Commonwealth Scientific and Industrial Research Organisation (CSIRO)
Computational Informatics Division
Erasmus Mundus Master's Programme in Pervasive Computing & Communications for sustainable Development PERCCOM

Julien Dhallenne

RCOS: Real Time Context Sharing Across A Fleet Of Smart Mobile Devices

2016

Supervisor(s): *Dr. Prem Prakash Jayaraman (CSIRO)*
Pr. Arkady Zaslavsky (CSIRO)

Examiners: *Pr. Eric Rondeau* (University of Lorraine)
Pr. Jari Porras (Lappeenranta University of Technology)
Pr. Karl Andersson (Luleå University of Technology)

This thesis is prepared as part of an European Erasmus Mundus programme PERCCOM - Pervasive Computing & COMmunications for sustainable development.

This thesis has been accepted by partner institutions of the consortium (cf. UDL-DAJ, n°1524, 2012 PERCCOM agreement).

Successful defense of this thesis is obligatory for graduation with the following national diplomas:

- Master in Master in Complex Systems Engineering (University of Lorraine)
- Master of Science in Technology (Lappeenranta University of Technology
- Master in Pervasive Computing and Computers for sustainable development (Luleå University of
  Technology)

ABSTRACT

Author's name: Julien Dhallenne

Title of thesis: RCOS: Real Time Context Sharing Across A Fleet Of Smart Mobile Devices

Universities: Lappeenranta University of Technology

Name of the school: School of Business and Management (LUT)

Name of the degree programme: Computer Science (LUT)

Name of the Master's degree programme: Erasmus Mundus Master's Programme in Pervasive Computing & Communications for Sustainable Development PERCCOM

Master's Thesis, 2016, 57 pages, 18 figures, 2 tables, 1 appendix

Examiners: Pr. Eric Rondeau (University of Lorraine), Pr. Jari Porras (Lappeenranta University of Technology), Pr. Karl Andersson (Luleå University of Technology)

Keywords: publish/subscribe; context-awareness; ontologies; knowledge representation; semantic web

Today, biodiversity is endangered by the currently applied intensive farming methods imposed on food producers by intermediate actors (e.g.: retailers). The lack of a direct communication technology between the food producer and the consumer creates dependency on the intermediate actors for both producers and the consumers. A tool allowing producers to directly and efficiently market produce that meets customer demands could greatly reduce the dependency enforced by intermediate actors. To this end, in this thesis, we propose, develop, implement and validate a Real Time Context Sharing (RCOS) system. RCOS takes advantage of the widely used publish/subscribe paradigm to exchange messages between producers and consumers, directly, according to their interest and context. Current systems follow topic-based model or a content-based model. With RCOS, we propose a context-awareness approach into the matching process of publish/subscribe paradigm. Finally, as a proof of concept, we extend the Apache ActiveMQ Artemis software and create a client prototype. We evaluate our proof of concept for larger scale deployment. A publication[1] was issued, based on this thesis work, in the international conference ruSMART'2016.

[1] Dhallenne, J., Jayaraman, P., & Zaslavsky, A. (2016). RCOS: Real Time Context Sharing Across A Fleet Of Smart Mobile Devices. In The 9th conference on Internet of Things and Smart Spaces ruSMART 2016. Proceedings (Vol. 9870). Springer.

ACKNOWLEDGEMENTS

This research is fully supported and funded by PERCCOM Erasmus Mundus Program [78] of the European Union. The authors would also like to show their gratitude and thanks to all the partner institutions, sponsors and researchers of PERCCOM program. Additionally, I would like to thank my supervisors Dr. Prem Prakash Jayaraman, for his support all along the way, and Prof. Arkady Zaslavsky, for his help and making this work possible. I would also like to thank Prof. Ahmed Seffah, Susanna Koponen, Prof. Jari Porras and Prof. Éric Rondeau for their support during the semester.

