observation` concept directly. Hence, the extension to the *MSSN-Onto* is not neccesary for modeling a scalar reading. However, concepts for modeling multimedia sensor readings are not given in the *SOSA/SSN* ontolgy. Hence, an extension is needed to be proposed. The *MSSN-Onto* extends the *SOSA/SSN* ontolgy and propose `mssn:MultimediaData`, `mssn:MediaSegment`, and `mssn:MediaDescriptor` concept (see Figure 4.2). Their formal definitions are given as follows.

Definition 6 MediaSegment: *A media segment, denoted as ms, is a segment of a multimedia data, defined as a 6-tuple ms =< iri, p, t, t', MD, SMS >, where as:*

- *iri: is an identifier to a media segment ms given in IRI format;*

- *p: indicates the type of the media segment ms and* $p \in \{Video, Audio, Image, Text\}$;
- *t: is the timestamp of the beginning of the media segment ms;*
- t': *is the duration of the media segment ms. If* $t' = null$, *this means that ms does not have duration (e.g., a segment of an image or a single video frame);*
- *MD: is a set of media descriptors, such that* $MD = \{md_1, md_2, md_3, ..., md_n\}$ *and* $\forall md_i \in MD$, md_i *is an MPEG7 media descriptor;*
- *SMS: is a set of sub-media segments, such that* $SMS = \{ms_1, ms_2, ms_3, ..., ms_n\}$ *and* $\forall ms_i \in SMS$, $ms_i =< p_i , t_i , t_i' , MD_i , SMS_i >$. *SMS or* SMS_i *can be* $\emptyset$.

Definition 7 Multimedia Data: *A multimedia data, denoted as mm, is a multimedia data file defined as a 3-tuple,* $mm =< iri, st, MS >$, *where as:*

- *iri: is an identifier to a multimedia data ms given in IRI format;*
- *st: is the location of the multimedia file m, within storage or memory;*
- *MS: is a set of media segments that are contained within the multimedia file m, defined as* $MS = \{ms_1, ms_2, ..., ms_n\}$, $\forall ms_i \in MS$, ms_i *is a media segment (Def. 6).*

In short, for a multimedia sensor reading case, a multimedia data file (produced by a sensor) is modeled by using the `mssn:MultimediaData` concept. This concept models a multimedia file as an object which contains multiple media segments. Each media segment is modeled using the `mssn:MediaSegment` concept. `mssn:MediaSegment` can be attached to multiple `mssn:Media-Desscriptor` to describe its low level features. Media descriptors that we use in our framework adopt the MPEG-7 standard. The `mssn:MediaDescriptor` is a generic concept for modeling low-level features. We categorize low-level features of multimedia data into six categories according to MPEG-7 descriptor groups [42]: (i) motion descriptor; (ii) annotation descriptor; (iii) audio descriptor; (iv) visual descriptor; (v) spatial descriptor; and (vi) temporal descriptor. Several main MPEG-7 predefined descrip-tors are added into *MSSN-Onto*. They are not all given in Figure 4.2 for the sake of clarity in the figure. They are instead given in Table 4.1.

Table 4.1 – List of predefined media descriptors in *MSSN-Onto*

Descriptor Type	Predefined descriptors
AnnotationDescriptor	TextAnnotationDescriptor
AudioDescriptor	FundamentalFrequency, HarmonicDescriptor, PowerDescriptor, SpectrumBasis, SpectrumCentroid, SpectrumEnvelop,SpectrumFlatness, SpectrumProjection, SpectrumSpread, WaveForm, HarmonicSpectralCentroid, HarmonicSpectralDeviation, HarmonicSpectralSpread, HarmonicSpectralVariation, LogAttackTime, SpectralCentroid, TemporalCentroid
MotionDescriptor	CameraMotionDescriptior, MotionActivityDescriptor, WarpingParameters, TrajectoryDescriptor, ParametricMotionDescriptor
SpatialDescriptor	BoundingBoxDescriptor, PointDescriptor
TemporalDescriptor	MediaTimePointDescriptor, MediaDurationDescriptor
VisualDescriptor	ColorLayoutDescriptor, ColorStructureDescriptor, ContourShapeDescriptor, DominantColorDescriptor, EdgeHistogrameDescriptor, FaceRecognitionDescriptor, ScalabaleColorDescriptor

It is noted that it is not obligatory to use all descriptors all the time. One can choose which low-level feature to index depending on the application requirements. Figure4.6 shows the instantiation of *MSSN-Onto* within the motivating scenario previously provided. The example in Figure 4.6 shows the illustration on the usage of the *MSSN-Onto* for modeling a result from a video camera. The video camera is modeled as an instance name `vid1`. The observation of this sensor is modeled using the instance `vid1_obv1`. The timestamp of the observation is recorded and modeled by attaching the timestamp literal with the instance through `resultTime` relation.

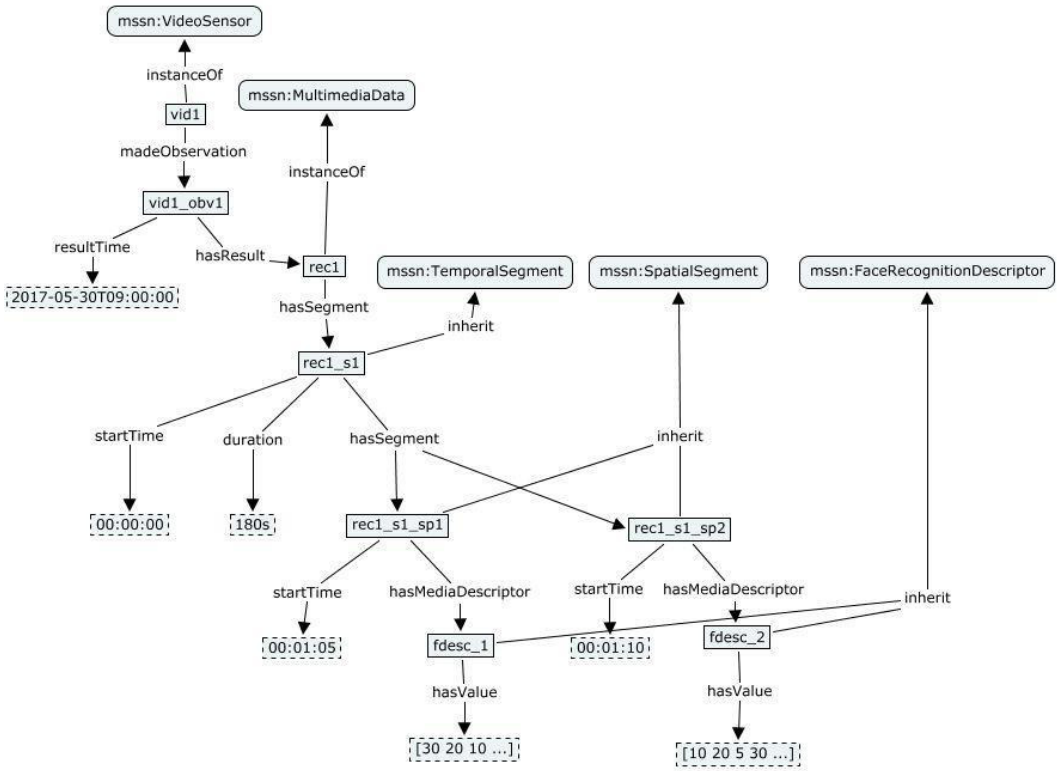


Figure 4.6 – Illustration of multimedia data modeling with *MSSN-Onto*

The actual multimedia data result produced through this observation is modeled by `rec1` instance which is an instance of `mssn:MultimediaData`. The raw multimedia data produced from `vid1` sensor contains only one video track. Hence, `rec1` contains only one segment modeled by the instance `rec1_s1`. Two more instances `rec1_s1_sp1`, `rec1_s1_sp2` are modeled for modeling low level features of the `rec1_s1`. In this case, we model the face recognition descriptor at the time *00:01:05* and *00:01:10*. Instances `fdesc_1` and `fdesc_2` are instantiated and attached to `rec1_s1_sp1` and `rec1_s1_sp2` for modeling the face recognition descriptor value.

4.2.4 Atomic/complex modeling extension

An atomic event is an event that can be detected using one sensor reading a sensor. To detect an atomic event, it is required to evaluate a given sensor reading against a predefined condition. For example, an atomic event `High Temperature` can be detected when a sensor reading from a temperature sensor satisfies the function: `A_reading_value_is_greater_than_25_Celcius`. In *MSSN-Onto*, we

model such a function for detecting an atomic event by using `mssn:AtomicEventCondition`. The formal definition of an `mssn:AtomicEventCondition` is given as follows.

Definition 8 Atomic Event Condition*: An atomic event condition, denoted as ACond is a user-defined function for detecting an atomic event. It is defined as a 4-tuple, ACond =< iri, S, f , P >, where:*

- *iri: is an identifier to an atomic event condition ACond given in IRI format;*
- *S: is a set of sensors that can be used as an input for ACond (see Def. 3);*
- *f : is the IRI to the user-defined function (or a built-in framework fuction) for detecting the atomic event;*
- *P: is a set of additional parameters that the function f requires to operate. P can be ∅.*

A complex event is an aggregation of multiple events connected together as a pattern. For example, an event *temperature in the office is too hot to work* can be modeled as an event that is composed of two simultaneous occurrences of atomic events `Face_Detected` and `High_Temperature` (i.e., a surveillance camera detects at least one person in the room while the temperature within the room is hot). In *MSSN-Onto*, a complex event pattern is modeled using the `mssn:EventStatement` concept. This concept consists of an *event statement* modeled in a plain text format The formal definition of the `mssn:EventStatement` concept is given as follows.

Definition 9 Event Statement: *An event statement, denoted as es, is a statement that describes a complex event pattern from predefined atomic event conditions. It is defined as a 2-tuple, es =< iri, st >, where as iri is the identifier to es given in IRI format, and st is the event statement according to the CEMiD language, given in a plain text format.*

Briefy, an event statement is a plain-text data which describes a complex event by combining multiple atomic event conditions together according to a given event operator. The syntax of the event statement is described later in Chapter 5. An actual occurrence of an event is modeled using `mssn:AtomicEventOccurence`, while an actual occurrence of a complex event is modeled using the `mssn:ComplexEventOccurrence` concept. The formalization of both types of event occurrences can be unified into one single formalization as follows.

Definition 10 Event Occurrence: *An event occurrence, denoted as ec, is a binary tree which a parent node represents an event operator, while left and right siblings are either sub-trees representing other event occurrences or atomic event occurrences (modeled as a tuple between an atomic event condition and an observation), ec =< iri, op, l, r > where:*

- *iri: is an identifier to an event occurrence ec given in IRI format;*
- *op: is a node for representing a complex event operator, op is null is an event occurrence is atomic;*
- *l (and r): is either a sub-tree representing another event occurrence or an atomic event occurrence. Formally, $l =< ACond_i , o_i > | < op_i , l_i , r_i >$), $ACond_i$ is an atomic event condition (see Def. 8), and o_i is an observation value (see Def. 5) which satisfies $ACond_i$ ($< ACond_i , o_i >$ is called an atomic event occurrence).*

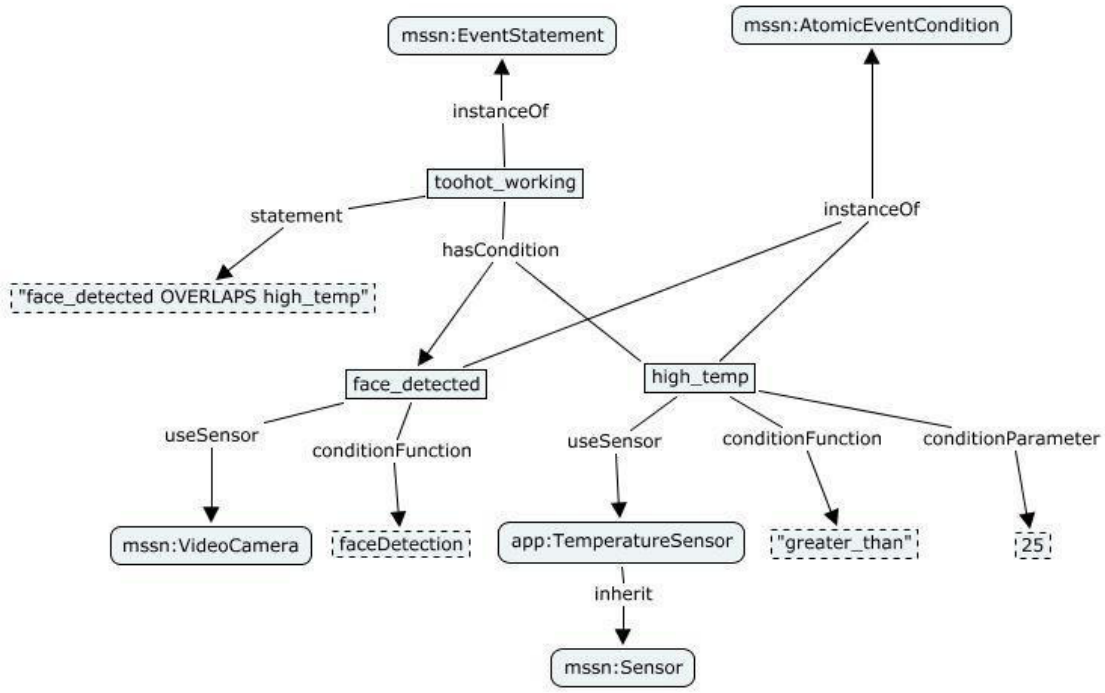


Figure 4.7 – Illustration of an atomic event condition and an event statement modeling

The illustration on how an atomic event condition and an event statement can be defined is shown in Figure 4.7. The figure depicts the instantiation of one event statement `toohot_working` and two atomic event conditions, `face_detected` and `high_temp`. The `face_detected` is used for modeling an atomic event condition for detecting a face within a video. The `high_temp` is used for modeling an atomic event condition for detecting a case which a temperature sensor reads a value that is higher than 25. Detection functions for both of these conditions are modeled by attaching a literal indicating a function name to the corresponding atomic event condition instance with `conditionFunction` relation. Some function may require an additional parameter (e.g., a *greater_than* function requires a decision threshold value). The additional parameter is modeled through the `conditionParameter` function. The illustration on how event occurrences related to atomic event conditions and an event statement of Figure 4.7 can be modeled is depicted in Figure 4.8. Figure 4.8 contains two atomic event occurrences and one complex event occurrence. The atomic event occurrence instances are `face_oc1` and `hightemp_oc1`. They are detected by applying `face_detected` and `high_temp` conditions against sensor reading instance `fdesc_1` and `temp_obv1`. Both of these occurrences satisfy an event statement `toohot_working`. Hence, we create an instance `toohot_oc1` and attach it to both of these atomic event occurrences through `hasOccurrence` relation.

To express a relation between `fdesc_1` and `temp_obv1` according to an event statement `toohot_-working`, the event statement in the plain text format of this instance is `face_detected OVERLAPS high_temp`. Hence, an event operator is instantiated with the event operator name indicated as `OVERLAPS`. The operator connects two occurrences `fdesc_1` and `temp_obv1` together with relations `operator` and `operand`. It is to be noted that we have not explained in details all the syntax and operators for expressing an event statement. This will be described later in Chapter 5 dedicated to the syntax and operators of language for describing complex events.

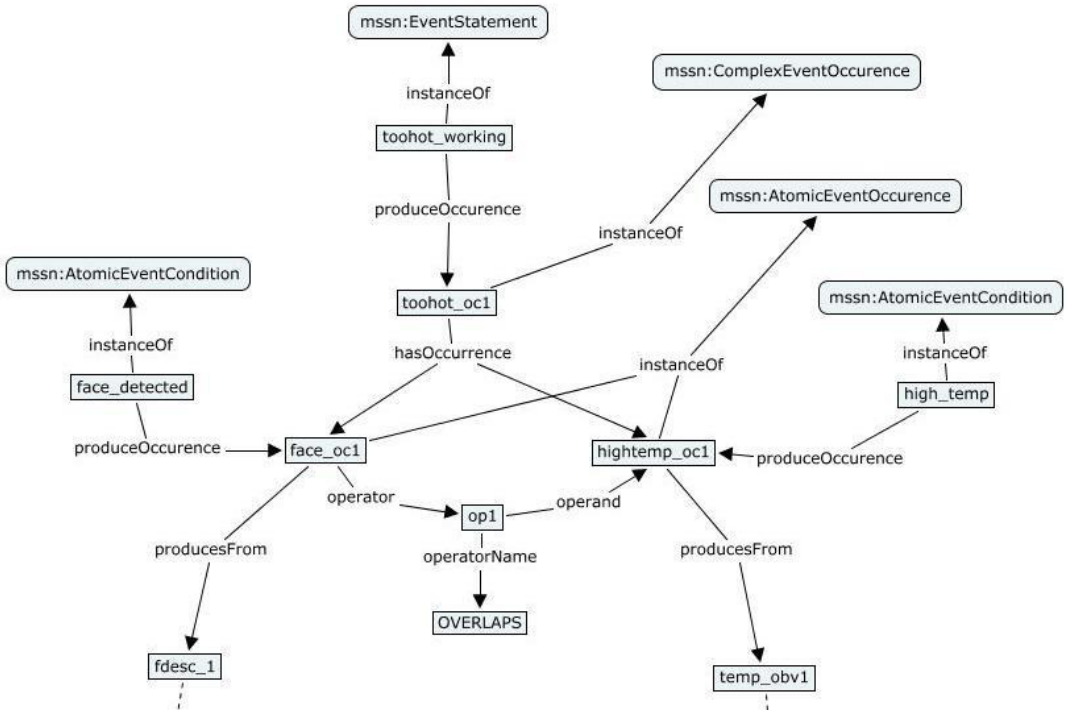


Figure 4.8 – Illustration on event occurrences modeling

4.2.5 Discussion

So far, we described our *MSSN-Onto* through both formal definitions and illustrations. *MSSN-Onto* addresses thoroughly the features are missing from the *SOSA/SSN* ontology for supporting multimedia sensor networks. Hence, we can not only use *MSSN-Onto* as a core ontology of our *CEMiD framework*, but it can also be used for modeling multimedia sensor networks in general. In order to verify and validate the suitability of *MSSN-Onto*, we dedicate Section 4.3 and 4.4 of this chapter for verifying and validating the suitability of *MSSN-Onto* as a standalone ontology without relying on the CEMiD framework.

4.3 Aligning MSSN-Onto with Application Domain Ontologies

In order to use the *MSSN-Onto* without having any process or framework built on top, it is obligatory to align the ontology with an application domain ontology to enable the capacity to define and detect events. This process is needed to be done manually in the case of using the *MSSN-Onto* as a standalone ontology. The manual alignment process consists of four steps as follows:

1. Importing *MSSN-Onto* The first step consists of importing *MSSN-Onto*[5] into an application domain ontology.
2. Aligning the application domain *sensor network* concept with `ssn:System`

   Next, the user should determine the concepts within their application domain ontology that have to be aligned with *MSSN-Onto*. In *MSSN-Onto* (and also in *SSN* ontology), the concept `ssn:System`

[5] Using the following URI: `http://mssn.sigappfr.org/`

is used for modeling a sensor network. This step for aligning *MSSN-Onto* with an application domain ontology is to identify the concepts that can be mapped with the `ssn:System` such that, they can be used for modeling a sensor network concept. Any concept of an application domain ontology used for representing a set, a collection, or an aggregation of sensor devices can be aligned with the `ssn:System` concept. For example, in a smart office ontology, a concept representing a *Room* (or a *Building*) can be aligned with the `ssn:System` since a room (or a building) can have multiple sensor devices installed inside. The alignment between such a concept and the `ssn:System` is to be done using an `inherits` relation such that, the sensor network concept from an application domain can inherit the properties of `ssn:System` concept.

3. Aligning application domain sensors with sensor concepts of the *MSSN-Onto*

   This step of the alignment is a straightforward alignment between concepts used for modeling sensors in the application domain with either `sosa:Sensor` or `mssn:MediaSensor`. In short, a scalar sensor concept should be aligned with the `sosa:Sensor` , while a multimedia sensor concept should be aligned with the `mssn:MediaSensor`. The alignment for both cases can be done using the `inherits` relation in such a way that `sosa:Sensor` or `mssn:MediaSensor` are superclasses and sensor concepts from the application domain are subclasses.

4. Managing `mssn:MediaDataConcerned`

   The last step of the aligning process is needed only if the application needs specific multimedia sensor type representations such as, a smart board sensor or a smart projector sensor. One needs to specifically define the output data type that these application-specific sensors produce. This can be described by adding the relation `has concerned media type` from each sensor concept to their corresponding `mssn:MediaDataConcerned` concepts. For example, in a smart meeting room application, we can model a camera *SmartBoard* (a whiteboard which can record both video and penstroke information) as a concept having `mssn:VideoData` and `mssn:TextualData` as related media data (pen strokes can be represented in a coded text format describing x-y position and color of the pen stroke).

The illustration on how the alignment process can be done is illustrated in the next section along with the experimental evaluation of *MSSN-Onto* and the framework design.

4.4 Validation: Application of the MSSN-Onto in a real-world sce-nario

In order to evaluate the *MSSN-Onto* in different aspects, we conduct three groups of tests:

- Generality Evaluation: the aim of this test is to demonstrate and verify that our *MSSN-Onto* is generic and practical to be used in different application domains;
- Modeling Capacity Evaluation: this test is to evaluate the event modeling and retrieving capacity of the *MSSN-Onto* by means of prototyping;
- Performance Evaluation: this test is to evaluate the retrieval performance of the framework and to measure the overhead of the store space (i.e., size of the data represented with the *MSSN-Onto*).

We have implemented our *MSSN-Onto* in Protége[6]. The *MSSN-Onto* is written in OWL language[7]. Its corresponding description is available online at http://mssn.sigappfr.org. In the following subsections, we detail each evaluation and obtained results.

[6] http://protege.stanford.edu/

[7] https://www.w3.org/TR/owl2-overview/

4.4.1 Generality Evaluation

The objective of the generality evaluation is to validate the capacity of the *MSSN-Onto* on accommodating different application domains through the ontology alignment process. So far, we align our ontology with two different ontologies of two different projects. The details of these projects and how we align the *MSSN-Onto* to their ontologies are given as follows.

4.4.1.1 Aligning the MSSN-Onto with the AMI Smart Meeting Room ontology

AMI Smart Meeting Room is a smart meeting room project which is initated by IDIAP Consortium in 2005 [56]. This project aims to propose a smart meeting room which is capable for summarizing the content of the meeting and detecting events occurred within a meeting room such as `presenting`, `change of a presenter`, `brainstorming`, or `end of the meeting`. To develop such an application, we need to deploy sensors within a meeting room and develop a system that can gather and detect events from the sensor readings. In this case, the IDIAP consortium has proposed to deploy sensors within a smart meeting room prototype. The prototype is named the AMI Smart Meeting Room. The architecture of AMI smart meeting room is given in Figure 4.9.