Table of Contents

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application program interface
CC-BY-SA	Creative Commons — Attribution-ShareAlike
MIT	Massachusetts Institute of Technology
REST	Representational state transfer
SQLITE	Structured Query Language Lite
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1. Introduction

1.1. Addressing sustainable development

Direct sales of farmers are decreasing to the profit of retailers who tend to obtain a quasi-monopoly on consumers' food distribution. This raises several issues. Firstly, due to their market position, food retailers ask the farmers in the EU to constantly lower production prices. Many of these farmers are starting to go bankrupt as we see it in 2016 in France and Finland. Secondly, most of the farmers still in activity are imposed to cultivate specific species selected by different stakeholders focused on a quantitative production. This endangers biodiversity and more species disappear every year. In 2016, the FAO (Food and Agriculture Organization of the United Nations) writes that "since the 1900s, some 75-percent of plant genetic diversity has been lost as farmers worldwide have left their multiple local varieties and landraces for genetically uniform, high-yielding varieties" [1]. The final issue that this procedure involves has to do with the transportation. Instead of being directly or even indirectly brought through a short circuit to the end user (consumer, cook in a restaurant, …), the food goes through different locations. This process affects its quality, nutritional values, taste and contributes to the global warming through greenhouse gas emissions.

In this thesis, we address the need for having a direct communication between producers and consumers in order to break the monopoly enforced by food retailers on food distribution. The proposed system allows producers to express and share the availability of new products and the contextual information about the products (e.g.: price, taste, location ...) by matching the contextual preferences of the consumer in real-time. To this end, we propose, develop, implement and validate a context-aware message exchange system based on the widely used publish/subscribe paradigm. Within the focus of this thesis, we define context being based on two levels of contexts, which are the location and the personal preferences of the consumer and producer. We consider the following context, namely location and personalization context, in-order to deliver the most appropriate response to the producer and consumer. For example, to determine the availability of Pink Lady Apple, we consider the location context of the publisher and subscriber, and the personalization context of the producer and subscriber (quality, cost, etc...). In order to model the context, we propose a semantic approach that uses ontologies to express context using a consistent representation well in line with the semantic web research community. As a proof of concept, we develop a platform enabling potential customers to be notified about new products matching their contextual preferences based on context provided by the producers. The platform brings near real-time information sharing to publishers and subscribers in a context aware manner. Near real-time stands for as fast as possible. This means that across a group of entities who subscribe to an interest with associated contextual preferences (e.g.: type of product, location and/or price range, etc…), the availability of a product based on contextual matching is to be delivered to the customer instantly. The entities in this case could be mobile devices used by customers who could assume the role of publisher/subscriber.

1.2. Motivating scenario – Apple distribution

Petri is producing Pink Ladies, which are sour apples, and he wants to sell them, for delivery and pick-up, in the surrounding areas (Lappeenranta). The price for a kilogram is 3.50 euros. In a normal situation, this producer would wait for contractors to call him or would look for new clients himself. This client finding procedure is often carried via a "word-of-mouth" channel or answering demands. In this given situation, the product "Pink lady apple" produced by the producer with the following context attributes, namely sourness, production location, delivery location, and cost need to be matched to an interested customer. This matching procedure is then often taken care of by a retailer. However, we want to give producers and customers the possibility to have their interests directly matched, without the help of a third party actor. We want the producer to be able to subscribe to a particular set of interests which will be matched in a near real-time manner when publishers publish their interests (i.e. clients and direct consumers) but it can also be the other way around. Customers (i.e. clients and direct consumers) can also create a subscription to be notified when a producer will be selling apples and will be willing to deliver / offer a pickup to them for a certain price.

Tero is holding a restaurant in the Lappeenranta area and he is looking for apples that are sour in this area. In this situation, the consumer Tero has a set of preferences defined by the context attributes which are the taste of the apple and the pickup location. If this set of preferences matches with an existing product, we have a semantic match. This is the case, since Petri is producing Pink Ladies in the Lappeenranta area.

In a typical publish/subscribe system, the producer has to specify, in a single string, the product attributes he wishes to be part of the matching process. Moreover, it is not possible to express relationships between the entities being matched, and only a limited matching based on logical operators and string comparisons can be made. However, with the proposed context-based system it is possible to do semantic matching. This semantic matching can also be, for instance, a location within a certain radius of another location. In a content-based system, this could only be handled by a client tool and not by the broker of the publish/subscribe system since it cannot process contextual data. In the case of a topic based system, we would be required to have an apple category with sub-categories such as “sour apple” or “apple produced in Lappeenranta”. In this situation, cross-matching would not be possible and the operation would imply a high computational resource need.