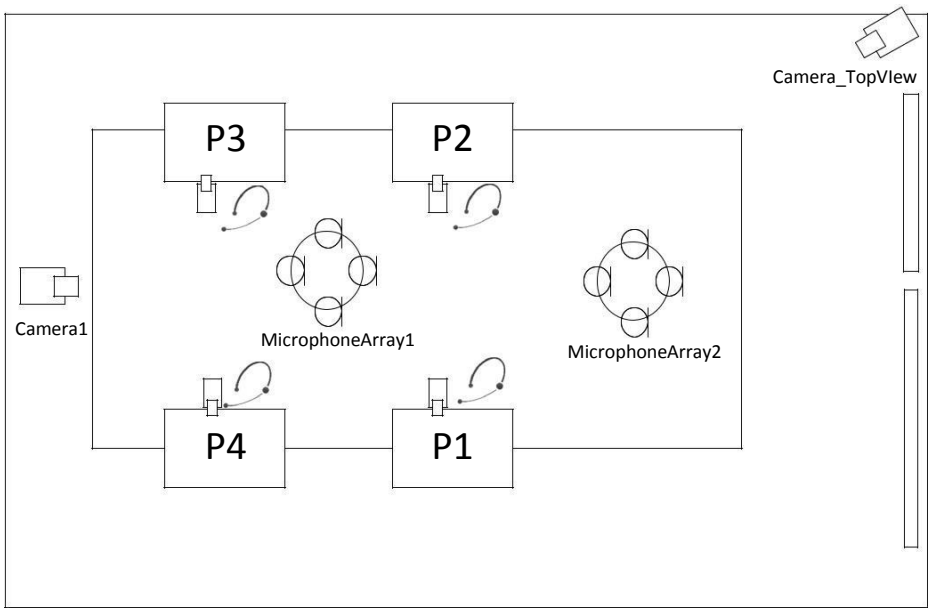


Figure 4.9 – Illustration on event occurrences modeling

In short, there are six types of sensors which are deployed within AMI smart meeting room. They are (i) video camera, (ii) webcam, (iii) microphone array, (iv) headset, (v) smart projector, and (vi) smart whiteboard. Two video cameras are installed to capture the top view angle of the room and the projector screen. Four webcams are installed to capture close up shots of every attendees. Microphone arrays[8] and headsets are used for capturing attendees' speech. One smart projector is installed to capture the projected screen. One smart whiteboard is installed for capturing penstrokes information on the whiteboard. IDIAP

[8] A microphone array is an audio sensor which composes of multiple microphones. It can capture both audio and direction of the originate audio source

consortium has released all the captured data from AMI smart meeting room online as a corpus named *AMI Corpus*. The corpus can be found online [9].

We choose to use AMI smart meeting room as one of the selected scenario for the validation process because the availibility of the AMI Corpus. There is no need for us to redeploy a sensor network for a smart meeting room from scratch as all the data is already available within the corpus. However, in order to use our *MSSN-Onto* with AMI Corpus. First, we need to propose an ontology designated to the AMI Corpus as it is not existed. Next, we need to align *MSSN-Onto* with the AMI ontology.

Towards the development of such system, we need to model the multimedia sensor network infrastruc-ture and the events that can be detected in this scenario. For this purpose, we formulated the AMI smart meeting room ontology, as given in Figure 4.10.

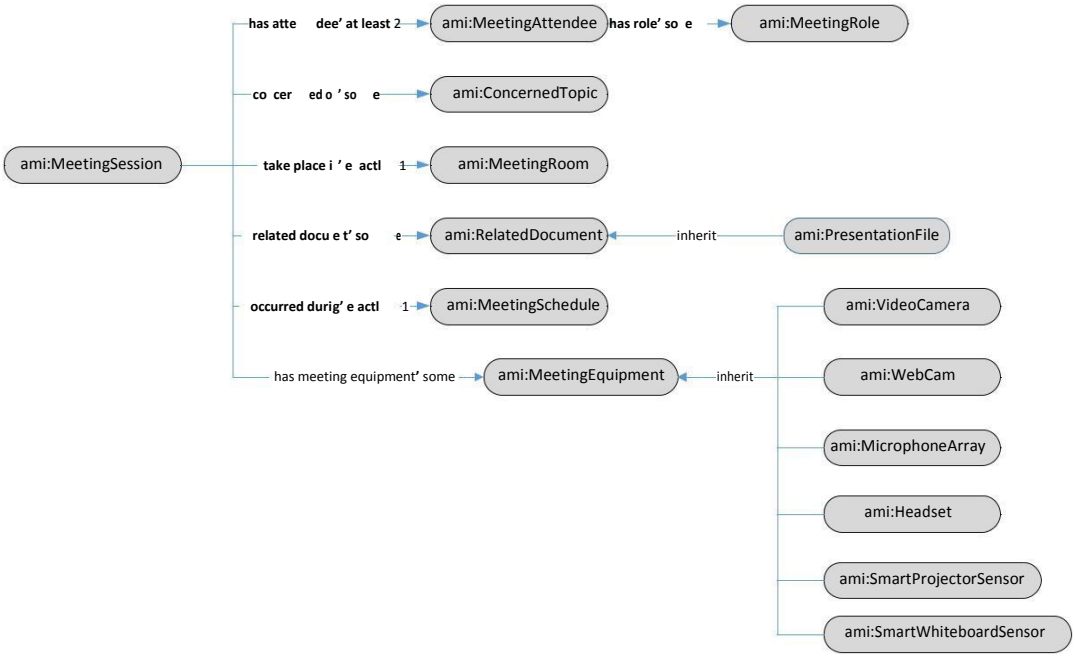


Figure 4.10 – The AMI Smart Meeting Room Ontology

Next, we aligned *MSSN-Onto* with the AMI smart meeting room ontology, by following the process as given in Section 4.3. We elaborate the alignment process for the AMI ontology case in a step by step manner as follows.

1. Aligning the AMI sensor network concept with `ssn:System`

   In the AMI ontology, the `ami:MeetingSession` is a concept that is used for representing a single session of a meeting. We deem that this concept can be used for representing a sensor network because there are several pieces of equipment (i.e., sensors) that are used within a meeting session. Hence, we aligned the `ami:MeetingSession` with `ssn:System`.

2. Aligning AMI `sensors concepts` with sensor concepts of the *MSSN-Onto*

   All meeting equipment concepts (i.e., `a`mi:MeetingEquipment and its sub-concepts) can be all considered as multimedia sensors. Hence, we aligned them with `mssn:MediaSensor`.

[9] http://groups.inf.ed.ac.uk/ami/corpus/

3. Managing `mssn:MediaDataConcerned`

   Among meeting equipment concepts in the AMI ontology, `ami:SmartProjectorSensor` and `ami:SmartWhiteboardSensor` are not existed within a set of predefined media sensor types (see Figure 4.2). Hence, we need to provide an information on the types of multimedia data that they produce. For example, a smart whiteboard sensor, `ami:SmartWhiteboardSensor`, produces video footages, images and pen stroke information as an output as an XML string. Thus, we aligned this concept with `mssn:VideoData`, `mssn:ImageData` and `mssn:TextualData` as their concerned media data type.

The result of the alignment can be seen in Figure 4.11.

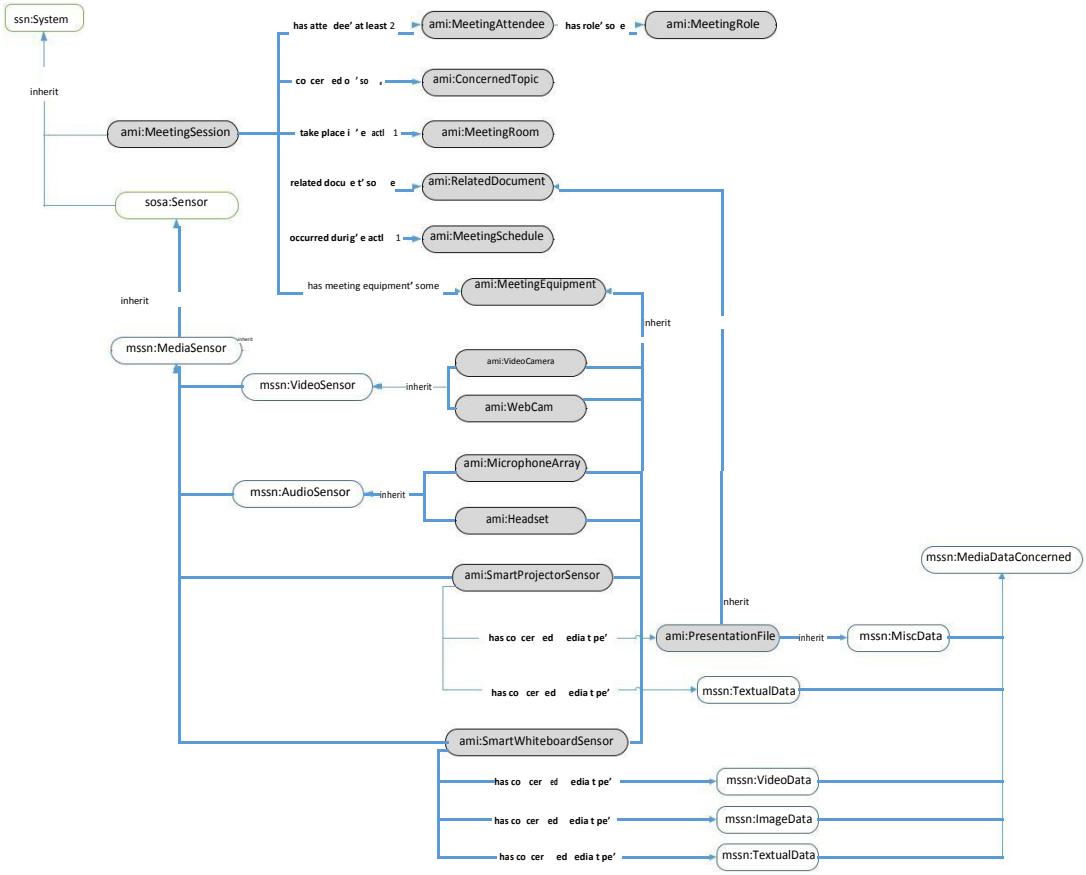


Figure 4.11 – The alignment between the AMI Smart Meeting Room Ontology and the *MSSN-Onto*

4.4.1.2 Aligning MSSN-Onto with the HIT2GAP ontology

The *HIT2GAP*[10] is an EU joint collaboration research project (EU/H2020 Grant Agreement No:680708) for developing a next generation building control tool for optimizing an energy usage. The main objective of this project is to propose a new paradigm of an energy management platform for a smart building. The

[10] http://www.hit2gap.eu

project members consist of 22 partners from 10 European countries. The *HIT2GAP* platform is an ontology based platform which allows different partners to query for data and events from a smart building data. The architecture of the *HIT2GAP* platform is given in Figure 4.12.

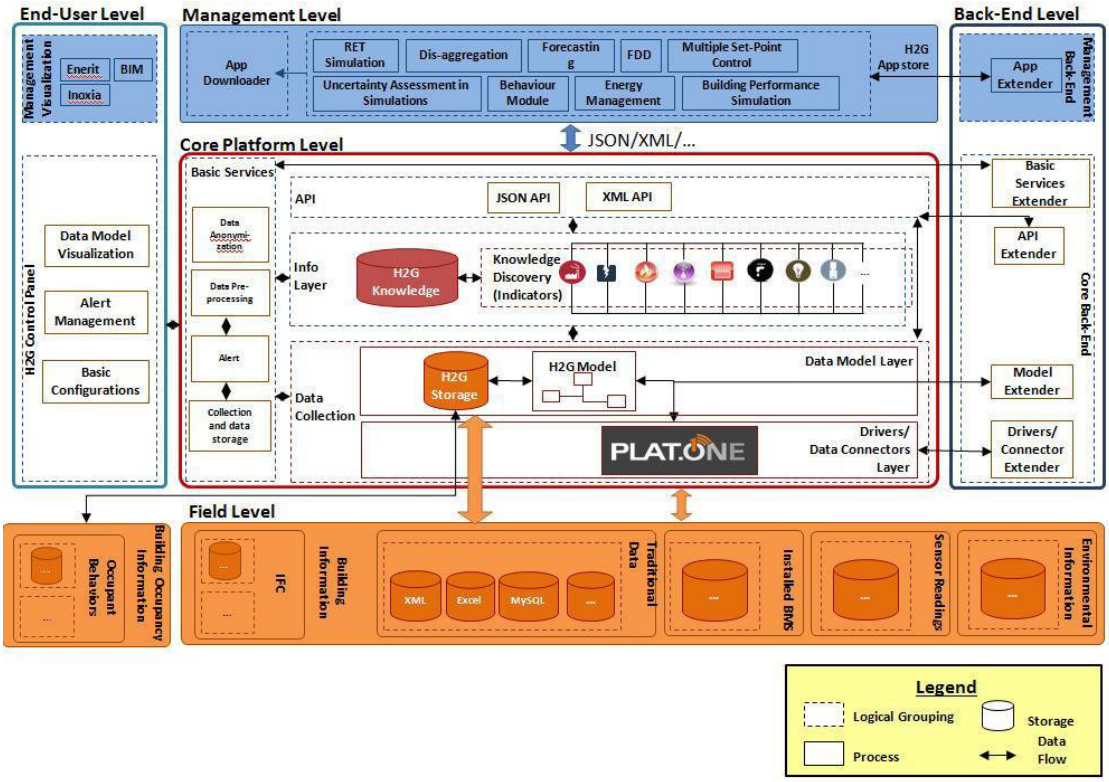


Figure 4.12 – The architecture of the *HIT2GAP* platform

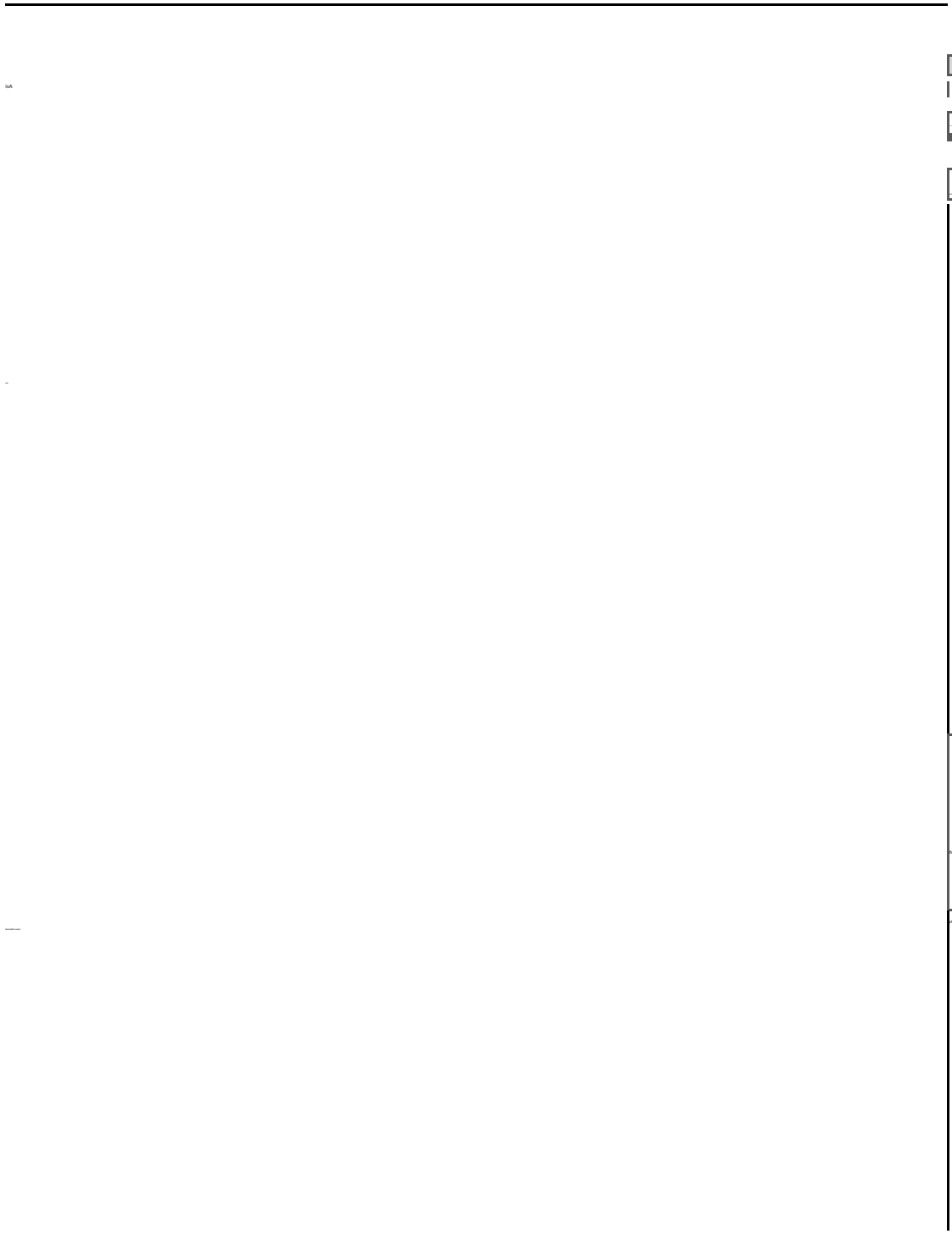
The Data Model Layer in Figure 4.12 is used for modeling and storing data within the platform. The data model is called the *H2G Model*. This data model is based on an ontology so called the *HIT2GAP* ontology developed by aligning ontologies provided by the project partners into a single ontology. Our *MSSN-Onto* is among one of the ontologies that are being aligned to enable the platform to model sensor readings that are gathered from multimedia sensors within a building (e.g., surveillance camera). The alignment can be seen in Figure 4.13

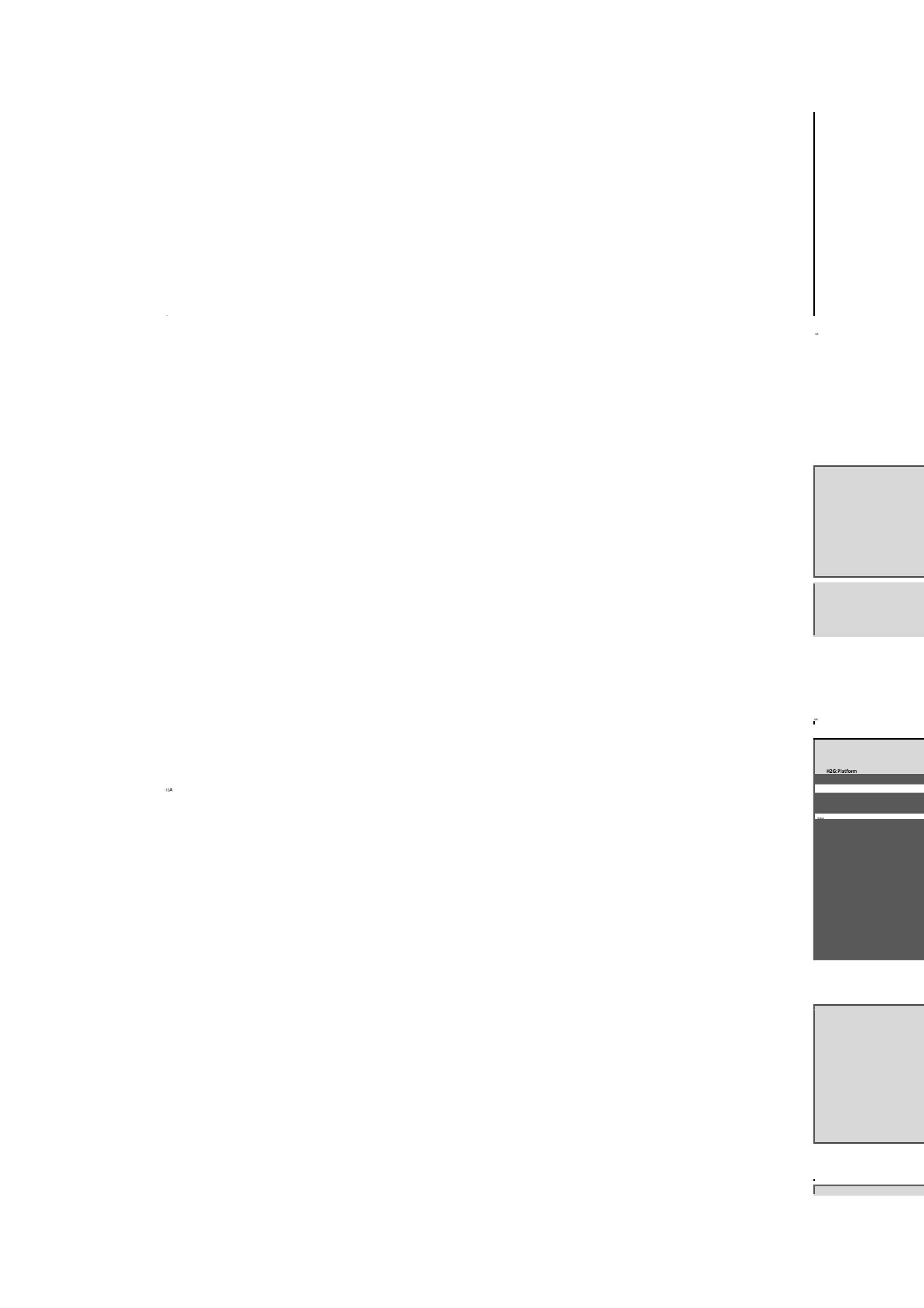
Not all the concepts within the *HIT2GAP* ontology are given in Figure 4.13 for the sake of brevity. It is to be noted that the concepts that are prefixed with `h2g:` are newly proposed in the *HIT2GAP* ontology. Concepts that are not prefixed are taken from other partners. To explain briefly, we followed the three steps alignment as given in Section 4.3. The details are given as follows.

1. Aligning the *HIT2GAP* sensor network concept with `ssn:System`

   In the *HIT2GAP* platform, there is a concept called `IfcRelAggregates` that is taken from a smart building ontology[11]. It is used for representing an aggregation of devices (i.e., a system that con-sists of multiple devices). Hence, we aligned the `ssn:System` with the `ifcRelAggregates` concept.

[11] http://openbimstandards.org/standards/ifcowl/


IfcObjectDefinition
IfcProduct


H2G:Platform

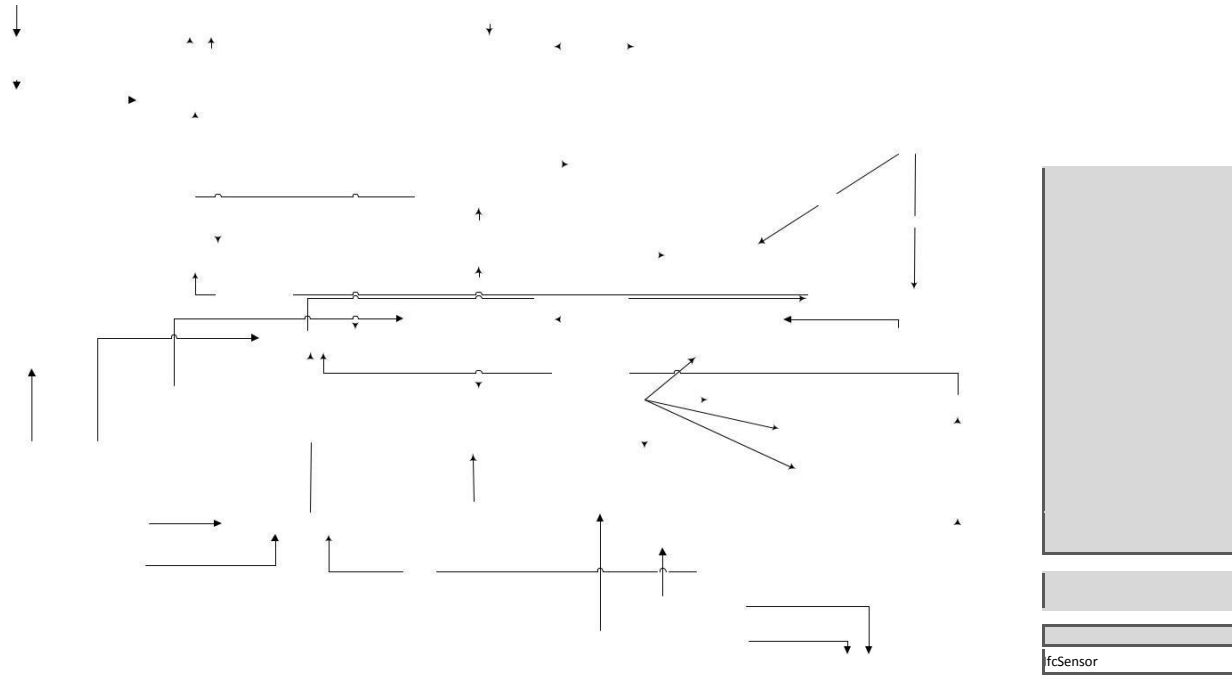


Figure 4.13 – The alignment of *MSSN-Onto* with the *HIT2GAP* ontology

2. Aligning *HIT2GAP* sensor concepts with sensor concepts of *MSSN-Onto*

   The only concept that is used for representing a sensor within the *HIT2GAP* ontology is the `ifcSensor`. We aligned it with the `sosa:Sensor` and `mssn:MediaSensor`. The *HIT2GAP* ontology aims to be a generic ontology for a smart building. Hence, unlike the AMI smart meet-ing room case, there is no specific type of sensors that is modeled within the ontology. Hence, we did not have to align any further concept within the ontology with either `sosa:Sensor` or `mssn:MediaSensor`. Such an additional alignment might be needed later in a practical imple-mentation when we have a specific list of sensors from a building that the *HIT2GAP* platform will be deployed.

3. Managing `mssn:MediaDataConcerned`

   No alignment has been done in this step (due to the lack of media sensors).

The *HIT2GAP* project is currently on-going. Hence, we have not had any experiment result in a real implementation scenario yet. The result is expected to be available by the end of 2019.

4.4.1.3 Remark

In this section, we aimed to show how we evaluated the generality of *MSSN-Onto* by means of practically aligning *MSSN-Onto* with different ontologies from different application domains. This method can be subjective and time-consuming as the alignment is done manually so far. Nevertheless, it is important to note that the alignment of MSSN-Onto to the *AMI* ontology and the Hit2Gap ontology was done easily

and quickly without any difficulty. Of course, an efficient method for assisting the aligning of the ontology (either an automatic or a semi-automatic alignment) will be beneficial and proposed in our future study.

4.4.2 Modeling Capacity Evaluation

The objective of the modeling capacity tests was to evaluate the expressiveness of *MSSN-Onto* and our framework in defining and retrieving complex events. To do so, we implemented the event detection and retrieval function for the AMI Smart Meeting application by using our ontology and framework. Due to the lack of the smart meeting room infrastructure, we chosed to develop our prototype by using multimedia data from a prerecorded AMI corpus. The details of the corpus, our prototype, and experiments are given as follows.

4.4.2.1 Introduction to the AMI Corpus

The AMI corpus is a multimedia data corpus that is recorded from the *AMI Smart Meeting Room* application. The complete AMI corpus[12] contains 100 hours of meeting recordings. The AMI corpus data is separated into multiple directories, where each directory contains data from one meeting session. The data within each category is furthered organized into multiple directories according to sensor types. They are described in Table 4.2.

Table 4.2 – AMI Corpus Directory Structure for a Meeting Session

Directory Name	Related Sensor(s)	Data Type(s) and Encoding Format(s)
video	- Video Cameras - Webcams	- Video (MPEG4/AVI)
audio	- Headset - Microphone Array	- Audio (WAV)
slides	- Smart Projector	- Image (JPG) - Text (Text File) - Timestamp (Text File)
pens	- Smart Whiteboard	- Video (MPEG4/AVI) - Image (JPG) - Penstroke (XML)

4.4.2.2 Prototype Architecture

In order to exploit our approach in the AMI corpus, we modeled the AMI smart meeting room as an application domain ontology and aligned it with *MSSN-Onto*. This is already described in the previous section and given in Figure 4.11. We developed a prototype for processing the AMI Corpus according to the aligned ontology and *MSSN-Onto* based framework. The prototype can be found online[13]. The architecture of our prototype can be seen in Figure 4.14.

Briefly, the MSSN-Indexer module takes the sensor data from the AMI corpus directly. It uses GStreamer[14], OpenCV[15], and Essentia[16] libraries for decoding and extracting low-level multimedia

[12] http://groups.inf.ed.ac.uk/ami/download/
[13] http://mssn.sigappfr.org/demo
[14] https://gstreamer.freedesktop.org/
[15] http://opencv.org/
[16] http://essentia.upf.edu/

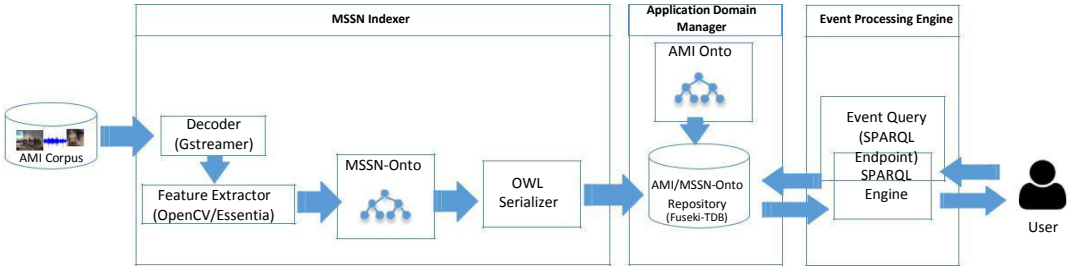


Figure 4.14 – The architecture of our prototype

features. The low-level features are indexed and stored within the repository kept by the Application Domain Manager. The AMI ontology aligned with *MSSN-Onto* (see Figure 4.11) is also stored in this repository. We used Apache Jena Fuseki TDB[17] to implement this repository, since it is well recognized for its performance and scalability. For detecting events, the user submits the query in SPARQL language, the SPARQL engine queries the Fuseki TDB to identify the events and then communicates the result to the user.

It is to be noted that we cannot use a traditional relational database for implementing the repository. The usage of the ontology based approach as we propose imply that all the data are modeled by using RDF[18]. Unlike a traditional database, the RDF stores data as a set of triples:h`subject`, `predicate`, `object`i, where `subject` and `object` are instances generated from concepts within an ontology, and `predicate` is a relation between `subject` and `object` as defined within an ontology. In this case, it is obligatory to use a triplestore software (e.g., Fuseki) for implementing the repository.

We selected ten events from the AMI scenario and defined them using the concerned descriptors according to *MSSN-Onto*, as shown in Table 4.3. They were selected because they represent complex events resulting from the aggregation of multiple sensor devices and multiple sensor readings to be detected. Additionally, these events reflect the typical flow of events that can happen from the start to the end of a meeting session. The details of each query can be found on the website of the prototype. It is to note that the online version does not allow users to write new queries but only execute predefined ones.

4.4.2.3 Evaluation Measures

To validate the modeling capacity of our *MSSN-Onto*, we used the following *Detection Accuracy* metrics:

- Precision is a metric between 0 and 1 used for measuring the relevancy of the results (detected events). The higher the precision is, the more relevant the results are. It is calculated as the number of True Positives (TP)[19] over the number of True Positives (TP) and False Positives (FP)[20] (see Eq. 4.1).

$$Precision = \frac{TP}{TP + FP} \quad (4.1)$$

- Recall is a metric between 0 and 1 used for measuring the correct relevant results. It is calculated as the number of True Positives (TP) over the number of True Positives (TP) and False Negatives (FN)

[17] https://jena.apache.org/documentation/fuseki2/
[18] https://www.w3.org/RDF/
[19] True positives, in event detection, is the number of events which the system can correctly detect.
[20] False positives, in event detection, is the number of events which the system is falsely detected.

Table 4.3 – List of events and their description

Event Name	Event Description	Concerned Sensor	Concerned Media Descriptor
Beginning of a meeting session	The beginning of a meeting where a slide is loaded on the screen and a participant starts speaking	Smart Projector, Microphone Array	mssn:TextAnnotationDescriptor, mssn:MediaTimePointDescriptor
Presenting the meeting's agenda	A participant introduces the meeting's agenda	Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor
Presenting a topic	A participant presents his/her work	Microphone Array, Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor
Using the active board	A participant uses the active board to illustrate some ideas while presenting his/her work	Whiteboard, Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor mssn:PointDescriptor mssn:ColorDescriptor
Changing slides	A point in time where a slide is changed	Smart Projector	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor
Changing to another presentation	Changing to a new presentation	Microphone Array, Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor
Brainstorming	Several participants discuss with each other on a topic while remaining on the same slide	Smart Projector, Microphone Array, Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor
Participant leaving their seats	A point in time where a participant leaves his/her seat	Webcam	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor mssn:FaceRecognitionDescriptor
Summarizing the meeting	A participant summarizes all the points discussed within the meeting	Microphone Array, Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor mssn:TextAnnotationDescriptor
Leaving the room	All the participants leave the room	Microphone Array, Headset	mssn:MediaTimePointDescriptor mssn:MediaDurationDescriptor

21(see Eq. 4.2). The higher the Recall is, the more completeness of the results are.

$$Precision = \frac{TP}{TP + FN} \tag{4.2}$$

- F-Measure is a harmonic mean between 0 and 1 based on the precision and recall. The higher the F-Measure is, the more accurate the detection is. It reflects the overall quality of the results. It is computed as follows:

$$F\text{-}Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.3}$$

4.4.2.4 Results

We conducted our experiments by randomly picking up four meeting sessions from the AMI corpus and indexing them offline. The querying processing is done online through the web interface of our prototype. To determine the ground truth of TP, FP, and FN , we first detected events through human observation. Then, we compared the result of each query with the human result. The number of occurrences of *Presenting a topic* and *Brainstorming* events is counted according to the time window during which those events were actually occurring. The other events (e.g., *The meeting is started, The slide is changed, The presenter has changed*) have occurred only at a certain point in time. Hence, the occurrences have been counted according to the number of times each event is detected within the data.

[21] False Negatives, in event detection, is the number of events which the system supposes to detect but fails to detect.

Table 4.4 – Event Detection Accuracy Result

Event	TP	FP	FN	Precision	Recall	F-Measure
Beginning of a meeting session	2	2	2	0.50	0.50	0.50
Presenting the meeting agenda	3	1	1	0.75	0.75	0.75
Presenting a topic	1312	538	699	0.71	0.65	0.68
Using the active board	11	3	1	0.79	0.92	0.85
Changing slides	17	0	0	1.00	1.00	1.00
Changing to another presentation	1	2	6	0.33	0.14	0.20
Brainstorming	1868	214	1260	0.90	0.60	0.72
Participants leaving their seats	1	12	17	0.08	0.06	0.07
Summarizing the meeting	2	2	2	0.50	0.50	0.50
Leaving the room	4	0	0	1.00	1.00	1.00

The results presented in Table 4.4 show that our *MSSN-Onto* based prototype can detect events from AMI corpus data with moderate accuracy despite using untune commercial on-the-shelf multimedia processing functions. Significant results are in the detection accuracy of *Brainstorming* and *Using the active board* events. They are both complex events for whose detection requires sensing from several sensors. The results show a *F-Measure*=0.72 for *Brainstorming* and *F-Measure*=0.85 for *Using the active board* events, which means highly accurate detection. *Changing slides* and *Leaving the room* events have perfect detection accuracy. This is because the smart projector sensor helps provide accurately when the slide is changed. We can also combine this information with the audio sensor information to accurately detect when all the participants leave the room. The *Participants leaving their seats* event has the lowest detection result, due to the usage (and not tuning) of the on-the-shelf visual feature recognition method of OpenCV.

This is also the case of *Beginning of Meeting Session*, *Change to Another Presentation*, and *Summarizing the Meeting* events, which are underperformed due to the default parameters used in the adopted voice activity detection and phrase segmentation methods of OpenCV. Note that the underperformed results are not caused by *MSSN-Onto* since it is related to the fact that we did not tune the multimedia feature extraction and recognition functions implemented in our prototype. Of course, this can be improved by tuning properly the existing methods or adopting others.

4.4.2.5 Discussion

In this section, we showed how we evaluated and demonstrated the event modeling capacity of our approach (the *MSSN-Onto* and the framework). The result shows that we can develop a system for defining and detecting complex events from AMI corpus successfully with an acceptable accuracy. We will expand our evaluation into more application domain in our future work.

It is noted that the development of the complete framework is still our on-going work. Hence, some parts of the prototype that are used for the experiment are hard-coded to the AMI smart meeting room application.

It is also noted that the event detection accuracy result as we show may not be directly relevant to the modeling capacity evaluation. However, the accuracy result allow us to demonstrate the capability of our ontology and framework on overcoming the syntactic and semantic interoperability requirements in MSNs. By adopting our approach, we can interoperate information within MSNs better. Hence, it allow us to detect events with a moderate accuracy despite the fact that we use an inaccurate multimedia feature extraction and recognition functions when conducting the experiment.

4.4.3 Retrieval Performance Evaluation

The framework that we proposed for processing complex events is an ontological based framework. Such a choice imposes questions and challenges on the performance issue whether it is capable to be used in real-time or near real-time processing or not due to its considerably high data overhead nature [57]. Hence, we conducted experiments to verify the feasibility of our approach for the completeness of our study.

To evaluate the retrieval performance when detecting complex events by using our framework, we used two data sets:

- AMI corpus: to measure the data access time on a real dataset; and
- Simulated corpus: to measure the overhead of the store space (i.e., size of the data represented with *MSSN-Onto*) and the performance of the query process in a high workload (large amount of sensors and multimedia data) scenario.

We detail each test below in following sub-sections. It is to be noted that every query that we ran within the experiments were executed after all the data were properly indexed using our ontology. The queries were not executed during the run-time (i.e., continuously executing each query against an incoming sensor data stream) due to the fact that we are using a prerecorded data corpus. This will be done in our future study.

4.4.3.1 Evaluation in AMI Corpus

To evaluate the retrieval performance when detecting complex events, we measured the time that our online prototype engine[22] takes to access the data from the Fuseki TDB repository (in which sensor and multimedia data descriptions are stored) without considering the parsing time (i.e., the time it takes to parse the *MSSN-Onto* data repository into the memory). To avoid the overhead on the HTTP transmission of our web-based prototype, we created a standalone version to conduct this performance experiment offline. All the components of the online prototype as previously described are kept the same in the standalone version as described in Figure 4.14. The machine that we used to conduct the test has Linux Mint 17 operating system, Intel Core 7 Quad Core 2.4 GHz CPU and 8 GBytes of RAM, on which we also installed the Fuseki repository and the SPARQL engine EasyRDF[23]. The overall size of the ABox (number of triples) is 163,531 triples. The ABox size of each query is given in Table 4.5. We performed each query five times and reported in milliseconds its average time over all the executions. It is noted that the total ABox size is not equal to the sum of ABox size for each query because these queries do not represent the whole information or events that can be found. The total ABox size contains our selected events, events that we do not select, sensor infrastructure information, and application domain information.

The experiment results, presented in Figure 4.15, show that the query that takes the longest time is the one related to *Using the active board* event, which is 1,447 ms. In fact, this query is the most complicated one to be executed, as it requires detecting a sequence related to a point in time where a presenter grabs a pen right before a *Presenting* event. We also observed that three complex events can be identified faster

[22] http://mssn.sigappfr.org/demo
[23] http://www.easyrdf.org/

Table 4.5 – ABox Size for each query in AMI Corpus experiment

Event	ABox Size	Total ABox Size
Beginning of a meeting session	4	163,531
Presenting the meeting agenda	4	
Presenting a topic	10	
Using the active board	14	
Changing Slides	47	
Changing to another presentation	10	
Participants leaving their seats	28	
Brainstorming	13	
Summarizing the meeting	4	
Leaving the room	4	

than the others: *Beginning of a meeting session*, *Leaving the room*, and *Change to another presentation* events, taking approximately 0.04 - 0.05 ms. This is related to the fact that their detection is easier since they require data only from the same type of sensors (in this case, multiple audio sensor readings).

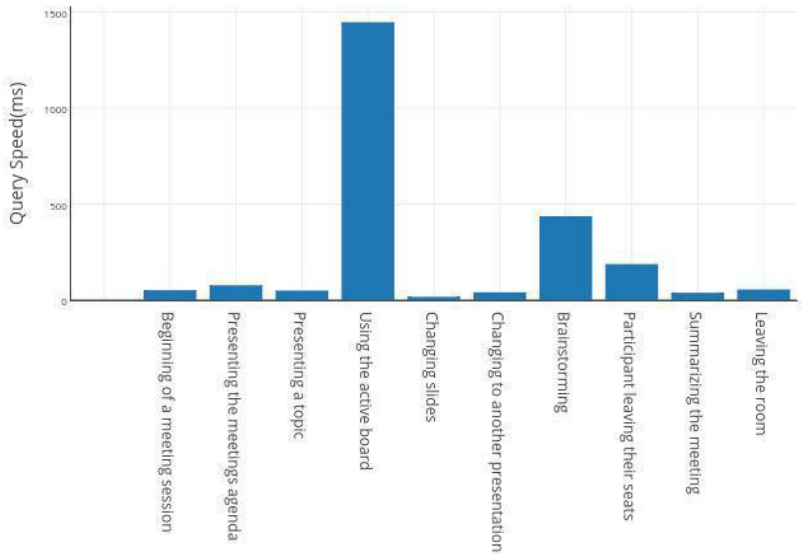


Figure 4.15 – Event Querying Performance Result

4.4.3.2 Evaluation using simulated data

In order to evaluate the performance on a bigger network, we performed other sets, by means of simulations, with high workloads (i.e., MSNs that contain large amount of sensors and multimedia data). To do so, we simulated MSNs with video camera sensors (considered as the most expensive ones in terms of processing and storage). We varied the number of sensors from 10 to 500 and the number of collected data, and more

precisely media descriptors, from 60 to 3600. All camera sensors in the MSNs are supposed to capture video data at the same time (in order to take the worst case scenario). The length of the video that each sensor captures is one hour. We generated the `mssn:MultimediaData`, `mssn:MediaSegment`, and `mssn:MediaDescriptor` directly by using mock up video data. The mock up video data contains programmatically generated video frames. Every generated video data is decomposed and indexed with respect to the temporal information. Thus, the `mssn:TemporalDecompositionSegment` is used when decomposing the video data into multiple segments. Each segment contains five media descriptors (time point descriptor, duration descriptor, text annotation descriptor, color structure descriptor, and face recognition descriptor), where each descriptor has always a fixed size. The simulation varies the number of total media descriptors by varying the duration of media segments. The simulation steps are given below.

1. Simulate 10 video camera sensors, each one producing a video data of 1 hour long;
2. Index and map every produced video with *MSSN-Onto*, and decompose the video contents into multi-ple temporal segments of 300 seconds long each, all of them also represented with *MSSN-Onto*. Each segment is attached to the 5 aforementioned media descriptors. This produces $3600_{sec}/300_{sec}$ = 12 media segments, which in turn produces 12_x5 = 60 media descriptors per each one hour video;
3. Measure the size of the file that is serialized after indexing the simulated sensors with *MSSN-Onto* and record the result in megabytes unit;
4. Import the serialized file into Fuseki triple-store;
5. Launch a SPARQL query for *Retrieving a media segment that contains a video frame at the 30 minutes position of the recorded video* to the Fuseki TDB repository. Record the time that the query needs to identify such event, in millisecond unit;
6. Redo the steps 2 to 4 by varying the segment size from 300 seconds to 60, 40, 20, and 5 seconds. This produces 60, 90, 180, and 720 media segments which also means 300, 450, 900, and 3600 media descriptors per one hour video, respectively;
7. Redo the steps 1 to 5 by varying the number of simulated sensors from 10 to 50, 100, 200, and 500 sensors.

Overhead Size Evaluation: The size of the generated *MSSN-Onto* data (i.e., sensors and multimedia data descriptions) from each simulation is given in Table 4.6.

Table 4.6 – Overhead size (in MBytes) according to the number of sensors and number of media descriptors

Number of Sensors	Number of Media Descriptors				
	60	300	450	900	3600
10	0.26	1.4	1.68	3.34	13.26
50	1.13	6.21	9.23	18.27	72.53
100	2.82	9.23	18.66	36.94	146.64
200	5.69	18.27	37.64	74.5	295.66
500	14.29	72.54	94.55	187.15	742.75

Note that as expected, the overhead size of the *MSSN-Onto* data depends on the number of media descriptors that are used. This overhead is huge when the division of the media segment are as small as 5 seconds per slice, in which case 3600 media descriptors per video are produced. In Figure 4.16, the data reported in Table 4.6 is also shown, to illustrate the effect of varying the number of sensors on the

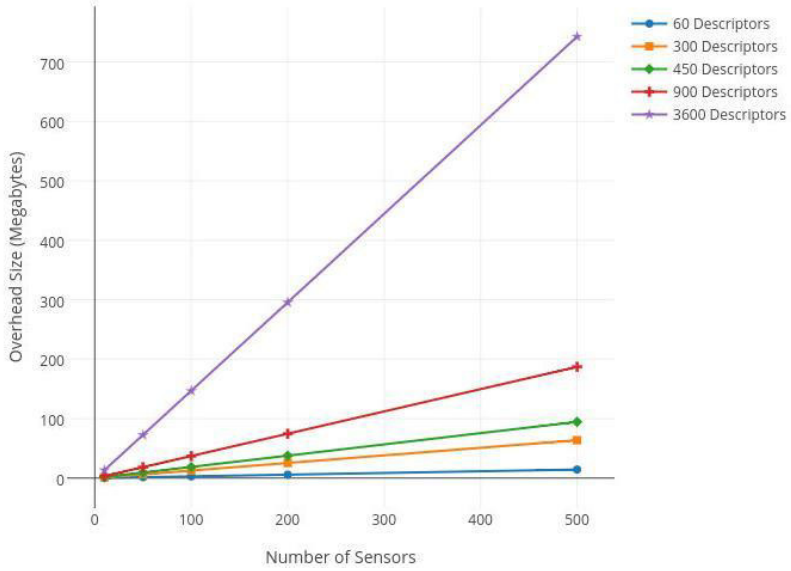


Figure 4.16 – Overhead size of *MSSN-Onto* from the simulation

overhead size. These results show that the relation between these two variables tends to be linear, which reflects a slow overhead size growth rate.

Stress Performance Evaluation: The metric used for measuring the query performance is also the query execution speed. We ran a query to look for one triple within all the triples that are recorded. The number of triples (ABox size) for each experiment is given in Table 4.7.

Table 4.7 – ABox Size for each experiment according to number of sensors and numbers of media descriptors

Number of Sensors	Number of Media Descriptors				
	60	300	450	900	3600
10	2,116	6,460	14,050	27,820	110,440
50	11,516	51,500	76,490	151,460	601,280
100	23,266	104,050	154,540	306,010	1,214,830
200	46,766	209,150	310,640	615,110	2,441,930
500	117,226	524,450	778,940	1,542,410	6,123,230

We plot the experiment results in an XY chart where X-Axis gives the number of sensors and Y-Axis gives the query execution speed in millisecond unit. The result can be seen in Figure 4.17.