1.3. Thesis aim and contribution

The aim of this thesis is to propose, develop, implement and validate a real time context sharing and subscription system using the widely used publish/subscribe paradigm. In this thesis, the term context sharing is used to describe the ability of entities to share their context based on the publish/subscribe paradigm. To address this aim, we break the aim into following research questions:

(a) What are the requirements of a context-aware subscription language allowing entities to share and subscribe to context?

In order to address this question, we have conducted an extensive literature survey to identify the current state-of-the art and gaps in publish/subscribe based systems and the corresponding context-aware capable subscription languages.

(b) How can we design and develop a context-aware subscription language that allows entities to share and subscribe context?

Currently, no publish/subscribe broker directly handles representation or processing of context data. Hence, we propose and investigate a semantic web-based approach in order to represent, share and subscribe to context. To this end, we proposed and developed RCOS, a real time context sharing system based on semantic web principles. The contribution includes the history based approach and a mobile application enabling entities to share context via mobile smart phones. The history is a graph, expanding as publications are removed from the main graph, with contextual attribute values and dates. The mobile application allows seamless exchange of context between the entities through RCOS.

(c) What are the performance issues imposed in such a system that performs context-aware publish and subscribe?

To answer this question, through proof of concept and evaluation, we study the performance of the system in order to determine the overhead imposed by the RCOS system.

The motivation behind our contribution is the lack of context-awareness, with easy integration to semantic web services, in the modern publish/subscribe systems. We also bring novelty by introducing a history consideration for certain attributes of ontologies, which to the extent of our knowledge, has not been introduced in context-aware publish/subscribe systems. To address this gap of knowledge we developed Real Time COntext Sharing System (RCOS), we proposed and developed an interoperable publish/subscribe system extension which is context aware and history-enabled.

1.4. Thesis structure

The below figure 1 presents the outline of this thesis.

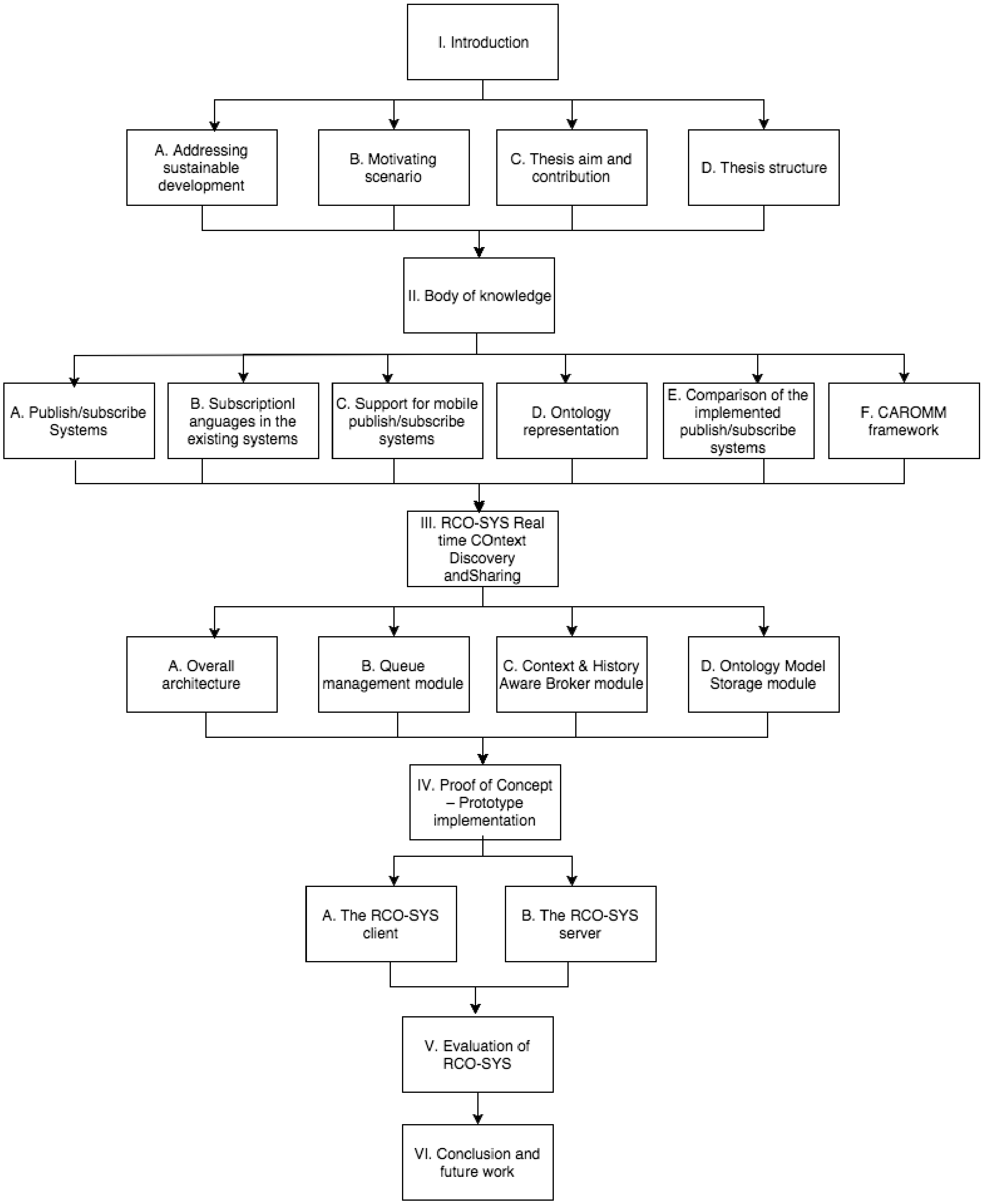


Figure 1. Thesis outline

In the second section, we look at publish/subscribe systems and their language, as well as their mobile support. We then dig into ontology representation in publish/subscribe systems. This leads us to create a comparison review. Lastly, we introduce the CAROMM framework effort which is used by RCOS to retrieve real-time context information.

In the third section, we describe our contribution, RCOS which includes a subscription language, the presentation of a context aware broker reasoning according to defined ontologies. Considering the apple ontology, we present the implementation of a proof of concept through an integration with Apache ActiveMQ Artemis publish/subscribe system in section four.

In order to evaluate our system for larger scale deployment, section four will be dedicated to measuring performance by introducing a high amount of ontologies, subscriptions and publication into our system. Section five will cover the performance analysis of our proof of concept. Finally, section six outlines the conclusion of this thesis as well as the future work.

2. Body of knowledge

In this section, we firstly review the different models of publish/subscribe systems introduced by Eugster et al. [2]. From these models we position our approach as a context-based approach. We later review the existing subscriptions language for publish/subscribe systems. Campailla et al. [3] define three types of subscription query languages. Our finding is, that currently, existing and researched context-aware publish/subscribe systems, such as the one introduced for Elvin [4] follow a Simple Subscription Language, which is our case as well, because non-defined attributes results in accepting any value for them. This allows us to keep the query representation as small as possible. Publish/subscribe systems with a consideration of mobile devices are then introduced. CUPUS [5] for instance, support mobile systems in the core of its broker. However, these systems only embed specific ontologies, such as location, and cannot take into account different context types. In our contribution, we introduce the consideration of different context types based on ontologies defined, or to be defined, for the semantic web effort [6]. This brings us to explore the current serialization languages which allow to model ontologies until the recent JSON-LD [7], which we think is the most suitable when considering mobile technologies, due to its minimal overheads. We then compare existing systems in terms of interoperability, and point out what RCOS bring. Finally, we introduce the CAROMM effort, on which RCOS relies for retrieving real context elements.

2.1. Publish/subscribe Systems

The publish/subscribe paradigm is constituted of publishers publishing information also known as events and subscribers sending subscriptions representing their interests. The distribution of such events to the corresponding subscribers is handled by a broker and the matching of these events is usually based on a string-matching process, or in simpler cases, the queue to which they are sent has defined subscribers. The broker allows subscribers and publishers to exchange information without knowledge of each other and in an efficient manner, based on interests expressed by subscribers during the subscription process. Subscribers are delivered information about the event published matching their interests and this process is usually called notification.