The relation between the number of sensors and query execution speed, according to Figure 4.17, tends to be linear. The longest query speed is of 31.66 seconds from the simulation of 500 sensors with 5 seconds media segment slice size. This ought to be acceptable as the overhead size of this case is 742.75 megabytes (6,123,230 triples). The query speed of the other case is ranging between less than 1 second to 12 seconds. This suggests that our *MSSN-Onto* is usable in a near-real time event detection case where the event detection is delayed for up to 1-2 minutes.

To sum up, we can confirm, thanks to these results, that *MSSN-Onto* can effectively model MSNs and related multimedia data in order to detect complex events. The event detection queries return positive results in most cases, except for the cases in which some multimedia pre-processing and tuning is required.

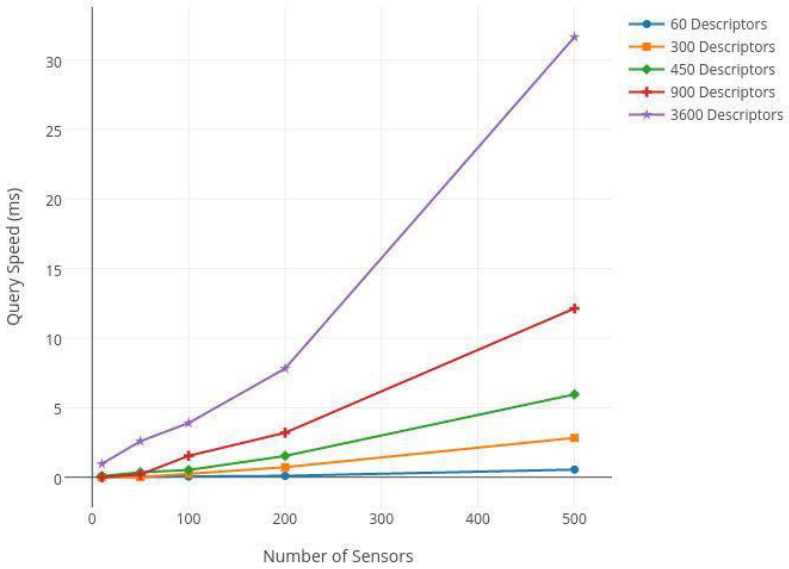


Figure 4.17 – Query execution speed results

Additionally, with this work we demonstrated the possibility of building an efficient query engine for complex event detection, based on *MSSN-Onto* representation even in scenarios with huge amount of data and big number of multimedia sensors.

4.5 Conclusion

In this chapter, we introduce an ontology for modeling multimedia sensor networks so-called the *MSSN-Onto*. The *MSSN-Onto* helps to (i) fully model multimedia sensor networks; (ii) provide syntactic interoperability; and (iii) provide semantic interoperability among all the data gathered within multimedia sensor networks. The *MSSN-Onto* is a generic ontology for multimedia sensor networks. Hence, we can use it in any application domain. We also conducted experiments and validations to confirm the suitability of the *MSSN-Onto*. The result suggests that *MSSN-Onto* is suitable to be used as a generic ontology for modeling multimedia sensor networks.

The *MSSN-Onto* is used by the Repository module of our framework. The *Ontology Handler* module helps to translate users' requests and all gathered sensor readings into the *MSSN-Onto* model. Users' requests are written in *CEMiD* language. The language is described in Chapter 5. The internal mechanism on how the framework used the *MSSN-Onto* is given later in Chapter 6.

Chapter 5

Complex Event Modeling and Detection (CEMiD) Language

Recall from Figure 3.1, *CEMiD Interpreter* is one of the core components of the framework. This component helps users to model sensor network infrastructure, modeling events that users would like to detect, and indicate actions and reports that the system needs to conduct when a certain event is detected. Users can model all the models by using *CEMiD* language, a high-level language what we propose for users to interact with the framework. The *CEMiD* language can also be serialized in JSON format in the case that users or developers prefer a JSON-based syntax for using in a web service-like environment.

The syntax style of *CEMiD* language is based on SQL/SPARL and ECA languages. This chapter begins with the description of the language syntax for each of the language functionalities. This follows by a dedicated section for describing a serialization of the *CEMiD* language in JSON-based syntax. The section after describes how the *CEMiD* language is processed by the framework. This chapter ends with the validation of the capacity of the language.

5.1 CEMiD Language

Functionalities of the *CEMiD* languages are: (i) Multimedia Sensor Network Infrastructure Modeling, (ii) Event Modeling, (iii) Action Definition, and (iv) Historical Data Querying. Syntaxes for each functionality are described in sub-sections as follows.

5.1.1 Multimedia Sensor Network Infrastructure Modeling

Recall from Chapter 4, data which has to be modeled for modeling multimedia sensor network infrastructure are a location map, and sensor infrastructure. Syntaxes for modeling them are given as follows.

5.1.1.1 Location Map Modeling

CEMiD language syntax for modeling a location map is based on the SQL/SPARQL style languages. All the statements for modeling a location map are proposed in response to Def. 1 and 2 in Chapter 4. The list of CEMiD commands for modeling a location map is given as follows.

Map creation/deletion

```
CREATE MAP <map_IRI>;
DELETE MAP <map_IRI>;
```

These statements are used for creating an empty location map or deleting an already defined map.

Location insertion/deletion

```
INSERT LOCATION <location_IRI> TO <map_IRI> [VALUE
<pos_value>]; DELETE LOCATION <location_IRI> FROM <map_IRI>;
```

These statements are used for adding or removing a location to/from the map. An IRI of a location is used as an identifier to the location. VALUE is an optional parameter for indicating a value that is associated with the location (see Def. 1). A location map is also considered as a location within the framework. Hence, we can also add an already defined location map into another location map.

Insert/delete location relation

```
INSERT RELATION TO <map_IRI> {
   <location_IRI>,<relation_IRI>,<location_IRI> .
   <location_IRI>,<relation_IRI>,<location_IRI> .
   ....}
DELETE RELATION FROM <location_map_IRI> {
   <location_IRI>,<relation_IRI>,<location_IRI> .
   <location_IRI>,<relation_IRI>,<location_IRI> .
   ....}
```

These statements are used for adding or removing relations between a pair of location within the map. A relation is described in a triple format of `<source, relation, destination>`, where `source` and `destination` are location IRIs and `relation` is an IRI of a location relation type.

The predefined location relation types in the *CEMiD* framework were presented in the previous chapter (see Section 4.2.1). Users can also freely propose a new kind of location relation in the case that they would like to propose an application specific location relation. To do so, users need to use the following statement.

Create/remove relation

```
CREATE RELATION <relation_IRI>
       TYPE <relation_type_IRI>
       [VALUE <value>];

DELETE RELATION <relation_IRI>;
<relation_type> = 'TOP' | 'DIS' | 'DIR'
```

Recall from 4.2.1, location relation can be either topological, distance, or directional. `TOP` variable is used as *<relation_type_IRI>* within the statement for indicating that a new relation type to be created is a topological relation. `DIS` is used for creating a new distance relation, and `DIR` is used for creating a new directional relation. The `VALUE <value>` is optional. It is required only if a relation to be created is a distance relation because the distance may required to be quantified numerically in some application (e.g., location A is 10 kilometres far from location B). A new location relation proposed by this statement is modeled as a sub-class pf `mssn:LocationRelation` concept of *MSSN-Onto* (see Figure 4.1.

We illustrate the use of the *CEMiD* language to create a location map in the following example.

```
PREFIX <office:http://cemid.sigappfr.org/office#>
PREFIX <relation:http://cemid.sigappfr.org/rel#>
PREFIX <relation:http://cemid.sigappfr.org/rel_office#>

CREATE MAP office:mapRoom1;

INSERT LOCATION office:table1 TO office:mapRoom1;
INSERT LOCATION office:table2 TO office:mapRoom1;
INSERT LOCATION office:table3 TO office:mapRoom1;

CREATE  RELATION rel_office:nextTo TYPE DIS;

INSERT RELATION TO office:mapRoom1 { office:table1,
   rel:opposite, office:table2 . office:table2,
   rel:opposite, office:table3 . office:table3,
   rel_office:nextTo, office:table1 .
};
```

`PREFIX` commands can be used in the same manner of the SPARQL language such that, an IRI can be specified through the short hand namespace-prefixed form. We first create an empty location map (with IRI as `http://cemid.sigappfr.org/office#mapRoom1`) with the statement `CREATE MAP`. Next, locations `office:table1`, `office:table2`, and `office:table3` are created and added to the map `office:mapRoom1`. We create a new distance relation, designated as `rel_office:nextTo`, with the `CREATE RELATION` statement. The relation type `rel:opposite` is one of the pre-defined location relations provided by our framework. The relation between each pair of locations are defined by `INSERT RELATION TO` statement. Hence, we have the location map `office:mapRoom1`, with three locations related by distance and directional relations. A user can insert `office:mapRoom1` in other location map, let us say `office:building1`, with the statement `INSERT LOCATION office:ma-pRoom1 TO office:building1`.

Beside the user defined relations as indicated within the statement `INSERT RELATION TO`, the framework automatically adds the topological relation `rel:contains` between a location map and every location within the map. In the example presented above, the framework automatically creates relations `<office:mapRoom1, rel:contains, office:table1>`, `<office:mapRoom1, rel:contains, office:table2>`, and `<office:mapRoom1, rel:contains, office-:table3>`.

5.1.1.2 Sensor Network Infrastructure Modeling

After a location map is modeled properly, users can define a sensor network by using *CEMiD* language statements as follows.

Create/Remove a sensor network

```
CREATE NETWORK <sensor_network_IRI>;
DELETE NETWORK <sensor_network_IRI>;
```

These statements are used for creating an empty sensor network or deleting an existing sensor network. This command creates/removes an ABox instance of `ssn:System` within the *MSSN-Onto* repository (see Def. 4).

Create/Remove a sensor type

```
CREATE SENSOR TYPE <sensor_type_IRI>
        MEDIA <media_type_list> METHOD
        <sensing_method_IRI> FEATURE
        <feature_of_interest_list>
        STIMULUS <stimulus_IRI>
        [ CAPABILITIES { <capability>, <value> .
                         <capability>, <value> .
                           ... }];
DELETE SENSOR TYPE <sensor_type_IRI>;
<media_type_list> = "video" | "audio" | "image"| "text"
                    | "numeric"
                    | <media_type_list>,<media_type_list>
```

Recall Def. 3, from Chapter 4, a sensor is modeled as a tuple which contains information regarding (i) media type that it produces, (ii) installation location, (iii) sensing method that it uses, (iv), feature of interest, (v) stimulus, (vi) a set of sensor capabilities and (vii) a set of coverage areas. It can be seen that two of these pieces of information, an installation location and coverage areas, are specific to a sensor instance, while the other informations (i.e., sensing method, feature of interest, stimulus, capabilities, and coverage areas) are generic to every sensor of the same type. (e.g., every video camera within the network may have similar characteristics, but they can be deployed in different location and have different coverage area). The framework takes the information which are generic to every sensor of the same type from the statement as given above and model them as a new TBox concept for modeling a multimedia sensor (i.e., generating a new concept for modeling a multimedia sensor by inheriting from `mssn:MediaSensor` concept). The actual instantiation (ABox instance) of each sensor can be done by using the statement as follows.

Insert/Remove a sensor to/from a network

```
INSERT SENSOR <sensor_IRI> TYPE <sensor_type_IRI> TO
    <sensor_network_IRI> PLATFORM <location_IRI>
    [COVERAGE_AREA <location_IRI_list>]
DELETE SENSOR <sensor_IRI> FROM <sensor_network_IRI> ;
```

These statements are used for adding or removing a sensor to/from a sensor network. The statement creates (or remove) an ABox instance of a sensor according to an already sensor type that is declared through the sensor type declaration statement.

The following example illustrates the usage of the *CEMiD* language for defining a new sensor network.

```
PREFIX <office:http://cemid.sigappfr.org/office#>
PREFIX <network:http://cemid.sigappfr.org/office/network#>
PREFIX <mssn:http://mssn.sigappfr.org/mssn#>

CREATE NETWORK network:room1_network FROM MAP office:mapRoom1;

CREATE SENSOR TYPE network:faceCamera
        MEDIA "video" FEATURE "face"
           CAPABILITIES {  mssn:videoWidth, 1280 .
                           mssn:videoHeight, 720 .
                           mssn:encodingFormat, "h264" . }
```

```
CREATE SENSOR TYPE network:temp
        MEDIA "numeric" FEATURE "Temperature Value"
           CAPABILITIES { mssn:accuracy, 1.0 .
                          mssn:drift, -100  .}

INSERT SENSOR network:camTable1 TYPE network:faceCamera
  TO network:room1_network PLATFORM office:table1
  COVERAGE_AREA office:table1;

   INSERT SENSOR network:tempRoom1 TYPE network:temp
     TO network:room1_network PLATFORM office:table1
  COVERAGE_AREA office:table1, office:table2, office:table3;
```

We first create a sensor network with the IRI as `office:room1_network`, which use the previous defined location map `office:mapRoom1`. Two types of sensors are declared: a video sensor, designated as `network:faceCamera`, and a temperature sensor, called `network:temp`. The `MEDIA` statement is used for declaring the type of output of each sensor. The `network:faceCamera` type sensor has `"video"` as the `MEDIA` value, thus it becomes a multimedia sensor type; while for `network:temp` sensor type, the value `"numeric"` is indicated for the `MEDIA` value, hence, it becomes a scalar sensor. Then, we create an instance of each sensor, `network:CamTable1` from `network:faceCamera` and `network:tempRoom1` from `network:temp`. Sensor `network:CamTable1` is installed in `PLATFORM office:table1` and the coverage area is the same location. Sensor `network:temp-Room1` is also installed in `PLATFORM office:table1`, however its coverage area is the three lo-cations defined in the location map `office:mapRoom1` (i.e., `office:table1`, `office:table2`, and `office:table3`).

5.1.2 Event Modeling

SQL/SPARQL version of the *CEMiD* language contains two separated commands for modeling an atomic event condition and an event statement. Their syntaxes are as follows.
Atomic Event Condition Modeling

```
CREATE COND <event_cond_IRI>
        USING  SENSOR <sensor_type_IRI | sensor_IRI>
               FUNCTION <detection_function_IRI>
               [PARAMS <additional_params>];
DELETE COND <event_cond_IRI>;
```

This statement is proposed in response to Def. 8 (see Chapter 4). The statement `CREATE COND` is used for modeling an atomic event condition. In short, a user needs to define a new IRI to a new atomic event condition. The `USING` clause is used for specifying which sensor or sensor types are compatible with this condition, detection function has to be used, and additional parameters (if any). The `DELETE COND` is used for deleting a condition from the framework. The following example shows how atomic event conditions are described with *CEMiD* language.

```
PREFIX <cond_repo:http://cemid.sigappfr.org/event_repo#>
PREFIX <network:http://cemid.sigappfr.org/office/network#>
PREFIX <cond_func:http://cemid.sigappfr.org/func#>
CREATE COND cond_repo:high_temp
```

```
   USING SENSOR  network:temp
         FUNCTION  func:greater_than
         PARAMS    35;

CREATE COND cond_repo:face_detect
   USING SENSOR  network:faceCamera
         FUNCTION  func:faceDetection;
```

Our example aboves contain two atomic event conditions. The first condition is for modeling an atomic event condition for a *high temperature* event. The sensor used in this condition is a temperature sensor indicated by the IRI `network:temp` (defined previously in the example of sensor network modeling). The detection function used is *func:greater_than*. This function evaluates if the incoming scalar value exceeds the threshold as given in the `PARAMS` clause. The second condition is an atomic event condition for a *face detected* event. Its input is a video camera (also defined in the previous example) and its detection function is `func:faceDetection`. The `func:faceDetection` function is a preprogrammed function for detecting a face in an incoming video or image stream. This function does not require any additional parameter. Hence, the `PARAMS` clause is not given.

After atomic event conditions are properly defined, complex events can be modeled as *CEMiD event statements*. The syntax of a *CEMiD* event statement is given as follows.

Event Statement Modeling

```
CREATE EVENT <event_statement_IRI> STATEMENT
       <spatio_temporal_condition> [FROM
       NETWORK <sensor_network_IRI>] [FILTER
       <filter_statements>]

<spatio_temporal_condition> =
             SEQ (<operand>, <operand>, <t>) |
             OVERLAPS (<operand>, <operand>) |
             <spatial_opr>(<operand>, <operand>)

<operand> = <event_cond_IRI> | <event_statement_IRI> |
            <location_IRI> | <spatio_temporal_condition>

<spatial_opr> = loc:at | loc:<relation_IRI>

<filter_statements> = <filter_statement> |
          <filter_statement> <filt_opr> <filter_statement>
<filter_statement> = <f_operand> <f_opr>
<f_operand> <f_operand> = <operand>
              | <operand.property>.<property>
              | <datetime> #XSD-date
              | <value> #Any alphanumeric value
<f_opr> = '=' | '<=' | '>=' | '<' | '>'
         'and' | 'or' | 'not' | <spatial_opr>
<property> = 'time_begin' | 'time_end' | 'locations' | 'values'
```

In short, an event statement consists of: (i) an IRI to be assigned to the statement; (ii) a spatiotemporal condition for describing the event; (iii) an IRI of a sensor network to be applied to the statement; and (iv) a

filtering statement. The first two components of the statement are mandatory, while the latter components are optional. An event statement is used by the framework for detecting events. Hence, the output of an event statement is an event detection result. An occurrence of an event within a detection result is called an *event occurrence* within the framework. An event occurrence has four main properties:

- time_begin: is a property for describing the timestamp of the beginning of an event occurrence
- time_end: is a property for describing the timestamp of the end of an event occurrence. $time\_end = null$ if an event is still on-going
- locations: is a set of locations that are related to the event occurrence
- values: is the list of observation values that are related to the event occurrence.

A spatio-temporal condition of the *CEMiD* event statement is used for modeling a pattern of events in term of relations between event occurrences. The language proposes two temporal operators and two spatial operators. The temporal operators are `SEQ`, and `OVERLAPS`. The spatial operators are the `loc:at`, and `loc:relation_iri`. Given that $ec1$ and $ec2$ are event occurrences which are produced using conditions $op1$ and $op2$, all of SEQ, OVERLAPS, loc:at, and loc:(relation_iri) are defined as follows:

- `SEQ` operator describes sequential events, denoted as: `SEQ(op1,op2,t)=`$\{< ec1_1, ec2_1 >, ..., < ec1_n, ec2_n >\} \leftrightarrow ec2_i.time\_begin - ec1_i.time\_end \leq t$ `and` $ec1_i$ `is an event occurrence detected by op1, while` $ec2_i$ `is an event occurre-nce detected by op2.`;
- `OVERLAPS` operator is used for describing that two event occurrences happen during the same period, denoted as `OVERLAPS(op1, op2)=`$\{< ec1_1, ec2_1 >, .., < ec1_n, ec2_n >\} \leftrightarrow (ec1_i.time\_begin < ec2_i.time\_begin) \wedge ((ec2_i.time\_begin < ec1_i.time\_end) \vee (ec1_i.time\_end = null))$, `and` $ec1_i$ `is an event occurrence detected by op1, while` $ec2_i$ `is an event occurrence detected by op2` ;
- `loc:at` operator is used for expressing that event occurrences happen at the same location, denoted as, `loc:relation_IRI(op1, op2) =` $\{< ec1_1, ec2_1 >, .., < ec1_n, ec2_n >\} \leftrightarrow (ec1_i.locations = ec2_i.locations)$;
- `loc:relation_IRI)` expresses the condition that event occurrences have locations related according to the `relation_IRI`, which is a topological, directional, or distance relation, as given in a predefined location map, denoted as; `Given a location map` $LM =< iri, V, E, F >$ `(Def. 2), loc:<relation_IRI>(op1, op2)=`$\{< ec1_1, ec2_1 >, .., < ec1_n, ec2_n >\} \leftrightarrow (ec1_i.l = op1.l \wedge ec2_i.l = op2.l) \wedge \exists (ec1_i.l, ec2_i.l) \in E \wedge relation\_I\ RI \in F(ec1_i.l, ec2_i.l)$, $ec1_i$ `is an event occurrence detected by op1,` $ec2_i$ `is either an event occurrence detected by op2 or a p-redefined location IRI in` LM.

Beside the spatio-temporal condition, the *CEMiD* event statement also allows users to specify the filtering statement by using the FILTER clause, for filtering the result produced from STATEMENT clause according to a user-defined condition. The filtering statement is designed based on the same principle of the `WHERE` clause of *SQL* and the `FILTER` clause of SPARQL. The related operators that a user can use for writing a filtering statement are `'='`, `'<='`, `'>='`, `'<'`, `'>'`, `'and'`, `'or'`, and `'not'`. It is noted that the FILTER clause does not appear as a seperate variable in Def. 9 in Chapter 4 as we assume that the FILTER clause is also a part of an event statement thus, it should be integrated as a part of the text describing the statement itself already.

An example of a *CEMiD* event statement is as follows.

```
PREFIX <cond_repo:http://cemid.sigappfr.org/event_repo#>
PREFIX <stm:http://cemid.sigappfr.org/stm#>
PREFIX <rel:http://cemid.sigappfr.org/rel#> PREFIX
<office:http://cemid.sigappfr.org/office#>
PREFIX <network:http://cemid.sigappfr.org/office/network#>

CREATE EVENT stm:people_feels_hot
 STATEMENT
  OVERLAPS( (loc:at (cond_repo:face_detect,office:table1),
            loc:at (cond_repo:high_temp, office:mapRoom1)))
  FROM NETWORK network:room1_network;
  FILTER (stm:people_feels_hot.time_begin
                                  > 2017-05-30T09:00:00)
```

In this example, we define an event statement to detect if *People feels hot*. This event can be described as an event where a camera sensor detects a face of a person (presumably a worker) in `table1` location, while a high-temperature event (`cond:high_temp`) is detected in the room described by `mapRoom1`. The `FROM NETWORK` clause is used for specifying that the statement `stm:people_feels_hot` will be applied only to sensors of the network `network:room1_network`. Next, the *CEMiD* framework will filter the result according to the condition

```
stm:people_feels_hot.time_begin > 2017-05-30T09:00:00.
```

Thus, the final result of this event statement is a set of event occurrences according to `stm:people_feels_hot` and the beginning timestamp of each of them is after 9:00 AM on 30th May 2017.

5.1.3 Action/Report Syntax

In the *CEMiD* framework, users can define which actuator(s) to trigger when a certain event is detected. Users can also define how a detected event should be reported. The syntax of an Action/Report statement of the *CEMiD* language is given as follows.

```
ON <event_statement_list> <action_statement>

<event_statement_list> = <event_statement_IRI> |
     <event_statement_IRI>, <event_statement_list>
<action_statement> = ACTIVATE <actuator_IRI>
     | REPORT <occurrence_properties> |
     ACTIVATE <actuator_IRI> REPORT <occurrence_properties>

<occurrence_properties> =
     <event_statement_IRI>.<property> |
     <event_statement_IRI>.<property>,<occurrence_properties>
<property> = 'time_begin' | 'time_end' | 'locations' | 'values'
```

The syntax as given above is inspired from an *Event-Condition-Action* style syntax. In short, the framework will detect events according to event statements that are given in `<event_statement_list>`. Whenever at least one of them is detected, the system will signal the activator as indicated by the

<actuator_IRI>. The system will also generate a report to the user according to <occurrence_pro-perties> which can be either time_begin, time_end, locations, or values. The usage of both ACTIVATE and REPORT are not obligatory. However, users need to specify at least either ACTIVATE or REPORT in their statements.