The broker can be a single entity on a certain server or can be distributed. We will mention more about the distributed publish/subscribe systems in the paragraph C, while covering mobile publish/subscribe systems. Figure 2 represents the architecture of a publish/subscribe system.

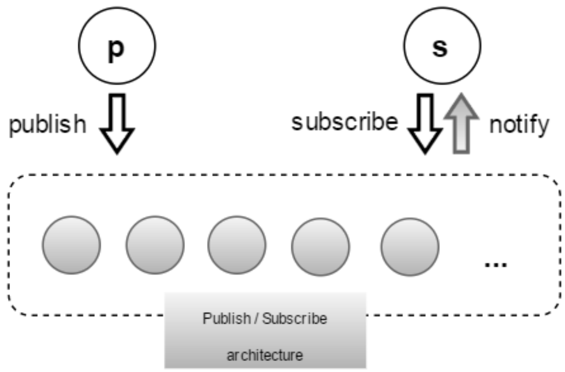


Figure 2. Publish/Subscribe architecture

There are various ways to specify the events emitted by publishers to the broker or brokers, in case of a distributed configuration. These models vary in the degree of expressiveness they offer to subscribers and the matching precision depends on this factor. Eugster et al. [2] differentiate four main types of subscription models.

Topic-based Model

Topic-based model has been widely available in the literature, as for instance, in the works of Oki et al. [8], Altherr et al. [9], Castro et al. [10] and the Object Management Group [11]. The organization of this model is based on topics, which are quite similar to groups. Notifications are transmitted to matching topics which subscribers have declared their interests for. They are consequently forwarded all the messages transmitted to these particular topics contained in the notifications for most of the systems. The topic, to which one can subscribe, is carried as an attribute of a given event, allowing distributing peers to distribute the message to the right subscribers. Topics ideally correspond to logical domains which allow the diffusion to be handled properly in a multicasting manner.
This approach is, however, non-hierarchical and makes it impossible for a subscriber to subscribe to a subset of events in a given topic. Eugster et al. [2] describe it as a flat approach. This issue is addressed by some implementations such as the one from Oki et al. [8] who are

creating and implementing a hierarchy model. This allows subscribers to specify, in a string-based attribute, which group or sub-group the peer wants to express its interest for. This model of hierarchy could be compared to the one used in the Usenet news network. Wildcards, which are usually represented by the "*" symbol, may also be used in the attribute. This method lacks flexibility and does not involve the content of a subscription in the process. The content-based model fills this gap. However, an advantage to this method is the low processing time required in order to answer a request.

Content-based Model

The previously mentioned constraint, despite the possibilities offered by the way they are implemented, was improved by Rosenblum et al. [12] by introducing a subscription scheme based on the actual content of the candidate event. Subscribers are also given the possibility to specify conditions over the content of the notification they wish to receive. This is expressed through a set of operators and a specific subscription language which regular expressions can be part of. The more complex the language is, the more complex the matching process will be. Examples of systems using this approach are JEDI [13], LeSubscribe [14], Ready [15], Rebeca [16], Hermes [17], Elvin [18] and MundoCore [19].

The content of the messages, can be given a standard for easier and more interoperable interpretation by the brokers, as it is the case in these systems [20, 21, 22] using XML as a subscription language. Subscription languages are discussed in the next paragraph. It is worth noticing that one of the disadvantage of XML is its mandatory envelope involving further processing.

To our understanding, some researches, however, differentiate XML-based from the content-based models such as the one conducted by Tarkoma et al. [23]. We consider them as being a subset of the content-based approach in the sense that the brokers will, to the extent of our knowledge, interpret the content without relying on an existing defined ontology.

Type-based Model

The type-based model has been introduced by Eugster et al. [24] due to the fact that topics usually regroup events presenting similar types. This approach that matches events to subscriptions based on type allows a direct encapsulation into attributes, as well as methods. The type safety is directly embedded into the publish/subscribe system rather than into the application by the programmer. This approach is considered a balance between the topic and the content-based models since it is a flat approach that also allows defining the constraint but on the typing. Systems such as [24] and [25] use this approach.