The example of a near real-time event detection statement is given as follows.

```
PREFIX <stm:http://cemid.sigappfr.org/stm#>
PREFIX <act:http://cemid.sigappfr.org/act#>
ON stm:people_feels_hot ACTIVATE act:HVAC_room1
REPORT stm:people_feels_hot.time_begin,
       stm:people_feels_hot.values;
```

The statement as given above is used for instructing the framework that it has to activate the actuator with the IRI act:HVAC_room1 whenever an event occurrence according to a statement stm:people_feels_hot is detected. The REPORT clause indicates that the framework should report the beginning timestamp and a list of observation values of each detected event occurrences to the user.

5.1.3.1 Historical Event Querying Syntax

As seen in Figure 3.1, every detected event occurrence is stored within the *CEMiD* repository for historical event querying later on. The syntax of the *CEMiD* language for the historical event querying is as follows.

```
SELECT <event_statement_list>
[FROM <sensor_network_IRI>]
[WHERE <filter_statement>]
[REPORT <occurrence_properties>];
```

The syntax of the statement above is inspired from the SELECT.. FROM .. WHERE clause of the SQL. The SELECT clause is used for selecting which event statements to be used for querying. The FROM clause is to specify which sensor network that an event occurence should be queried from. The parameter of FROM clause is an IRI to a specified sensor network (see *Create/Remove a sensor network* statement). The filter statement is used for filtering only the output that is matched with a filtering statement. The syntax of a filtering is similar to the filtering part of the *CEMiD* event statement definition. The REPORT clause is used for selecting which properties of each event occurrence to report to a user. This clause is optional. If it is not specified, the framework will report all properties of each event occurrence. The example a historical event querying statement is given as follows.

```
PREFIX <stm:http://cemid.sigappfr.org/stm#>
PREFIX <network:http://cemid.sigappfr.org/office/network#>
SELECT stm:people_feels_hot
FROM network:room1_network
WHERE (stm:people_feels_hot.time_begin > 2017-05-
30T09:00:00) REPORT stm:people_feels_hot.time_begin,
       stm:people_feels_hot.values;
```

The query above is used for selecting an already detected event occurrences from the *CEMiD* repository that are detected by using a statement stm:people_feels_hot. The FROM and WHERE clauses indicate that the event occurrences that are going to be reported have to be an event occurrence which hap-pens in the sensor network network:room1_network and the beginning timestamp of the occurrence has to be after 9:00 AM of 30th May 2017.

5.2 CEMiD Language: JSON Serialization

The syntax of the *CEMiD* language, as already described, is mostly inspired from SQL/SPARQL and ECA style language. The language is designed to be expressive and intuitive to users. However, in some use-case, users may need to serialize the language by using JSON[1] such as, web-application development use-case. JSON syntax may be less expressive and less intuitive when comparing with SQL/SPARQL style syntax. However, it offers users more compactness of the syntax, and also allow the ease of interoperating the data with external web-application. Therefore, we propose a template for serializing the *CEMiD* language into JSON. They are described as follows.

```
{ //location map model
   "location_map":{
      "map_name":<location map name>,
      "locations":[<location name>, ...,
      <location_name>] "relations":[
            {"source":<location name>,
            "relation":<relation name>,
            "destination":<location name>},
             .....]},

   //sensor network
   model "network": {
    "name": <network name>,
    "map": <location map
    name>, "sensors":[
     {"sensor":<sensor name>
      "type":<sensor type name>,
      "location":<installation location>,
      "output_format":[ <output type>, ... <output
      type>], "capabilities":{
            <capability_name>:<value>,
            ..,
        }
     }, ....]},

  //events model
   "atomic_event_conditions":[
  {"cond_name": <atomic condition name>,
  "sensors": [ <list of sensors> ],
  "function": <IRI of a detection
  function>, "params": [list of additional
  parameters] },....],
  "event_statements":[
    {"name":<event statement name>,
     "statement", <event statement>,
     "actuator", <IRI of an actuator>},
      ...]}
```

[1] JSON (http://www.json.org/) is an abbreviation of JavaScript Object Notation. JSON is a data model which is widely used in a web-application development context.

The file above is a template that *CEMiD* framework uses for serializing *CEMiD* statements into JSON. The template consists of three main sections: (i) location map model, (ii) sensor network model, and (iii) event model. The example of the usage of using JSON-serialized version of the CEMiD language is given as follows.

```
{ "location_map":{
    "map_name": "Room204Campus1",
    "locations":["table1", "table2", "table3" ]
    "relations":[
         {"source":"table1", "relation":"opposite",
                              "destination":"table2"},
         {"source":"table2", "relation":"nextTo",
                              "destination":"table3"}]},
  "network": {
    "name": "Room204Network", "map":
    "Room204Campus1", "sensors":[
       {"sensor":"cam1",
           "type":"MediaSensor", "location":"table1",
           "output_format":["video"],
           "capabilities":{"width":640, "height":480,
                 "codec":"MPEG4"}},
     {"sensor":"temp1",
           "type":"Sensor", "location":"table2",
           "output_format":["numeric"],
           "capabilities":{"drift":-5, "precision": 1.0}
    },]}
  "atomic_event_conditions":[
   {"cond_name": "face_detected", "sensors": [ "cam1" ],
   "function":"opencv:haarcascade","params":"haar.xml"},
   {"cond_name": "high_temp", "sensors": [ "temp1" ],
   "function": "math:greater_than", "params": 30} ],
  "event_statements":[
   { "name":"people_feels_hot",
   "statement":"OVERLAPS(face_detected, high_temp)"}
   "actuator":"http://campus1.sigappfr.org/room1/hvac",
   { "name":room_becomes_hot",
     "SEQ(face_detected, high_temp, 2)",
     "actuator":"http://campus1.sigappfr.org/room1/hvac"
   }]}
```

This JSON file example contains a location map, a sensor network, two atomic event conditions, and two event statements. In short, a location map named `Room204Campus1` is created. Three locations `table1`, `table2`, and `table3` are added to the map. The relations among them are: `table1` is `opposite to table2`, and `table2` is `nextTo table3`. Two sensors, `cam1` and `temp1` are created. `cam1` is a video camera which produces an *MPEG4* encoded video stream with video dimension of 640x480. `temp1` is a temperature sensor which can detect a temperature level with drift value of -5 and precision of 1.0.

Two atomic event conditions are defined in this example. The first condition is `face_detected`, which takes inputs from `cam1` sensor and pass to the HaarCascade function of the OpenCV library[2]. The

[2] http://opencv.org/

`opencv:HaarCascade` is one of a built-in internal library in *OpenCV* for detecting a face of a person according to pre-trained data file (indicated by `"params":"haar.xml"` in the example) . The second atomic event condition is `high_temp`, which uses a built-in function `greater_than` for determining if the sensor `temp1` reads a value higher than 30. These two atomic event conditions are used for modeling two event statements.

5.3 CEMiD Language Processing

The *CEMiD* language serves as the high-level interface for users to communicate with the framework. The framework repository is an ontology-based repository which uses the *MSSN-Onto* as the core ontology. This kind of repository can only work with a low-level language for manipulating ontology repository such as the SPARQL. Hence, in order to be able to allow users to communicate with the framework by using the *CEMiD* language, the framework needs to translate all the *CEMiD* language statements into SPARQL statements to communicate with all the repositories within the framework.

In general, all the *CEMiD* statements that are related to location map modeling, sensor network modeling and event modeling, are translated by the framework into an INSERT statement of the SPARQL language. This allows the data to be added into the framework repository which is an ontology-based repository. The template of the statement is given as follows.

```
PREFIX <rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <rdfs:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <schema:http://schema.org/>
PREFIX <ssn:http://www.w3.org/ns/ssn/> PREFIX
<sosa:http://www.w3.org/ns/sosa/> PREFIX
<mssn:http://mssn.sigappfr.org/mssn#>
<additional_prefixes>

INSERT { <predicates>
}
```

The template statement above is the template of the INSERT statement used by the framework. All the prefixes that are common to all the SPARQL statements in general are added which are `rdf`, `rdfs`, and `schema`. These prefixes are follows by the prefixes used by the *CEMiD* framework which are `ssn`, `sosa`, and `mssn`. The additional prefixes that may be used by an application domain are added to the `<additional_prefixes>` within the statement template.

All the *CEMiD* statements for modeling a location map, a sensor network, and an event are translated into a SPARQL predicate format. The basic structure of a SPARQL predicate is as follows

```
<subject> <predicate> <object> .
```

The basic structure of SPARQL predicate consists of three sections which are `subject`, (ii) `predicate` and (iii) `object`. The `subject` and `object` are instances of objects, classes, or resources to be described. The `predicate` describes the relation between `subject` and `object`. Each line of predicate is terminated by using `.` symbol. The example of SPARQL predicates are given as follows.

```
office:room1 rdf:type ssn:System .
office:room2 rdf:type ssn:System .
office:room1 rel:nextTo office:room2 .
```

The example above gives three SPARQL predicates. The first two predicates use the `rdf:type` predicate to describe that the resources `office:room1` and `office:room2` are `ssn:System` type. The third predicate is used for describing that the `office:room1` and `office:room2` are related according to the predicate `rel:nextTo`.

The following sub-sections describe how *CEMiD* statements for modeling a location map, a sensor network infrastructure, and event can be translated and reconstructed into SPARQL predicates.

5.3.1 SPARQL predicates for modeling a location map

To construct the SPARQL predicates for modeling location maps from a *CEMiD* statement, the *CEMiD* statements are parsed into three main pieces of information which are (i) a location map identifier, (ii) list of locations within the map, and (iii) list of relations between locations.

Algorithm 1 Algorithm for generating location map predicates

```
1: function GEN_MAP_PREDICATES(map_I RI , Locations, Relations)
2:     predicates = map_I RI + " rdf:type mssn:LocationMap ."
3:     for each location in Locations do
4:         predicates += location[I RI] + "rdf:type mssn:Location ."
5:     for each relation in Relations do
6:         src_IRI= relation[src]
7:         rel_IRI= relation[rel]
8:         dst_IRI= relation[dst]
9:         predicates += src_IRI+" "+rel_IRI+" "+dst_IRI .
10:    return predicates
```

The algorithm for constructing SPARQL predicates are given in Algorithm 1. The algorithm takes three main parameters that has to be parsed from CEMiD statements. The first parameter is *map_I RI* which is an IRI of a location map to be modeled. The second parameter is *Locations* which contains a set of locations to be serialized into SPARQL. Each of *Locations* has IRI to the location to be modeled. The final parameter is *Relations* which contains a set of location relations. Recall from location map mod-eling statements, a relation is modeled as a triple of `source, relation, destination`. Hence, `relation[src], relation[rel]` and `relation[dst]` refer to `source, relation,` and `destination` components of a location relation triple. The algorithm is firstly generated a new location map by creating a predicate `<map_IRI> rdf:type mssn:LocationMap ..` Next, for each loca-tion, a predicate `<location[IRI]> rdf:type mssn:Location .` is created to declare a new location. Finally, every location relations predicates are generated by the last for loop. For example, given the *CEMiD* statement in JSON serialized format as follows:

```
{ "location_map":{
   "map_name": "Room204Campus1",
   "locations":["table1", "table2", "table3" ]
   "relations":[
     {"source":"table1", "relation":"opposite", "destination":"table2"},
     {"source":"table2", "relation":"nextTo", "destination":"table3"}]},
```

the example statement is reconstructed into the SPARQL query as follows:

```
PREFIX <rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <rdfs:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX <schema:http://schema.org/>
PREFIX <ssn:http://www.w3.org/ns/ssn/>
PREFIX <sosa:http://www.w3.org/ns/sosa/>
PREFIX <mssn:http://mssn.sigappfr.org/mssn#>
PREFIX <map:http://mssn.sigappfr.org/map#>
PREFIX <relation:http://mssn.sigappfr.org/mssn/relation#>

INSERT {
   map:Room204Campus1 rdf:type mssn:LocationMap .

   map:table1 rdf:type mssn:Location .
   map:table2 rdf:type mssn:Location .
   map:table3 rdf:type mssn:Location .

   map:table1 relation:opposite mssn:table2 .
   map:table2 relation:opposite mssn:table3 .
}
```

In short, the example query contains one location map named `Room204Campus1` and three locations. Hence, the INSERT statement creates the map, all the three locations, and predicates for describing relations between locations as given in the *CEMiD* statement into the SPARQL predicate format.

5.3.2 SPARQL predicates for modeling sensor network infrastructure

To generate SPARQL predicates for generating a sensor network, the IRI to a sensor network to be created and information of every sensor within the network are parsed from the *CEMiD* statement and send into Algorithm 2.

Algorithm 2 Algorithm for generating sensor network infrastructure predicates

```
1:  function GEN_SENSOR_PREDICATES(sn_I RI , Sensors)
2:      predicates = sn_I RI + " rdf:type ssn:System ."
3:      for each sensor in Sensor do
4:          iri = sensor[I RI]; type = sensor[typeI RI]; = sensor[loc]
5:          predicates += iri + " rdf:type " + type + " ."
6:          predicates += iri + " sosa:onPlatform " + loc + " ."
7:          predicates += sn_I RI + " ssn:hasSubSystem " + iri + " ."
8:          for each cap in sensor[cap] do
9:              capIRI = cap[I RI]
10:             capValue = cap[value]
11:             predicates += capIRI +" rdf:type ssn:Property ."
12:             predicates += capIRI +" mssn:hasValue "+ capValue .
13:             predicates += iri + " ssn:hasSystemCapability " + capIRI + " ."
14:     return predicates
```

Algorithm 2 utilizes a nested for loops to construct predicates for modeling a sensor network. The algorithm takes a sensor network IRI (denoted as $sn_I RI$), and a set of sensors in the network (de-noted as *Sensors*) as an input. It is to be noted that `cap[IRI]`, and `cap[value]` are taken from `CAPABILITIES` clause of the sensor creation statement. `cap[IRI]` is the identifier to a capability to be modeled, and `cap[value]` is the value of the capability. (e.g, mssn:videoWidth capability can have 640 as its value for modeling that the width of the video to be captured is 640 pixels.). First, the algorithm creates a new predicate `<sn_IRI> rdf:type ssn:System ..` Next, for every *sensor* within *Sensors*, predicates for modeling its type, location, and capabilities are generated. To illustrate the SPARQL predicate for modeling sensor infrastructure, given the CEMiD statement in JSON serialization format as follows:

```
"network": {
    "name": "Room204Network", "map":
    "Room204Campus1", "sensors":[
       {"sensor":"cam1",
           "type":"MediaSensor", "location":"table1",
           "output_format":["video"],
           "capabilities":{"width":640, "height":480}},
        "sensor":"cam2",
           "type":"MediaSensor", "location":"table2",
           "output_format":["video"],
           "capabilities":{"width":640, "height":480}},
                }
```

the example statement is reconstructed into the SPARQL query as follows:

```
PREFIX <rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <rdfs:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <schema:http://schema.org/>
PREFIX <ssn:http://www.w3.org/ns/ssn/>
PREFIX <sosa:http://www.w3.org/ns/sosa/>
PREFIX <mssn:http://mssn.sigappfr.org/mssn#>
PREFIX <map:http://mssn.sigappfr.org/map#>
PREFIX <network:http://mssn.sigappfr.org/network#>
PREFIX <app:http://mssn.sigappfr.org/app#>

INSERT {network:Room204Network rdf:type ssn:System .
   app:Cam1 rdf:type mssn:VideoSensor .
   app:Cam2 rdf:type mssn:VideoSensor .
   app:Cam1 sosa:onPlatform mssn:VideoSensor .
   app:Cam2 sosa:onPlatform mssn:VideoSensor .

   network:Room204Network ssn:hasSubSystem app:Cam1 .
   network:Room204Network ssn:hasSubSystem app:Cam2 .

   app:capWidth640 rdf:type ssn:Property .
   app:capWidth640 mssn:hasValue "640" .

   app:capHeight480 rdf:type ssn:Property .  .
   app:capWidth480 mssn:hasValue "480" .
```

```
app:Cam1 ssn:hasSystemCapability app:capWidth640 .
app:Cam1 ssn:hasSystemCapability app:capWidth480 . }
```

The example query models a sensor network with two video cameras. The sensor network name, sensor list, installation locations and capabilities are re-modeled into the SPARQL INSERT statement.

5.3.3 SPARQL predicates for modeling events

To generate SPARQL predicates for modeling events, all *CEMiD* statements for modeling atomic event conditions and event statements are parsed and send to Algorithm 3.

Algorithm 3 Algorithm for generating event predicates

```
1: function GEN_EVENT_PREDICATES(AConds, Statements)
2:     for each acond in AConds do
3:         iri = acond[I RI]
4:         sensor = acond[sensorI RI]]
5:         func = acond[f uncI RI]
6:         value = acond[f uncV alue]
7:         predicates += iri + " rdf:type mssn:AtomicEventCondition . "
8:         predicates += iri + " mssn:hasSensor " + sensor + " ."
9:         predicates += iri + " mssn:hasFunction " + func + " ."
10:        predicates += iri + " mssn:hasParameter " + value + " ."
11:    for each stm in Statements do
12:        iri = stm[I RI]
13:        stm_text = stm[statement]
14:        act = stm[actuator]
15:        predicates += iri + " rdf:type mssn:EventStatement . "
16:        predicates += iri + " mssn:hasEventStatement " + stm_text + " ."
17:        predicates += iri + " mssn:hasActuator " + act + " ."
18:    return predicates
```

Algorithm 3 takes two inputs: a set of atomic event conditions (denoted as *ACond*), and a set of event statements (denoted as *Statements*). The first for loop is used for generating predicates for modeling each atomic event condition *acond* within *ACond*. The second for loop is used for generating predicates for modeling each event statement *stm* in *Statements*. To illustrate the SPARQL predicate for modeling events, given the CEMiD statements for modeling atomic event conditions and event statements, in JSON serialization format as follows:

```
"atomic_event_conditions":[
  {"cond_name": "face_detected", "sensors": [ "cam1" ],
   "function":"opencv:haarcascade","params":"haar.xml"},
  {"cond_name": "body_detected", "sensors": [ "cam2" ],
   "function": "opencv:haarcascade", "params": "haar_body.xml"} ],
 "event_statements":[
  { "name":"face_and_body",
   "statement":"OVERLAPS(face_detected, body_detected)"}
   "actuator":"http://campus1.sigappfr.org/room1/hvac",
  }
```

the example statements can be reconstructed into SPARQL statement as follows:

```
PREFIX <rdf:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <rdfs:http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX <schema:http://schema.org/>
PREFIX <ssn:http://www.w3.org/ns/ssn/>
PREFIX <sosa:http://www.w3.org/ns/sosa/>
PREFIX <mssn:http://mssn.sigappfr.org/mssn#>
PREFIX <map:http://mssn.sigappfr.org/map#>
PREFIX <relation:http://mssn.sigappfr.org/mssn/relation#>
PREFIX <event:http://mssn.sigappfr.org/mssn/event#>
PREFIX <app:http://mssn.sigappfr.org/app#>

INSERT {

event:face_detected rdf:type mssn:AtomicEventCondition .
event:face_detected mssn:hasSensor app:cam1.
event:face_detected mssn:hasParameter "haar.xml" .
event:face_detected mssn:hasFunction "opencv:HaarCascade" .

event:body_detected rdf:type mssn:AtomicEventCondition .
event:body_detected mssn:hasSensor app:cam2.
event:body_detected mssn:hasParameter "haar_body.xml" .
event:body_detected mssn:hasFunction "opencv:HaarCascade" .


event:face_and_body rdf:type mssn:EventStatement .
event:face_and_body mssn:hasEventStatement
                "OVERLAPS (event:face_detected, event:body_detected)" .
event:face_and_body mssn:hasActuator
                http://campus1.sigappfr.org/room1/hvac .
```

The example above depicts two atomic event conditions and an event statement which is constructed from two camera sensors. Atomic event conditions that are modeled are face detection event and full body detection event. The event statement is modeled to express the case when one camera detects a face and another camera detect a full body at the same time. It is noted that this example is for illustration purpose only. Hence, it is designed to be concise and easy to read, but the event definitions may not reflect the actual use-case.

5.4 Validation

So far, we describe the *CEMiD* language and how the *CEMiD* framework translates *CEMiD* language statements into SPARQL statements. It can be seen that the *CEMiD* language statements are considerably shorter than using solely SPARQL statements. This means that using *CEMiD* language helps users to avoid writing a long SPARQL statements. This section is dedicated to explaining the experiments done to measure the difference in query sizes through *CEMiD* Language, JSON Serialization version of *CEMiD*, and using purely SPARQL. These experiments verify the compactness of the *CEMiD* language. To do

so, we create a simulation script which generate all the queries in *CEMiD* language, their equals JSON serialized, and pure SPARQL serialized query based on our pre-configured simulated scenario.

The simulated script generated statements in a sensor network application with 20 to 100 locations within a location map, 20 to 100 video camera sensors within a network, 20 to 100 atomic events, and 20 to100 event statements, all of them incremented by 20 in each simulation step. The query size of each version of the *CEMiD* language and pure SPARQL language is calculated to compare the compactness level of the three versions. The query size is measured in terms of the number of no-space literals within the statement (e.g., number of tokens in "INSERT INTO map:test2" is three tokens).

The experiment was conducted by simulating a network with 20 locations, 20 video camera sensors, 20 atomic events and 20 event statements. Each of these parameters is increased at the incremental of 20 until 100 of locations, video cameras, atomic events and event statements are all simulated. The result is given in Table 5.1.

Table 5.1 – Query Size and Compactness Percentage

Number of Locations, Sensors, Atomic Event Conditions and Event Statements	Query Size (Number of Tokens)			Compactness (%)	
	Pure SPARQL	CEMiD Language		CEMiD	JSON-CEMiD
		CEMiD	JSON-CEMiD		
20	1,578	772	493	51.08%	68.75%
40	3,138	1,512	973	51.82%	68.99%
60	4,698	2,252	1,453	52.06%	69.07%
80	6,258	2,992	1,933	52.19%	69.11%
100	7,818	3,732	2413	52.26%	69.13%

Results of calculating the query size of CEMiD language, JSON-CEMiD serialization version and, using purely SPARQL language are shown in Table5.1. The compactness percentage of using *CEMiD* language, comparing with using purely SPARQL, is approximately 52% in all cases. Adopting JSON serialized version of the *CEMiD* language is furthered increase the compactness to approximately 69% in all cases. This proves that the *CEMiD* language helps the users to greatly reduce the size of the statements that they need to write for modeling data into the framework repository. The JSON-serialization version of the language shows more compactness level than using actual *CEMiD* language syntax. However, the expressivity and readibility of the JSON-serialization version are less than the actual *CEMiD* language itself. Nevertheless, developers may choose to either use the full *CEMiD* language or adopted only the JSON serialized version in favor of its compactness, but with the tradeoff on limiting the functionality, expressivility and readliblity of the statement.

5.5 Conclusion

In this chapter, we introduce CEMiD language which is a language for modeling multimedia sensor networks and events that users would like to detect. CEMiD language adopts SQL/SPARQL and ECA style language syntax. By having CEMiD language, users do not need to interact with the *MSSN-Onto*-based repository of our framework directly. Thus, the compactness of the query which users need to write is approximately 52% comparing with using purely SPARQL. CEMiD language also proposes a JSON serialized version which may have limited functionalities, but the syntax is more concised. The JSON serialized version of CEMiD language helps to increase the compactness of the query further to 69%. The latter part of our contribution is the complex event processing engine which process and detect complex events which takes users' queries provided in CEMiD language (and also *MSSN-Onto*) as inputs. These are described in the following chapter.

Chapter 6

GST-CEMiD: Complex Event Processing Engine Algorithms, Implementation Details and Validation

So far, we have presented the overview architecture of the *CEMiD* framework and two of the three main components of the framework (i.e., *MSSN-Onto* and *CEMiD* language). This chapter is dedicated to the description of all the algorithms and implementation details that are needed for developing the complex event processing engine and the full *CEMiD* framework. We first describe all the event detection algorithms of the framework. Then, we explain how the *CEMiD* framework can be implemented in practice. The core engine of the *CEMiD* framework for processing events is called the *GST-CEMiD*. This chapter ends with the experimental validation of our framework in a Smart Office domain.

6.1 Algorithms for Processing Complex Events of the CEMiD frame-work

The Complex Event Processing Engine is responsible for detecting events, as defined by the users through the *CEMiD* language, from incoming sensor data streams. The detection process happens automatically in response to every incoming sensor readings and according to the location map, sensor network infrastructure, and event statements provided by users. This section describes, through an illustrative example, the algorithm to detect events used by the Complex Event Processing Engine. Figure 6.1 shows the example.

The engine keeps a queue for every atomic event condition presented in an event statement. In the example of Figure 6.1, the event statement is `OVERLAPS(OVERLAPS(face_detect, working), high_temp)`, which comprises three atomic events: `face_detect`, `working`, and `high_temp`. Hence, three queues are created. Next, another queue for storing complex events is also created in response to each event operator. In the example in Figure 6.1, the system creates two queues. One is for evaluating `face_detect OVERLAPS working` and the other one is for evaluating `OVERLAPS (OVERLAPS-(face_detect, working), high_temp)`. The algorithm for constructing queues for evaluating an event statement can be written by using a recursive algorithm, as shown in Algorithm 4.

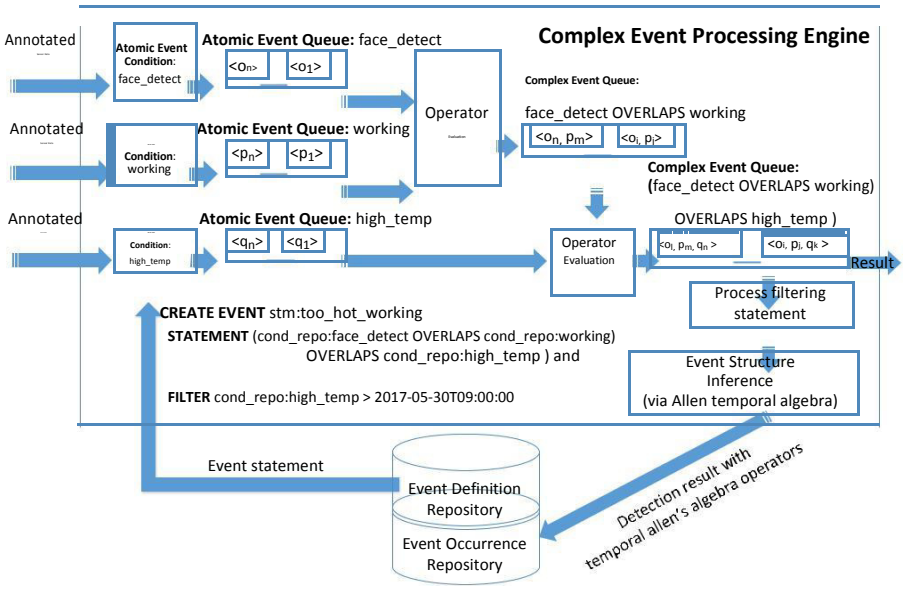


Figure 6.1 – Illustration: Event Detection Algorithm

The Algorithm 4 works by recursively binarizing the statement into the form of *op*1, *opr*, *op*2 (line 2), where *op*1 and *op*2 are operands and *opr* is an operator. The operands can be either an atomic event condition or an event statement (lines 3 to 6). The function *create_atomic_queue* is used for creating a queue for an atomic event condition operand (lines 4 and 9). The function *create_complex_queue* is used for creating a complex event queue (line 12). On each of these queues, it is stored the detection result for the corresponding atomic or complex event condition, represented as an *event occurrence*. An *event occurrence*, according to Def. 10 (presented in Section 4.2.4 of Chapter 4), does not have any spatio-temporal property (i.e., timestamp, locations, values) by default. This information is automatically extracted within the framework through the functions `get_time_start`, `get_time_end`, `get_locations`, and `get_values`, which are built-in functions only used by the framework (i.e., users do not needed to use them).

Algorithm 4 Algorithm for generating queues

```
1:  procedure PROCESS_STATEMENT(event_statment)
2:      op1, opr, op2 = binarized_statement (event_statement)
3:      if is_atomic(op1) then
4:          create_atomic_queue(op1)
5:      else
6:          process_statement(op1)
7:      if opr , null then
8:          if is_atomic(op2) then
9:              create_atomic_queue(op2)
10:         else
11:             process_statement(op2)
12:         create_complex_queue(op1, opr, op2)
```

Regarding the temporal event operators, our study chooses to adopt the operators as indicated in Allen's interval algebra [43]. However, unlike previous studies, we deem that not every Allen's interval algebra operator is suitable for a near real-time event detection. Thus, for near real-time event detection in *CEMiD* framework, only the SEQ and OVERLAPS operators are considered. Other Allen's operators are used only in past event querying process.

Algorithm 5 and Algorithm 6 show the procedures that are executed to evaluate the SEQ and OVER-LAPS operators, respectively. Given two event queues, P and Q, representing operands of SEQ and OVERLAPS, an *event occurrence* is added to a resulting queue, R, if the corresponding SEQ or OVER-LAPS conditions is met. Both algorithms work by spawning an infinite loop thread. First, each algorithm waits for event occurrences to arrive in both queues, P and Q (lines 3 and 4 in both algorithms). When at least one event occurrence arrives in each queue, their start and end times are extracted (lines 5 to 9 in both algorithms). Then the SEQ and OVERLAPS operators conditions are evaluated (lines 10 in Alg.5 and Alg. 6). If the condition is satisfied, both occurrences are combined into a new occurrence, pushed to the result queue R, and dequeued from P and Q (lines 11 to 17 in Alg.5 and Alg. 6). If the condition is not satisfied, the queue head with the lowest timestamp (i.e., the oldest event occurrence) is dequeued from their queue.

Algorithm 5 SEQ Operator Algorithm

```
1:  procedure SEQ(P, Q, t, R)
2:      while true do
3:          wait_for_occurrence(P)
4:          wait_for_occurrence(Q)
5:          p = P.head, q = Q.head
6:          t_p ← get_time_start(p)
7:          t_p ← get_time_end(p)
8:          t_q ← get_time_start(q)
9:          t_q ← get_time_end(q)
10:         if (t_p ≠ null) ∧ (t_q ≥ t_p) ∧ (t_q − t_p ≤ t) then
11:             tk_r = < p, SEQ, q, t >
12:             push_to_queue(R, tk_r)
13:             dequeue(p), dequeue(q)
14:         else if (t_p ≤ t_q) then
15:             dequeue(p)
16:         else
17:             dequeue(q)
```

SEQ and OVERLAPS operators may be sufficient to be used in a near real-time event detection case. However, they might not be sufficient for modeling complex events in a historical event querying case. In such a case, according to Allen's interval algebra, the SEQ and OVERLAPS operator can be replaced with other operators that can describe event structure in a more precise manner if a certain condition is met. For example, the `A OVERLAPS B` can be replaced with `A EQUALS B` if the duration of A and B are exactly equal. `SEQ(A, B, 2)` can be replaced with `A MEETS B` if `A` finishes at the same time that `B` starts.

In *CEMiD* framework, right before an event is stored within the *CEMiD* repository, the framework will try to determine whether the SEQ and OVERLAPS operators can be replaced with a more precise temporal operator according to Allen's interval algebra or not. The operators that are considered to replace SEQ operator are MEETS, MET BY, FINISHED BY, and FINISHES. The operators that are considered to replace OVERLAPS operator are CONTAINS, DURING, STARTS, STARTED BY, and EQUALS. Hence, given these following variables:

Algorithm 6 OVERLAPS Operator Algorithm

```
1:  procedure OV ERLAPS(P, Q, R)
2:      while true do
3:          wait_f or_occurrence(P)
4:          wait_f or_occurrence(Q)
5:          p = P.head, q = Q.head
6:          t_p ← get_time_start(p)
7:          t_p ← get_time_end(p)
8:          t_q ← get_time_start(q)
9:          t_q ← get_time_end(q)
10:         if ( t_p ≤ t_q) ∧ ((t_q ≤ t_p ) ∨ (t_p = null)) then
11:             tk_r =< p, OV ERLAPS, q, null >
12:             push_to_queue(R, tk_r )
13:             dequeue(p), dequeue(q)
14:         else if (t_p ≤ t_q) then
15:             dequeue(p)
16:         else
                dequeue(q
            )
17:
```

- ec_1, ec_2: are event occurrences;
- $ec_1.t$ = $get\_start\_time(ec_1)$;
- $ec_2.t$ = $get\_start\_time(ec_2)$;

The operators between ec_1 and ec_2 in Table 6.1 are met.

- $ec_1.t' = get\_end\_time(ec_1)$;
- $ec_2.t' = get\_end\_time(ec_2)$;

are replaced with a more precise operator when conditions as given

Table 6.1 – Allen's Interval Algebra Operators

Relation	Condition
meets	$ec_1.t = ec_2.t$
met by	$ec_2.t = ec_1.t$
finished by	$ec_1.t < ec_2.t \wedge ec_1.t = ec_2.t$
finishes	$ec_1.t > ec_2.t \wedge ec_1.t = ec_2.t$
contains	$ec_1.t < ec_2.t \wedge ec_1.t > ec_2.t$
during	$ec_1.t > ec_2.t \wedge ec_1.t < ec_2.t$
starts	$ec_1.t = ec_2.t \wedge ec_1.t < ec_2.t$
started by	$ec_1.t = ec_2.t \wedge ec_1.t > ec_2.t$
equals	$ec_1.t = ec_2.t \wedge ec_1.t = ec_2.t$

Replacing SEQ and OVERLAPS with a more precise operator allows the framework to have more precise knowledge of the structure of an event. Thus, the historical event query statements (see Chapter 5) of the *CEMiD* language can report the structure of an event with a more precised detail.

6.2 GST-CEMiD: A Pipeline-Based Complex Event Processing En-gine for CEMiD Framework

As we showed in the previous section, event detection algorithms of the *CEMiD* framework are mostly based on a pipeline-based processing technique in which all the raw sensor data stream are continuously fed into processing queues of the framework, and also continuously sent to the event detection process queues. To implement this kind of mechanism practically, the complex event processing engine of the framework is needed to construct an event processing pipeline in response to each event statement provided by users. In *CEMiD* framework, the engine for generating an event processing pipeline is called the *GST-CEMiD*. The pipeline generated by the *GST-CEMiD* consists of three main interconnected components: *Input Handling*, *Event Detection*, and *Output Sink*. They are depicted in Figure 6.2.

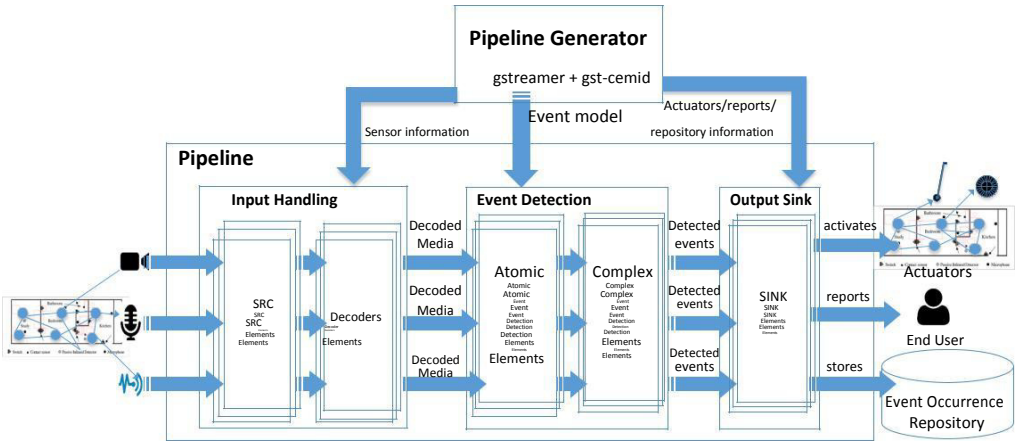


Figure 6.2 – The structure of an event processing pipeline

Details of each component in Figure 6.2 are given below:

- *Input Handling:* This component is responsible for receiving raw sensor data streams from sensors and decoding them. Thus, this component is in turn divided into two sub-components: *SRC Elements* and *Decoders* (see Figure 6.2). The raw sensor data reading is handled by the *SRC Elements*, while the decoding task is handled by the *Decoders*. *SRC Elements* consist of multiple sub-processes (one per sensor) for reading raw data stream. In general, one SRC element process is created in response to one incoming input raw sensor data stream;
- *Decoders:* consist of multiple sub-processes for decoding raw data from sensors.
- *Event Detection:* It is responsible for processing atomic and complex events. One event detection process is created in response to each atomic event condition and event statement, as given in the predefined *CEMiD* language statements provided by users.
- *Output Sink:* This component is responsible for handling the output of the *Event Detection* com-ponent. The output sink consists of multiple sub-processes where each process is referred to as a *SINK Element*. A sink element takes an event detection result of an event detection process and reacts according to a predefined behavior in the predefined *CEMiD* language statements. Whenever an event is detected, a sink element can activate an actuator, report the result to users, or store the detection result.

To demonstrate the feasibility of the implementation of our *CEMiD* framework, we develop *GST-CEMID* relying on an extension of GStreamer[1] (a framework for processing multimedia data in Linux-based operating systems), as a basis for generating the processing pipeline. We extended the GStreamer to support sensor readings and event processing. Next sub-sections describe the original elements of GStreamer framework and the newly proposed plugins.

6.2.1 GStreamer Framework

The *GStreamer* framework is a pipelining based framework that is used by every Linux-based operating system for handling multimedia data. In short, a GStreamer pipeline consists of multiple sub-processes, where each process is referred to as an *element* of the pipeline. The elements can be categorized into three main groups: *source* elements, *filter* elements, and *sink* elements. *Source* elements are used for handling the inputs of the pipeline. Examples of source elements include a *filesrc* element, for handling inputs from files or a *tcpclientsrc* for handling inputs from a TCP-based protocol. *Filter* elements are a group of elements that can be used for processing the multimedia data. More than 100 filter elements are proposed in GStreamer. A notable filter element is the *decodebin* element, which can automatically detect a type of an incoming multimedia data stream, picking an appropriate decoding algorithm, and decode the data in one single element. Finally, *Sink* elements of the GStreamer are used for handling the outputs of the filter elements. In general, *sink* elements of the framework can be used for either producing an output to a file, displaying on screen (in case of a video or an image stream), or playing the output to the system speakers (in case of an audio stream).

An example of a GStreamer pipeline for decoding an MP4 video file from a filename `test.mp4` and displaying the content on the screen, is given as follows.

```
filesrc location="test.mp4" ! decodebin ! xvimagesink
```

The elements in this pipeline are `filesrc`, `decodebin`, and `xvimagesink`. The `filesrc` element denotes the input file `test.mp4`, into the pipeline. The *decodebin* element is connected to the *filesrc* for detecting the multimedia type and decoding the content. The *xvimagesink* is an element for displaying the result of the video decoding on screen.

The complex event processing engine of the *CEMiD* framework generates one GStreamer pipeline in response to each atomic event condition and event statement as given by the users in *CEMiD* language. However, the GStreamer framework does not contain any of the functionalities for handling sensor input data stream, detecting an atomic event, detecting a complex event, or handling an event detection result. Hence, we propose a new plugin for GStreamer framework, called *gst-cemid*, for proposing a set of new elements with all the necessary functionalities for processing events in multimedia sensor networks. The detail of the *gst-cemid* plugin is given in the following sub-section.

6.2.2 gst-cemid plugin

gst-cemid is a plugin for GStreamer framework to support the complex event modeling and detecting fea-tures. Four new elements are proposed: `cemidsensorsrc`, `cemidatomiccond`, `cemidoperator`, and `cemideventsink`:

- `cemidsensorsrc`: is used for handling an incoming raw sensor data stream. In general, one `cemidsensorsrc` element is needed to be created in response to one sensor that is defined within the user defined *CEMiD* statements . The `cemidsensorsrc` uses the HTTP protocol for the communication between a sensor and a `cemidsensorsrc` element;

[1] http://gstreamer.net/

- `cemidatomiccond`: is used for detecting an atomic event. One `cemidatomiccond` is created in response to one *atomic event condition* within the *CEMiD* statement provided by users;

- `cemidoperator`: is used for combining outputs from multiple atomic event conditions into a complex event. One `cemidoperator` element is created in response to one event operator within the *CEMiD* statements. This element can work as either an OVERLAPS or a SEQ operator;

- `cemideventsink`: is used for handling the output of the pipeline. It is capable of reporting event detection results to a user, storing event detection results within an event repository, and activating an actuator.

Our *GST-CEMiD* engine uses the *gst-cemid* plugin elements along with the elements of the GStreamer framework to construct an appropriate event processing pipeline in response to an event statement as given in the user-defined *CEMiD* language statement. This means that one pipeline is generated in response to one event statement within the *CEMiD* language statement. Figure 6.3 shows an example of a generated pipeline by our *GST-CEMiD* engine.

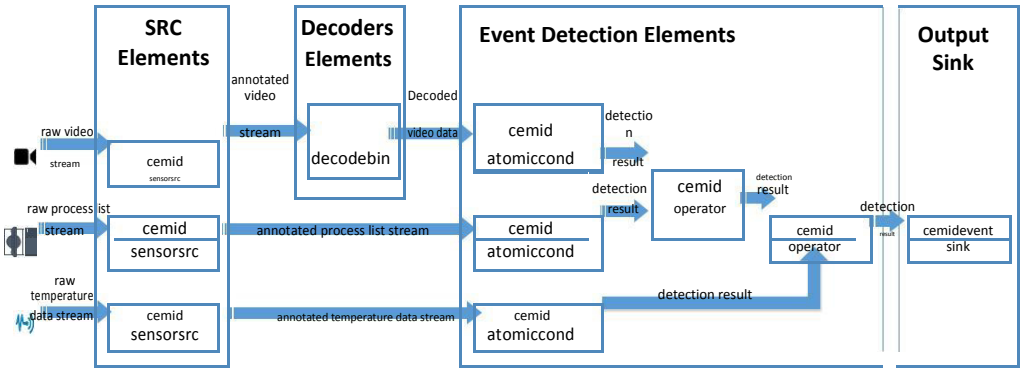


Figure 6.3 – An example of a GST-CEMID processing pipeline

The example in Figure 6.3 depicts a pipeline for an event statement `(face_detected OVERLAPS working) OVERLAPS high_temperature`. This reflects the `Overpowered Heater` event as given in Table 1.2. This statement uses one video camera, one computer, and one temperature sensor as inputs. Three `cemidsensorsrc` elements are created for handling the input stream from each of these sensors. A `decodebin` element is also created and connected to the `cemidsensorsrc` element of the video camera stream. The temperature sensor and the computer are considered as scalar sensors; thus their input streams do not require any multimedia decoding. Next, three `cemidatomiccond` elements are created. One is configured for detecting a face within a video stream. Another is configured for detecting whether a list of processes on a computer contains at least one productivity application (e.g., Word, Powerpoint). The last one is configured with a condition for detecting a high-temperature event. Two `cemidoperator` elements are created for connecting `cemidatomiccond` elements according to the statement. In this case, *GST-CEMID* framework connects the `cemidatomiccond` ele-ments of the the video camera and the computer to model `(face_detected OVERLAPS working)`. The result of this operator is connected to another `cemidoperator` to model `(face_detected OVERLAPS working) OVERLAPS high_temperature`. The final detection results are sent to the `cemideventsink` to report to the user and stored into the repository.

6.3 Experiments

Our framework is validated by means of experiments. The objective of our experiments was to validate the performance of the framework whether it can support a near real-time event processing in multimedia sensor networks, and a near real-time event processing under a high workload case (i.e., a case with a large number of sensors). The metrics for validating the framework performance are (i) *detection latency* and (ii)*throughput*. The detection latency is the gap between the time that an event is detected by the framework and the time that such an event were actually happened. The throughput is calculated by measuring a number of events that are detected by the framework in one minute.

Our experiment is carried out under the smart office application domain (see Section 1.3). In order to avoid the overhead produced from hardware and network communication, we chose to conduct our experiments by means of simulations. To do so, we developed a simulation script which can simulate streams of data from a sensor network. The sensor network that we simulate is the smart office sensor network as given in Figure 1.2. However, instead of simulating the whole room as described in the figure, the simulation script can be configured such that, the number of temperature sensors, light sensors, computer sensors and, video cameras within the room can be changed. This allows to simulate a high workload case by increasing number of sensors within the simulation. The characteristic of each type of sensors within the simulation is as follows:

- Video Camera: A simulated video camera sensor produces a random image frame at a configurable sampling frequency. Possible images are either an image of an office with no people inside or an image with at least one people inside;
- Temperature sensor, Light sensor: Both of these sensors produce a reading by randomizing a value based on a standard deviation;
- Computer Sensor: A simulated computer sensor produces a random list of processes that are currently running on the computer. The possible processes are *Word, Powerpoint* and *Skype*. The list of processes can also be empty to reflect that there is no process that is currently running.

Within the simulation, we can freely configure the sampling frequency of each sensor. For example, a simulation script can configure a temperature sensor to produce a reading for every 1 second and configure a video camera to produce an image frame for every 100 milliseconds.

According to the design of our framework, we identified three main factors that might affect the performance: (i) number of sensors, (ii) sampling frequency and, (iii) number of operators within an event statement. We conducted three sets of experiments to verify the effect of varying each of these factors. The specification of the computer that is used in our experiments is: Ubuntu 16.04 operating system, CPU Intel Core I7 2.4 GHz and 8GB of RAM. The experiment setups and results are described in sub-sections as follows.

6.3.1 Impact of the number of sensors

To measure the impact of the number of sensors on the performance of *GST-CEMiD*, we followed the following steps:

1. Initiate 100 simulation instances, each instance is configured to have one video camera and one temperature sensor, representing a scenario with 200 sensors. Each sensor is configured to produce a sensor reading with a sampling rate of 500 milliseconds per reading.
2. Each simulation instance is configured to detect the following event statements:
    - `OVERLAPS(face_detected, high_temp)`

- `SEQ(face_detected, high_temp, 2)`

3. Run the simulation for 10 minutes, measure the event average detection latency and throughput per minute.
4. Redo all steps by changing the number of simulation instances to 200, 300, 400, and 500, which means scenarios with 400, 600, 800, and 1000 sensors, respectively.

Figure 6.4 and Figure 6.5 show that temporal operators SEQ and OVERLAPS have different performance since they have different processing algorithms which may affect the latency and throughput in a different manner. However, their trends are similar.