Context-based Model

This approach is based on ontologies. Some works also refer to this model as concept-based [23]. All the above models assume that, according to Tarkoma et al. [23]**:** "participants have to be aware of the structure of produced events, both under a syntactic (i.e., the number, name and type of attributes) and a semantic (i.e., the meaning of each attribute) point of view." He also mentions, "concept-based addressing allows to describe event schema at a higher level of abstraction by using ontologies that provide a knowledge base for an unambiguous interpretation of the event structure by using metadata and mapping functions." This concept based addressing is introduced by Buchmann et al. [26]. In our system, we use a context-based model using ontologies as a knowledge base. These ontologies are defined in the JSON-LD format [7].

Specific ontology based Models

Tarkoma et al. [23] also distinguish a model based on location. To our understanding, any ontology could be fitted into this category. There are several examples of publish/subscribe systems using the location as the factor for the messages to be distributed to subscribers, such as [22, 27, 28]. This approach, however, limits to a single and specific ontology, distinguishing the broker message distribution by the value of the attributes of this ontology. In their paper and implementation, Eugster et al. propose a location-based publish/subscribe system [29]. This mixes the notion of a specific ontology based model and the content based model. This approach gives them more contextual possibilities than classical content-based models.

Context awareness in existing publish/subscribe systems

Works on implementing the context awareness paradigm in publish/subscribe brokers already exists in the research community. Loke et al. [30] introduced a context-based addressing effort for Elvin. This work contributes to allowing to distribute messages to users in a chosen context according to ontologies interpretation**.** For this, they also created a context-aware capable language [4]. Elvin was also included in the ECORA framework from Padovitz et al. [31], which provides a hybrid architecture for context-oriented pervasive computing. However, Elvin as well as its open source implementation Avis [32] suffer from a lack of popularity nowadays, and are missing maintenance as well as cross-language support, since they handle the messaging process using its own standard that is not widely implemented.

Other recent studies such as the one realized by Tarkoma et al. [33] propose very efficient and flexible solutions. However, the concept of ontology is not fully considered and implemented. It is also worth noting the work and vision brought by Cugola et al. [34], which brings a distributed protocol to publish/subscribe systems according to their location to allow a more efficient distributed broker system. Our vision, however, places the location as a specific ontology. While their work is focused on the physical distribution of messages, our contribution is meant to bring a more generic way to embed context-awareness into publish/subscribe paradigm.

In [35], Zahariadis et al. introduce a novel context-aware publish/subscribe system. This system is developed within the effort of a single digital market in the European Union. Their context-based broker, however, does not include a subscription language, nor does it support preliminary defined ontologies as knowledge base with history. It is also limited to the REST protocol.

Commercial publish/subscribe systems

Table 1 is a compilation of the commonly used commercial publish/subscribe systems with the communication protocols they support.

Table 1. Protocols supported by common commercial publish/subscribe systems

Protocol :	AMQP	MQTT	OpenWire	REST	STOMP	STOMP over Websockets	XMPP	RESP
ActiveMQ	1.0	X	X	X	X	X	X	-
Apollo	1.0	X	X	X	X	X	-	-
ActiveMQ Artemis	X	X	X	X	X	X	-	-
Qpid	X	-	-	-	-	-	-	-
RabbitMQ	X	X	-	X	X	X	Gateway	-
ZeroMQ	0.9.1	-	-	-	-	-	-	-
Redis	-	-	-	-	-	-	-	X
FIWARE-Orion [35]	-	-	-	X	-	-	-	-

Within the commercial publish/subscribe systems, Apache ActiveMQ is a mature one that supports a wide range of protocols. Apache Apollo is a performance oriented publish/subscribe system, however, it has now been abandoned as a project. Comparing to the original ActiveMQ, the Artemis project has as its main strength, a better performance [36]. While ActiveMQ 5.x requires mapping REST to JMS, which is a less "native" approach, Artemis directly handles both of them [37]. As of April 2015 [38], the Apache foundation started considering using this subproject as the base for the sixth version of ActiveMQ. The presence of a migration page for ActiveMQ also shows this case as being the most probable one for the future [39]. The Qpid system only supports AMQP. RabbitMQ is often chosen as a favourite publish/subscribe system. It is, however, written in Erlang which does not provide the same interoperability with mobile systems as Java does due to its wider adoption. ZeroMQ is meant for distributed systems and cannot involve a central configuration. Redis only supports its own protocol RESP. FIWARE-Orion is the "commercial" result of Zahariadis et al. research [35]. It is compatible with any system using REST API but it does not provide nor a subscription language, nor backward compatibility with existing publish/subscribe systems using older protocols.