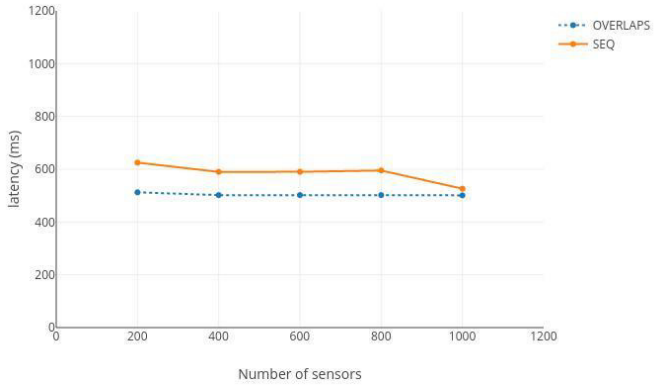


Figure 6.4 – Impact of number of sensors on detection latency

Results suggest that the detection latency is approximately 501 - 510 ms in all cases for OVERLAPS operator and approximately 520 - 620 ms for SEQ operator statements. This shows that increasing the number of sensors has very little effect on the detection latency as Figure 6.4 reveals that the detection latency tends to be constant regardless the number of sensors used.

Figure 6.5 shows the effect of increasing the number of sensors against throughput (in the unit of numbers of detected events per minute). The result shows that the framework can support up to 11,631 events per minute in a statement with OVERLAPS operator and 10,034 events per minute in a SEQ operator case. Thus, this shows that our framework can be used in a high workload case.

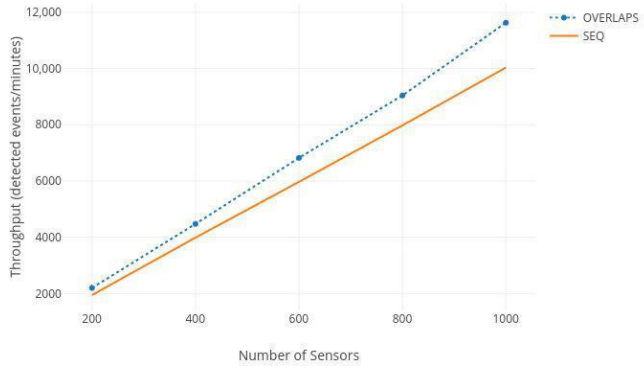


Figure 6.5 – Impact of number of sensors on throughput

6.3.2 Impact of sensor sampling frequency

The variation of the sampling frequency of each sensor means more numbers of sensor readings produced per second. This may have a direct effect on event detection latency and throughput because the number of incoming sensor readings per minute could be different. In order to verify this, we conducted this set of experiments with the same experiment instruction as in previous experiments, except varying the sampling frequency from 500 ms as in previous experiments to 200 ms, 400 ms, 600 ms, 800 ms and 1000 ms. The number of sensors used for each experiment is also fixed to 1000 sensors (i.e., 500 simulation instances as in the previous experiment).

Figure 6.6 shows that the effect of increasing the sampling frequency on detection latency tends to be linear. The SEQ operator has more latency for approximately 10-30 ms in all cases. The figure also reveals that the detection latency for all cases tends to be almost equal to the sampling frequency.

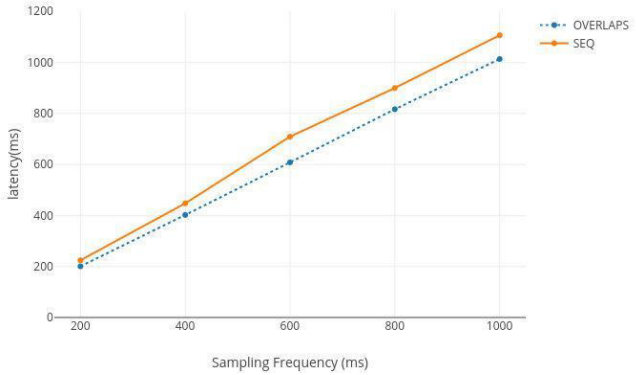


Figure 6.6 – Impact of of sensor sampling frequency on detection latency

Figure 6.7 shows that the effect of increasing sampling frequency on throughput. The result reveals that the throughput per minute decreases in a logarithmic pattern as the sensor sampling frequency is increased.

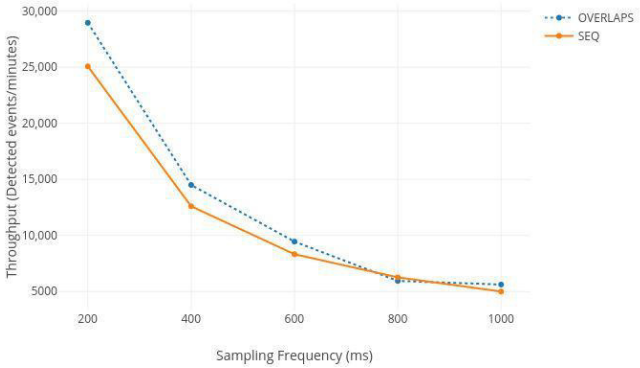


Figure 6.7 – Impact of sensor sampling frequency on throughput

6.3.3 Impact of number of operators

To verify the effect of varying the number of event operators against the detection latency and throughput of *GST-CEMID*, we performed the following steps:

1. Initiate 100 simulation instances, each instance is configured to have one video camera and one temperature sensor, representing a scenario with 200 sensors. Each sensor is configured to produce a sensor reading with a sampling rate of 500 milliseconds per reading.
2. Each simulation instance is configured to detect the following event statements:
   - `CREATE EVENT exp:o1`
     `STATEMENT OVERLAPS(cond:face,cond:high_light)`
   - `CREATE EVENT exp:o2`
     `STATEMENT OVERLAPS(`
     `OVERLAPS(cond:face,cond:high_light),`
     `cond:high_temp)`
   - `CREATE EVENT exp:o3`
     `STATEMENT OVERLAPS( OVERLAPS( OVERLAPS(cond:face,cond:high_light),`
     `cond:high_temp), cond:working)`
   - `CREATE EVENT exp:o4`
     `STATEMENT OVERLAPS( OVERLAPS( OVERLAPS( OVERLAPS(cond:face,cond:high_light),`
     `cond:high_temp), cond:working),`
     `cond:videoconf)`
3. Run the simulation for 10 minutes, measure the average detection latency of every detected event occurrences.
4. Restart steps 2-4 and changes every operator in each statement of step 2 from OVERLAPS to SEQ.

The event statements used in this experiment were designed solely for testing the performance. Hence, these statements may not totally reflect the real world use case. The experiment results are given in Figure 6.8 and Figure 6.9.

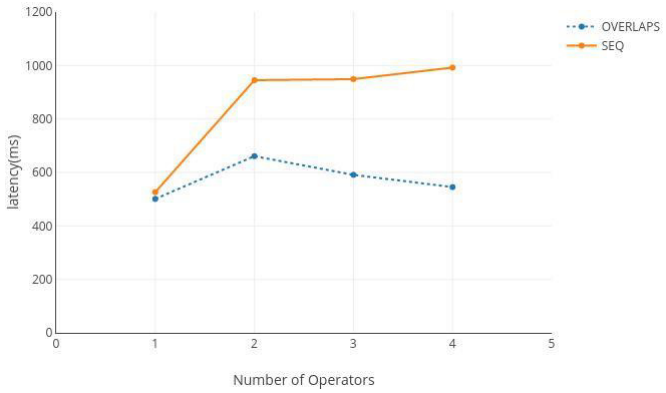


Figure 6.8 – Impact of number of operators on detection latency

Figure 6.8 depicts the result of the effect of varying the number of event operators against detection latency. For OVERLAPS operator case, varying the number of operators show only minor effect. However, the difference in detection latency for SEQ operator changes when using one SEQ operator and two SEQ

operators. The detection latency for one operator case is approximately 550 ms while two operators, three operators, and four operators case show the latency for almost 1000 ms. This may be caused by the queue congestion during the SEQ operator processing within the pipeline.

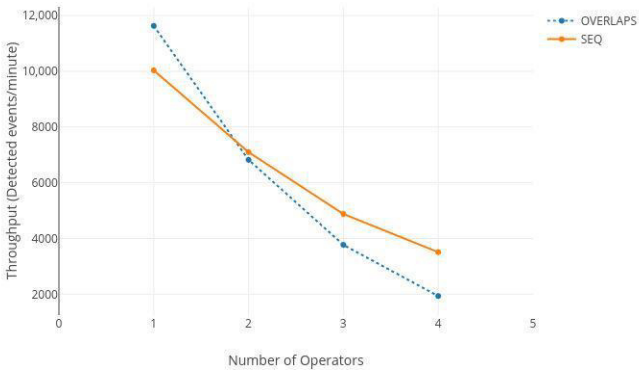


Figure 6.9 – Impact of number of operators on throughput

Figure 6.9 shows the effect of increasing the number of operators against throughput. The result reveals that increasing the number of operators decreases the overall throughput. As expected, the pattern of decreasing tends to be linear. The pattern may be different if the number of operators increases further than four operators. However, an event statement which requires more than five operators is rare in practice according to our experience.

6.3.4 Discussion

So far, we conducted experiments to verify the performance of our framework by means of simulation. The experiments that are conducted are designed to study the effect of increasing number of sensors, varying sensor sampling frequency and the variation of the number of operators in a statement. The result shows that the detection latency is less than 1000 ms in almost every cases. This shows that the performance of our framework is sufficient to be used in a near real-time scenario. Commercial grade event detection engines may claim for a better performance, such as ESPER[2] library, whose reported latency lies in three microseconds average. However, we can support event processing in MSNs, while ESPER cannot.

Due to the fact that our experiment is conducted under the simulated environment, the overhead on network communication and multimedia data were not taken into account within these results. However, both of the latency produced by these effects are ought to be proportional. Hence, the pattern of the result on the detection latency and throughout is ought to similar to our simulation in the practical case.

[2] http://www.espertech.com/esper/

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Multimedia sensor networks have received an extensive focus from both academic and commercial sector in recent years, in the context of everyday live applications, such as monitoring, tracking, and facilitating daily life activities. They have been used in advanced applications, such as smart buildings, smart homes, and smart cities. However, despite their popularity, processing complex events in multimedia sensor networks is an open challenge to be overcome. The major cause is the fact that multimedia sensor networks infrastructure, the data types that they can produce, and the different kinds of events that users would like to detect can be very diverse. This diversity makes it difficult to propose a single yet full-fledged framework which is able to process complex events in multimedia sensor networks in various domain

7.1.1 Analysis of existing solutions and related studies

On the course of our study, we survey the most recent and extensive used solutions that can be used for processing complex events in multimedia sensor networks. A suitable solution for processing complex events in multimedia sensor networks is needed to allow users to: (i) model their own sensor network infrastructure; (ii) work with multimedia sensors and multimedia data; and (iii) model their events that they would like to detect on their own. So far, we find that there is no existing solution that can comprehensively address all of these aspects. Studies that are related to sensor network infrastructure modeling can support well the infrastructure modeling. However, they cannot work with multimedia sensors and multimedia data. Studies that focus on multimedia processing can work with multimedia data well, but they cannot be used for modeling sensor networks and events within the sensor networks. Studies that focus on complex event processing are also mostly focused on business activities monitoring. Hence, they can neither be used for modeling sensor networks nor work with multimedia data.

7.1.2 Contributions

In this study, we propose the *CEMiD* framework to support Complex Events ModelIng and Detection in multimedia sensor network. The framework is a semantic-based framework which allows users to model their own multimedia sensor network infrastructure and events that they would like to detect from multimedia sensor network applications. The framework utilizes the ontology-based data model by adopting our newly proposed ontology so called the *Multimedia Semantic Sensor Network* Ontology (*MSSN-Onto*) for modeling all the necessary data. The *CEMiD* language is also proposed as a high-level

interface for users to model all the infrastructure and events. The framework uses all the predefined sensor network models and event models provided by users to preprocess all the incoming raw multimedia sensor data stream and translating them into events. All the processing algorithms are described in this study. The practical implementation of our *CEMiD* framework is based on the *GST-CEMiD* engine. It is developed by extending the GStreamer framework and adding all the *CEMiD* event processing functionality as a plugin so called the *gst-cemid*.

The core contributions of this dissertation are the *MSSN-Onto*, *CEMiD* language, and the *CEMiD* framework (i.e., GST-CEMiD engine). To validate the effectiveness of our contributions, we conduct the validation as follows:

- MSSN-Onto Validation: We validate the suitability of the *MSSN-Onto* itself whether it can be used for modeling multimedia sensor networks effectively. The ontology validation is done separately without adopting the full *CEMiD* framework in order to verify solely the capability of the ontology. The result suggests a positive result as the ontology can be effectively aligned with different kind of application domains. It can be used as a basis for creating an application for modeling and detecting events in the *AMI Smart Meeting Room* application effectively with an acceptable event querying performance and accuracy. The ontology itself is also currently being adopted in the HIT2GAP project which is an EU collaborative research project on a smart building application;
- CEMiD Language Validation: We validate the suitability of the CEMiD language by measuring its compactness comparing with the case of using purely SPARQL to manipulate the *MSSN-Onto* directly. The result shows that CEMiD language can reduce the query size by 52%. The JSON serialized version of the language can also helps to reduce the query size further to 69% comparing with using purely SPARQL;
- CEMiD Framework Validation: We develop the *CEMiD* framework on top of the *MSSN-Onto* and the *CEMiD* language. Through the course of the development, we propose a pipeline-based event detection engine so called the *GST-CEMiD*. We validate the performance of the *GST-CEMiD*, and the full framework itself under the *Smart Office* scenario. We choose to conduct the experiment under a simulated environment such that, we can freely adjust types of sensors and number of sensors within a simulated office to simulate different types of workload. The simulations were designed to study the effect of increasing number of sensors, varying sensor sampling frequency and varying the number of operators in a statement. The result showed that the detection latency is less than 1000 ms in almost every case. This shows that the performance of our framework is sufficient to be used in a near real-time scenario. Commercial grade event detection engines may claim for a better performance, such as ESPER[1] library, whose reported latency lies in three microseconds average. However, we can support event processing in multimedia sensor networks, while ESPER cannot.

7.2 Future Work

List of possible future directions of this dissertation that can be carried out can be categorized into two main directions, technical direction and scientific direction. They are given as follows

7.2.1 Technical directions for the possible future work

The list of possible future work in a technical direction are given as follows:

[1] //www.espertech.com/esper/

1. Developing a full version of the framework: The development of the *CEMiD* framework is currently at its prototyping state. Hence, the current version of the framework still lacks the programming interface or API which users or developers can use for applying the framework in their application domain yet. This is currently our on-going work;

2. Proposing more user-friendly interface for interacting with the framework: The current ver-sion of our framework offers users only the *CEMiD* language for interacting with the framework. In a practical case, some end-user might find that the *CEMiD* language is still too technical or too complicated for them to use. Hence, proposing a more user-friendly interface (e.g., a visual style language, simpler version of the *CEMiD* language) is one of the possible direction of the future work;

3. Adopting the framework in different application domains: In order to prove that our framework is generic in practical usage, we need to try to adopt the framework in different application domains. In this study, we based our scenario and validation experiments on the smart office scenario. In the future, we will try to apply the *CEMiD* framework in more application domains. Currently, our framework is adopted in HIT2GAP European H2020 project[2] to detect events related to energy. Results will be published in the future.

7.2.2 Scientific directions for the possible future work

The list of possible future work in a scientific direction are given as follows:

1. Time Synchronization: Current version of the framework assumes that the timestamp of all the data arrived from sensors are always synchronized. However, such an assumption may not be able to be used in every use-case. A better time synchronization algorithm from non-clock synchronization sensor networks may need to be proposed;

2. Optimizing Operator Processing: The near-real time operator of the current version of our framework processes events with latency of approximately 200 - 1000 ms. This may be acceptable in a near-real time use-case. However, in the case of a very large sensor network, operating latency can become slower due to resource limitation. Hence, the algorithm for processing operators may need to be optimize such that, it can process event faster, consume less processing power, and can be used in a larger sensor network size. The ideal goal is to reach the same performance level with ESPER library;

3. Sensor as a Service: Current version of the framework relies often on external functions modeled as an IRI (e.g., sensing method, atomic event detection function). In future study, these methods could be modeled as a service. Hence, sensors and methods that each sensor used can always be reused in different sensor networks as a service-like manner;

[2] http://www.hit2gap.eu

Bibliography

[1] Mike Botts et al. “OGC Sensor Web Enablement: Overview and High Level Architecture”. en. In: ed. by Silvia Nittel, Alexandros Labrinidis, and Anthony Stefanidis. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 175–190. ISBN: 978-3-540-79995-5, 978-3-540-79996-2.

[2] Shailendra Aswale and Vijay R. Ghorpad. “Wireless Multimedia Sensor Network: A Survey on Multimedia Sensors”. In: *International Conference on Recent Trends in Information, Telecommunication and Computing*. Aug. 2013, pp. 31–37.

[3] Taner Cevik, Alex Gunagwera, and Nazife Cevik. “A Survey of multimedia streaming in wireless sensor networks: progress, issues and design challenges”. In: *CoRR* abs/1512.03565 (2015). URL: http://arxiv.org/abs/1512.03565.

[4] John Soldatos et al. “Multimedia Search over Integrated Social and Sensor Networks”. In: *Proceedings of the 21st International Conference on World Wide Web*. WWW ’12 Companion. Lyon, France: ACM, 2012, pp. 283–286. ISBN: 978-1-4503-1230-1. DOI: 10.1145/2187980.2188029.

[5] Michael Compton et al. “The SSN ontology of the W3C semantic sensor network incubator group”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* Volume 17 (Dec. 2012), pp. 25–32. ISSN: 1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826812000571 (visited on 12/16/2013).

[6] J. Caleb Goodwin and David J. Russomanno. “Ontology integration within a service-oriented architecture for expert system applications using sensor networks”. In: *Expert Systems* 26.5 (2009), pp. 409–432. ISSN: 1468-0394. DOI: 10.1111/j.1468-0394.2009.00505.x.

[7] Chrisa Tsinaraki, Panagiotis Polydoros, and Stavros Christodoulakis. “Interoperability Support for Ontology-Based Video Retrieval Applications”. en. In: ed. by Peter Enser et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Jan. 2004, pp. 582–591. ISBN: 978-3-540-22539-3, 978-3-540-27814-6.

[8] Richard Arndt et al. “COMM: Designing a Well-Founded Multimedia Ontology for the Web”. In: ed. by Karl Aberer et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Jan. 2007, pp. 30–43. ISBN: 978-3-540-76297-3, 978-3-540-76298-0.

[9] François Bry and Michael Eckert. “Rule-Based Composite Event Queries: The Language XChangeEQ and Its Semantics”. In: *Web Reasoning and Rule Systems*. Lecture Notes in Computer Science 4524. Springer Berlin Heidelberg, June 7, 2007, pp. 16–30. ISBN: 9783540729815 9783540729822. URL: http://link.springer.com/chapter/10.1007/978-3-540-72982-2_2 (visited on 08/11/2016).

[10] Darko Anicic et al. “ETALIS: Rule Based Reasoning in Event Processing”. In: *Reasoning in Event-Based Distributed Systems*. Studies in Computational Intelligence 347. Springer Berlin Heidelberg, 2011, pp. 99–124. ISBN: 9783642197239 9783642197246. URL: http://link.springer.com/chapter/10.1007/978-3-642-19724-6_5 (visited on 08/11/2016).

[11] Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal. “Energy efficiency in wireless sensor networks: A top-down survey”. In: *Computer Networks* 67.Supplement C (2014), pp. 104–122. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2014.03.027. URL: http://www.sciencedirect.com/science/article/pii/S1389128614001418.

[12] Adolph Seema and Martin Reisslein. “Towards Efficient Wireless Video Sensor Networks: A Survey of Existing Node Architectures and Proposal for A Flexi-WVSNP Design”. In: *IEEE Communications Surveys Tutorials* 13.3 (Third 2011), pp. 462–486. ISSN: 1553-877X. DOI: 10. 1109/SURV.2011.102910.00098.

[13] Priyanka Rawat et al. “Wireless sensor networks: a survey on recent developments and potential synergies”. In: *The Journal of Supercomputing* 68.1 (Apr. 2014), pp. 1–48. ISSN: 1573-0484. DOI: 10.1007/s11227-013-1021-9. URL: https://doi.org/10.1007/s11227-013-1021-9.

[14] Jeff Adkins et al. *Event Processing Glossary Version 2.0*. Real Time Intelligence & Complex Event Processing. July 2011. URL: http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2/ (visited on 08/11/2016).

[15] Michael Eckert et al. “A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed”. In: *Reasoning in Event-Based Distributed Systems*. Studies in Computational Intelligence 347. Springer Berlin Heidelberg, 2011, pp. 47–70. ISBN: 9783642197239 9783642197246. URL: http : / / link . springer . com / chapter / 10 . 1007 / 978 - 3 - 642 - 19724 - 6 _ 3 (visited on 08/11/2016).

[16] Alan J Demers et al. “Cayuga: A General Purpose Event Monitoring System.” In: *CIDR*. Vol. 7. 2007, pp. 412–422.

[17] EsperTech. *Chapter 5. EPL Reference: Clauses*. URL: http : / / www . espertech . com / esper/release-5.3.0/esper-reference/html/epl_clauses.html (visited on 08/11/2016).

[18] Masoud Mansouri-Samani and Morris Sloman. “GEM: a generalized event monitoring language for distributed systems”. In: *Distributed Systems Engineering* 4.2 (1997), p. 96. URL: http : //stacks.iop.org/0967-1846/4/i=2/a=004.

[19] Chinnapong Angsuchotmetee and Richard Chbeir. “A Survey on Complex Event Definition Lan-guages in Multimedia Sensor Networks”. In: *Proceedings of the 8th International Conference on Management of Digital EcoSystems*. MEDES. Biarritz, France: ACM, 2016, pp. 99–108. ISBN: 978-1-4503-4267-4. DOI: 10.1145/3012071.3012098.

[20] Samuel Madden and Michael J. Franklin. “Fjording the stream: an architecture for queries over streaming sensor data”. In: *Proceedings 18th International Conference on Data Engineering*. 2002, pp. 555–566. DOI: 10.1109/ICDE.2002.994774.

[21] Phillip B. Gibbons et al. “IrisNet: an architecture for a worldwide sensor Web”. In: *IEEE Pervasive Computing* 2.4 (Oct. 2003), pp. 22–33. ISSN: 1536-1268. DOI: 10.1109/MPRV.2003.1251166.

[22] Seongwoon Jeong et al. “A distributed cloud-based cyberinfrastructure framework for integrated bridge monitoring”. In: *Proc SPIE* 10168 (2017). DOI: 10.1117/12.2270716.

[23] Alan G. Labouseur et al. “The G* graph database: efficiently managing large distributed dynamic graphs”. In: *Distributed and Parallel Databases* 33.4 (Dec. 2015), pp. 479–514. ISSN: 1573-7578. DOI: 10.1007/s10619-014-7140-3. URL: https://doi.org/10.1007/s10619-014-7140-3.

[24] Umit Isikdag and Morakot Pilouk. “Integration of Geo-Sensor Feeds and Event Consumer Ser-vices for Real-Time Representation of Iot Nodes”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (June 2016), pp. 267–274. DOI: 10.5194/isprs-archives-XLI-B4-267-2016.

[25] Cihan Küçükkeçeci and Adnan Yazıcı. “Big Data Model Simulation on a Graph Database for Surveillance in Wireless Multimedia Sensor Networks”. In: *Big Data Research* (2017). ISSN: 2214-5796. DOI: https://doi.org/10.1016/j.bdr.2017.09.003. URL: http://www.sciencedirect.com/science/article/pii/S2214579617300102.

[26] Tong Lee Chung et al. “Semantic Technology: Third Joint International Conference, JIST 2013, Seoul, South Korea, November 28–30, 2013, Revised Selected Papers”. In: ed. by Wooju Kim, Ying Ding, and Hong-Gee Kim. Cham: Springer International Publishing, 2014. Chap. Constructing City Ontology from Expert for Smart City Management, pp. 187–194. ISBN: 978-3-319-06826-8. DOI: 10.1007/978-3-319-06826-8_15.

[27] Thanos G. Stavropoulos et al. “BOnSAI: A Smart Building Ontology for Ambient Intelligence”. In: *Proceedings of the 2Nd International Conference on Web Intelligence, Mining and Semantics*. WIMS ’12. Craiova, Romania: ACM, 2012, 30:1–30:12. ISBN: 978-1-4503-0915-8. DOI: 10.1145/2254129.2254166. URL: http://doi.acm.org/10.1145/2254129.2254166.

[28] Khouloud Salameh et al. “Artificial Intelligence Applications and Innovations: 11th IFIP WG 12.5 International Conference, AIAI 2015, Bayonne, France, September 14-17, 2015, Proceedings”. In: ed. by Richard Chbeir et al. Cham: Springer International Publishing, 2015. Chap. A Generic Ontology-Based Information Model for Better Management of Microgrids, pp. 451–466. ISBN: 978-3-319-23868-5. DOI: 10.1007/978-3-319-23868-5_33. URL: http://dx.doi. org/10.1007/978-3-319-23868-5_33.

[29] Michael Compton. “The SSN ontology of the W3C semantic sensor network incubator group”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* Volume 17 (Dec. 2012), pp. 25–32. ISSN: 1570-8268. URL: http://www.sciencedirect.com/science/article/ pii/S1570826812000571 (visited on 12/16/2013).

[30] Carlos Rueda, Lui Bermudez, and Janet Fredericks. “The MMI Ontology Registry and Repository: A portal for Marine Metadata Interoperability”. In: *OCEANS 2009*. Oct. 2009, pp. 1–6.

[31] Claudia Villalonga et al. “Modeling of sensor data and context for the Real World Internet”. In: *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. Mar. 2010, pp. 1–6. DOI: 10.1109/PERCOMW.2010.5470594.

[32] Omer B. Sezer, Serdar Z. Can, and Erdogan Dogdu. “Development of a smart home ontology and the implementation of a semantic sensor network simulator: An Internet of Things approach”. In: *2015 International Conference on Collaboration Technologies and Systems (CTS)*. June 2015, pp. 12–18. DOI: 10.1109/CTS.2015.7210389.

[33] Qin Ni, Iván Pau, and Ana-Belén García-Hernando. “A foundational ontology-based model for human activity representation in smart homes”. In: *JAISE* 8 (2016), pp. 47–61. DOI: 10.3233/ AIS-150359.

[34] Tony C. T. Kuo and Arbee L. P. Chen. “Content-based query processing for video databases”. In: *IEEE Transactions on Multimedia* 2.1 (Mar. 2000), pp. 1–13. ISSN: 1520-9210.

[35] Chenglang Lu, Mingyong Liu, and Zongda Wu. “SVQL: A SQL Extended Query Language for Video Databases”. In: *International Journal of Database Theory and Application* 8.3 (), pp. 235–248.

[36] Thomas Kurz et al. “SPARQL-MM - Extending SPARQL to Media Fragments”. In: *The Semantic Web: ESWC 2014 Satellite Events*. Lecture Notes in Computer Science 8798. Springer International Publishing, May 25, 2014, pp. 236–240. ISBN: 9783319119540 9783319119557. URL: http://link.springer.com/chapter/10.1007/978-3-319-11955-7_26 (visited on 08/11/2016).

[37] Nicolas Durand et al. “Ontology-Based Object Recognition for Remote Sensing Image Interpreta-tion”. In: *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*. Vol. 1. Oct. 2007, pp. 472–479. DOI: 10.1109/ICTAI.2007.111.

[38] Ioannis Kompatsiaris Vasileios Mezaris and Michael G. Strintzis. “An ontology approach to object-based image retrieval”. In: *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*. Vol. 2. Sept. 2003, DOI: 10.1109/ICIP.2003.1246729.

[39] Jane Hunter. “Enhancing the semantic interoperability of multimedia through a core ontology”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.1 (Jan. 2003), pp. 49–58. ISSN: 1051-8215. DOI: 10.1109/TCSVT.2002.808088.

[40] Roberto García and Òscar Celma. “Semantic Integration and Retrieval of Multimedia Metadata”. In: *2nd European Workshop on the Integration of Knowledge, Semantic and Digital Media*. Galway, Ireland, 2005.

[41] Lee, W., Bürger, T., Sasaki, F., Malaisé, V., Stegmaier, F., Söderberg, J. *Ontology for Media Resource 1.0. Tech. rep., W3C Media Annotation Working Group (06 2009)*. URL: http://www.w3.org/ TR/mediaont-10/ (visited on 06/24/2015).

[42] Shih-Fu Chang, T. Sikora, and A. Purl. “Overview of the MPEG-7 standard”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 11.6 (June 2001), pp. 688–695. ISSN: 1051-8215. DOI: 10.1109/76.927421.

[43] James F. Allen. “Maintaining Knowledge About Temporal Intervals”. In: *Commun. ACM* 26.11 (Nov. 1983), pp. 832–843. ISSN: 0001-0782. DOI: 10 . 1145 / 182 . 358434. URL: http : //doi.acm.org/10.1145/182.358434.

[44] Sharma Chakravarthy and Deepak Mishra. “Snoop: An expressive event specification language for active databases”. In: *Data & Knowledge Engineering* 14.1 (Nov. 1, 1994), pp. 1–26. ISSN: 0169-023X. URL: http : / / www . sciencedirect . com / science / article / pii / 0169023X9490006X (visited on 08/11/2016).

[45] Roger S. Barga and Hillary Caituiro-Monge. “Event Correlation and Pattern Detection in CEDR”. In: *Current Trends in Database Technology – EDBT 2006: EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 919–930. ISBN: 978-3-540-46790-8.

[46] Daniel Gyllstrom et al. “SASE: Complex Event Processing over Streams”. In: *arXiv:cs/0612128* (Dec. 22, 2006). arXiv: cs/0612128. URL: http://arxiv.org/abs/cs/0612128 (visited on 08/11/2016).

[47] Arvind Arasu, Shivnath Babu, and Jennifer Widom. “CQL: A Language for Continuous Queries over Streams and Relations”. In: *Database Programming Languages*. Lecture Notes in Computer Science 2921. Springer Berlin Heidelberg, Sept. 6, 2003, pp. 1–19. ISBN: 9783540208969 9783540246077. URL: http://link.springer.com/chapter/10.1007/978-3-540-24607-7_1 (visited on 08/11/2016).

[48] *StreamSQL Guide*. URL: https://docs.tibco.com/pub/streambase_cep/7.3.10/doc/streamsql/index.html (visited on 08/11/2016).

[49] Jürgen Krämer and Bernhard Seeger. “PIPES: A Public Infrastructure for Processing and Exploring Streams”. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’04. Paris, France: ACM, 2004, pp. 925–926. ISBN: 1-58113-859-8.

[50] Davide Francesco Barbieri et al. “Querying RDF Streams with C-SPARQL”. In: *SIGMOD Rec.* 39.1 (Sept. 2010), pp. 20–26. ISSN: 0163-5808. (Visited on 08/11/2016).

[51] Matthew Perry, Prateek Jain, and Amit P. Sheth. “SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries”. In: *Geospatial Semantics and the Semantic Web: Foundations, Algorithms, and Applications*. Boston, MA: Springer US, 2011, pp. 61–86. ISBN: 978-1-4419-9446-2.

[52] Jean-Paul Calbimonte et al. “Enabling Ontology-Based Access to Streaming Data Sources”. In: *The Semantic Web – ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 96–111. ISBN: 978-3-642-17746-0.

[53] Darko Anicic et al. “EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning”. In: *Proceedings of the 20th International Conference on World Wide Web*. WWW ’11. New York, NY, USA: ACM, 2011, pp. 635–644. ISBN: 978-1-4503-0632-4. (Visited on 07/03/2015).

[54] Alex Kozlenkov et al. “Prova: Rule-Based Java Scripting for Distributed Web Applications: A Case Study in Bioinformatics”. In: *Current Trends in Database Technology – EDBT 2006: EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers*. Ed. by Torsten Grust et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 899–908. ISBN: 978-3-540-46790-8.

[55] Adrian Paschke et al. “Reaction RuleML 1.0: Standardized Semantic Reaction Rules”. In: *Rules on the Web: Research and Applications: 6th International Symposium, RuleML 2012, Montpellier, France, August 27-29, 2012. Proceedings*. Ed. by Antonis Bikakis and Adrian Giurca. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 100–119. ISBN: 978-3-642-32689-9.

[56] Pierre Wellner, Mike Flynn, and Maël Guillemot. “Browsing Recorded Meetings with Ferret”. In: ed. by Samy Bengio and Hervé Bourlard. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Jan. 2005, pp. 12–21. ISBN: 978-3-540-24509-4, 978-3-540-30568-2.

[57] Gianfranco E. Modoni, Marco Sacco, and Walter Terkaj. “A survey of RDF store solutions”. In: *2014 International Conference on Engineering, Technology and Innovation (ICE)*. June 2014, pp. 1–7. DOI: 10.1109/ICE.2014.6871541.

Resumé

Introduction

L'avancée spectaculaire de la technologie de capteur, des communications sans fil et de l'électronique numérique a favorisé le développement de réseaux de capteurs multimédias (sans fil) multifonctionnels (sans fil). Les réseaux de capteurs multimédias sont des réseaux de dispositifs interconnectés capables de récupérer de façon omniprésente du contenu multimédia, tels que des flux vidéo et audio, des images et des données de capteurs encore scalaires provenant de l'environnement. De nos jours, les réseaux de capteurs multimédias deviennent de plus en plus populaires et importants dans la vie de tous les jours, pour surveiller, suivre et détecter des événements selon différents scénarios (e.g., maisons intelligentes, immeubles intelligents, villes intelligentes).

La popularité des réseaux de capteurs (également des réseaux de capteurs multimédias) a conduit à l'énorme quantité de données produites chaque jour par des capteurs. Ceci conduit à la difficulté de maintenir efficacement les données provenant des réseaux de capteurs, de sorte que les événements puissent être détectés efficacement parmi les données collectées. L'existence de dispositifs différents et intelligents dans le réseau rend ce défi plus difficile à surmonter en raison de la diversité des types de capteurs, d'une grande variété de formats de sortie de capteurs et de la difficulté à modéliser les données multimédias. un évènement. Pourtant, les approches pour la modélisation des réseaux de capteurs, la modélisation des données collectées et l'assistance aux utilisateurs pour la modélisation et la détection des événements sont à la traîne.

L'adoption de réseaux de capteurs multimédias ou hybrides (réseau de capteurs composé de capteurs scalaires et multimédias) sur l'utilisation de capteurs uniquement scalaires permet à une application de détecter des événements plus complexes qui ne peuvent être détectés sans utiliser le multimédia. Les données. Par exemple, un réseau de capteurs de vitesse sur le bord de la route permet uniquement la détection d'un événement de vitesse de conduite excessive. Cependant, enrichir le réseau avec des caméras de surveillance permet de détecter des événements plus complexes, tels que les événements Traffic Jam ou les accidents de voiture. L'analyse des images de caméra de surveillance et des capteurs de vitesse peut également être utilisée pour détecter le numéro de plaque de la voiture qui roule plus vite qu'une limite de vitesse. Même si les réseaux de capteurs multimédias sont largement utilisés de nos jours, les problèmes concernant la difficulté de traitement des événements persistent. Les principales raisons peuvent être brièvement développées comme suit.

- L'absence d'un modèle de données pour la modélisation des données collectées à partir des réseaux de capteurs multimédias, ce qui peut faciliter la modélisation et la détection des événements;
- L'absence d'approche pour la modélisation et la détection d'événements dans les réseaux de capteurs multimédias;
- L'absence d'interface, d'outil ou de langage pour accompagner les utilisateurs dans la modélisation des événements complexes qui les intéressent.

Dans un cas d'utilisation idéal d'une application de réseau de capteurs multimédias, les utilisateurs doivent pouvoir modéliser l'infrastructure de réseau de capteurs et les événements qu'ils souhaitent détecter par eux-mêmes sans avoir recours à un codage matériel ou une réimplémentation par les développeurs. Cependant, l'absence d'un modèle de données adapté, un langage de modélisation des événements et une approche de traitement des événements complexes dans les réseaux de capteurs multimédias forcent les développeurs à coder en dur tous les modèles de données et le processus de détection. Pour surmonter ce problème, un modèle de données approprié pour la modélisation de réseaux de capteurs multimédias, un langage qui aide les utilisateurs à modéliser leurs propres besoins, et une approche pour traiter les événements qui sont désignés pour des réseaux de capteurs multimédias doivent être proposés.

CEMiD:Un framework de traitement semantic d'événement dans les reseaux des capteurs multimedias

Répondant aux objectifs proposés, cette étude propose un framework semantique appelé CEMiD (Complex Event Modeling and Detection Framework). L'objectif principal du framework CEMiD est de permettre aux utilisateurs de différents rôles de définir librement des événements en fonction de leurs besoins, tout en conservant une architecture de bas niveau identique quel que soit le rôle ou le domaine d'application des utilisateurs. L'architecture de CEMiD est donnée dans la figure 7.1.

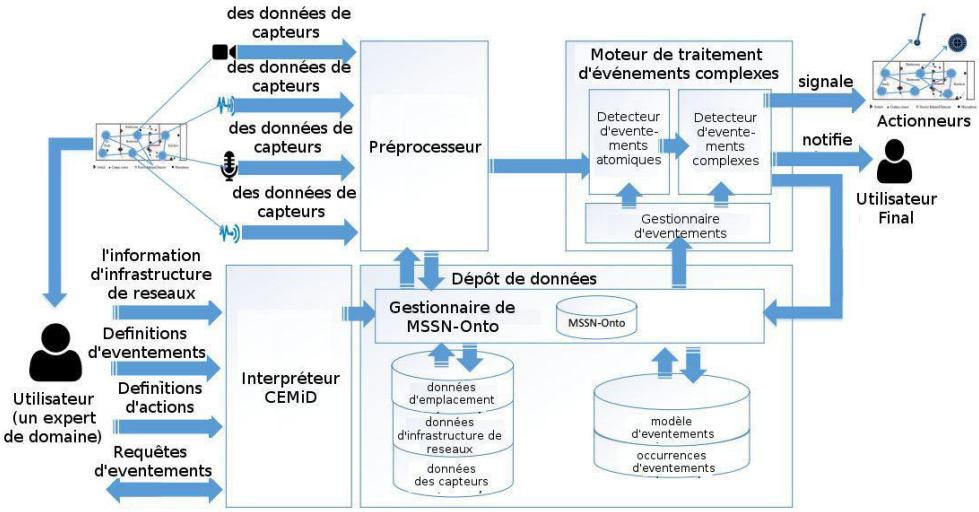


Figure 7.1 – L'architecture de framework *CEMiD*

L'architecture du framework CEMiD est décrite en quatre modules principaux: (i) Dépôt de données semantique (ii) CEMiD Intrepreter (iii) Préprocesseur de données, et (iv) Complexe de traitement des événements complexes. Leurs brefs détails sont décrits comme suit:

- Dépôt de données semantique: Le dépôt de données semantique de framework CEMiD composant dans lequel toutes les données sont stockées. Les cinq types de données sont stockés dans leurs dépôt respectifs qui sont (i) emplacements de reseaux des capteurs, (ii) infrastructure de reseaux des captures, (iii) données des capteurs, (iv) modèles des événements, et (v) occurrence des événements. Toutes les données sont modèlisées conformément à notre nouvelle ontologie proposée pour la modélisation de réseaux de capteurs multimédias, appelée *Multimedia Semantic Sensor Network (MSSN-Onto)*;

- Interprèteur CEMiD: Interprèteur CEMiD sert d'interface de haut niveau pour aider les utilisateurs à interagir avec le framework. Notre cadre propose un langage appelé langage CEMiD pour modéliser leurs besoins;

- Pré-processeur: Ce composant est responsable du pré-traitement et de l'annotation des flux de données de capteur entrants en utilisant MSSN-Onto. Ce module fonctionne automatiquement en réponse à chaque lecture de capteur brut entrant;

- Moteur de traitement des événements complexes Le moteur de traitement des événements com-plexes est responsable de la détection des événements tels que définis par les utilisateurs via le langage CEMiD. Le processus de détection est lancé automatiquement en réponse à chaque lecture de capteur. Le mécanisme interne de gestion de tous les pipelines de détection d'événements dans le framework CEMiD est appelé GST-CEMiD.

Selon l'architecture décrite, on peut voir que le framework est développé au-dessus de trois éléments de base qui sont MSSN-Onto, CEMiD Language, Complex Event Processing Engine. Ces trois éléments de base sont également la principale contribution de ces études. Leurs brefs détails sont décrits comme suit.

- Ontologie de réseau de capteurs sémantiques multimédias (MSSN-Onto): Cette ontologie a été récemment proposée dans cette étude pour être utilisée comme une ontologie pour la modélisation de réseaux de capteurs multimédias. Cette contribution aide à surmonter le défi concernant l'absence d'un modèle de données approprié pour la modélisation de réseaux de capteurs multimédias. Il permet de modéliser l'infrastructure des réseaux de capteurs multimédias et toutes les données collectées de telle sorte que toutes les données modélisées puissent être connectées à plusieurs domaines d'application;

- Langage CEMiD le langage CEMiD est un langage destiné aux utilisateurs pour modéliser des événements complexes dans des réseaux de capteurs multimédias. Ce langage aide les utilisateurs à modéliser leurs propres événements sans compter sur le codage ou la réimplémentation du développeur. La base du langage CEMiD est prise à partir de plusieurs langages de traitement d'événements complexes dans un style de syntaxe différent qui inclut des langages de style ECA (Event-Condition-Action) et des langages de style SQL/SPARQL. Le langage CEMiD est également disponible dans un format de sérialisation JSON;

- Moteur GST-CEMiD pour CEMiD Framework: GST-CEMiD fonctionne comme un prépro-cesseur de données et un moteur de traitement d'événements complexes pour le framework CEMiD. Le moteur est construit sur le MSSN-Onto et un framework de traitement multimédia existant appelé GStreamer. Le moteur détecte les événements en créant des pipelines de détection d'événements pour gérer tous les prétraitements de données, la détection d'événements atomiques et la détection d'événements complexes en temps quasi réel.

Validations & Résultats

Le framework CEMiD est validé au moyen d'expérimentation. La première étape de la validation consiste à valider la capacité de MSSN-Onto en tant qu'ontologie fondamentale pour la modélisation de réseaux de capteurs multimédias. La validation montre un résultat satisfaisant en terme de généralité et d'expressivité de l'ontologie. La généralité de l'ontologie est validée par la tentative d'alignement de l'ontologie dans différents domaines d'application. Jusqu'à présent, l'ontologie est alignée avec une ontologie de bâtimente intelligente et une ontologie de salle de réunion intelligente. L'expressivité de l'ontologie est validée en

appliquant l'ontologie dans une application de salle de réunion intelligente. Les résultats montrent que l'ontologie peut être utilisée efficacement pour modéliser des événements qui peuvent être trouvés dans un scénario de salle de réunion intelligent tel qu'un discussion, une présentation ou un changement de présentateurs.

La deuxième étape de la validation est la validation du framework CEMiD complet lui-même. Nous validons le framework complet dans un scénario de bureau intelligent simulé. L'objectif de la validation est de valider si le framework peut détecter des événements de manière quasi-temps réel dans un cas de charge de travail élevée (cas avec un grand nombre de capteurs). Nous avons choisi de conduire au moyen d'une simulation afin que le nombre de capteurs dans la simulation puisse être modifié librement. Les résultats de la simulation montrent que le cadre peut encore détecter les événements de manière quasi-instantanée dans notre cas extrême sélectionné (un cas avec 500 caméras vidéo et 500 capteurs de température).

Conclusion et Perspective

Cette étude propose un cadre de modélisation et de détection des événements dans les réseaux de capteurs multimédias, appelé CEMiD. Le framework est construit sur trois éléments principaux qui sont (i) le MSSN-Onto, (ii) le langage CEMiD et (iii) le moteur GST-CEMiD. Le résultat montre que notre infrastructure peut être utilisée pour détecter des événements dans un scénario de bureau virtuel simulé avec une charge de travail élevée (grand nombre de capteurs).

Deux directions possibles des travaux futurs peuvent être réalisées: (i) la direction technique et (ii) la direction scientifique. Ils sont donnés comme suit.

- Direction technique
  - Développer une version complète du framework: Le développement du framework est actuellement à son état de prototypage. Par conséquent, la version actuelle du framework n'a toujours pas l'interface de programmation ou API que les utilisateurs ou les développeurs peuvent utiliser pour appliquer le framework dans leur domaine d'application pour le moment. C'est actuellement notre travail en cours;
  - Proposer une interface plus conviviale pour interagir avec le framework: La version actuelle de notre framework n'offre aux utilisateurs que le langage CEMiD pour interagir avec le framework. Dans un cas pratique, certains utilisateurs finaux pourraient trouver que le langage CEMiD est encore trop technique ou trop compliqué pour être utilisé. Par conséquent, en proposant une interface plus conviviale (par exemple, un langage de style visuel, une version plus simple de la langue est l'une des directions possibles du travail futur);
  - Adopter le framework dans différents domaines d'application: Afin de prouver que notre framework est générique dans une utilisation pratique, nous devons essayer d'adopter le framework dans différents domaines d'application. Dans cette étude, nous avons basé nos expériences de scénario et de validation sur le scénario de bureau intelligent. À l'avenir, nous essaierons d'appliquer le framework CEMiD dans plusieurs domaines d'application. Actuellement, notre cadre est adopté dans le projet HIT2GAP européen H2020 pour détecter les événements liés à l'énergie. Les résultats seront publiés dans le futur.
- Direction scientifique
  - Synchronisation temporelle: La version actuelle de l'infrastructure suppose que l'horodatage de toutes les données provenant des capteurs est toujours synchronisé. Cependant, une telle hypothèse peut ne pas pouvoir être utilisée dans tous les cas d'utilisation. Un meilleur algorithme de synchronisation temporelle à partir de réseaux de capteurs de synchronisation non-horloge peut devoir être proposé;

– Optimisation du traitement des opérateurs: L'opérateur en temps quasi-réel de la version actuelle de notre framework traite les événements avec une latence d'environ 200 à 1000 ms. Cela peut être acceptable dans un cas d'utilisation en temps quasi réel. Cependant, dans le cas d'un très grand réseau de capteurs, la latence de fonctionnement peut devenir plus lente en raison de la limitation des ressources. Par conséquent, l'algorithme de traitement des opérateurs peut devoir être optimisé de telle sorte qu'il puisse traiter les événements plus rapidement, consommer moins de puissance de traitement et être utilisé dans une taille de réseau de capteurs plus grande. L'objectif idéal est d'atteindre le même niveau de performance avec la bibliothèque ESPER qui peut détecter les événements avec latence au niveau microsecondes;

– Capteur en tant que service: La version actuelle du cadre repose souvent sur des fonctions externes modélisées comme un IRI (par exemple, méthode de détection, fonction de détection d'événement atomique). Dans une étude future, ces méthodes pourraient être modélisées en tant que service. Par conséquent, les capteurs et les méthodes utilisés par chaque capteur peuvent toujours être réutilisés dans différents réseaux de capteurs en tant que service.