Considering these factors, we focused our choice on Apache ActiveMQ Artemis due to its interoperability and number of protocols supported. Our choice of the REST protocol working with ActiveMQ Artemis is due to the fact that this type of communication involves smaller network load and faster interpretation, as well as high and lightweight interoperability [37]. Our proof-of-concept of RCOS in section 4 could, however, be ported to other publish/subscribe systems.

2.2. Subscription languages in the existing systems

Subscription languages used in publish/subscribe systems are more or less descriptive. The more operations are defined in a language, the higher is the complexity and processing time. Thus, according to Carzaniga et al. [40], in practice, scalability and expressiveness are two conflicting goals that must be traded off.

In [3], Campailla et al. define three types of subscription query languages, which are SiSL, StSL and DeSL. They describe them as follows:

- The Simple Subscription Language, SiSL, type of language is used where all messages are total. This subscription language is directed to messages of known format, which are typically used in a non-distributed setting or for specialized applications. If an attribute is not defined in the query, it matches the pattern "*", which means any value queried would return true.

- The Strict Subscription Language StSL is an extension of SiSL where all attributes that occur in the query must be defined.

- In the Default Subscription Language DeSL, all attributes are initialized to a default value, which are then updated by the message. Using the default values, it is possible to test if the attributes are defined by a message. This way, DeSL extends the functionality of SiSL to heterogeneous message formats, as it is often the case in distributed settings. Default can be symbolized using one of the NULL semantics such as those provided by JMS [41]

Campailla et al. also note that over total messages, SiSL, StSL and DeSL are equally expressive. The approach they have for their filtering engine is based on binary decision reasoning while Elvin [18] uses Łukasiewicz's tri-state logic. In our approach, we use binary decision reasoning as well for a more efficient processing.

To the best of our knowledge, Elvin [4] is the only publish/subscribe system in the research community until 2014 with a subscription language defined so that it could be extended to incorporate a general context awareness capability considering ontologies due to its <action> <proposition> tuple integration. Others such as [33] and [29] do not fully incorporate the notions of ontologies. Tarkoma et al. and Eugster et al. are focused on location context while we want to incorporate context defined via ontologies.

An example of Elvin's subscription language, reproduced from [5], is defined in figure 3.

```
(request-whenever
:sender (agent-identifier :name i)
:reeiver (agent-identifier :name es)
:content
   "((action (agent-identifier :name es)
      (inform
         :sender (agent-identifier :name es)
         :receiver (agent-identifier :name i)
         :content \"(notification n)\"
      ))
      (mathes (subexp e) (notification n))
   )
   "
)
```

Figure 3. Elvin's subscription query reproduced from [5]

For this case, the subscription expression be would as states the first line of figure 4 and the notification as it is following in figure 4.

```
(TYPE == "Apple" &&  TASTE == "Sour" && ORIGIN == "France") && (PRICE >= 
1.50 && PRICE <= 2.10)

TYPE: "Apple"
PERSON: "http://example.org/profile/Tero5872"
TASTE: "Sour
ORIGIN: "France"
PRICE: ">= 1.50"
PRICE: "<= 2.1"
TIMEOUT: 10
Message-Id: "08cf0b15003409-5i3N7XDKbEVaQ-88cf-12"
```

Figure 4. Elvin's subscription expression and notification

Using a subscription language based on XML or JSON-LD [7] does not require distinguishing the subscription expression from the notification.

In our proposed subscription language modeled through JSON-LD, the previous examples can be expressed as in figure 5:

```
{
   "@context":
   ["http://schema.org/",
   {
      "lfd": "http://example-localfood.org/"
   }],
   "@type": "Person",
   "@id": "http://example.org/profile/Tero5872"
   "seeks": {
      "@type": "Demand",
      "itemOffered":{
         "@type": "lfd:Apple",
         "lfd:taste":"sour",
         "lfd:origin":"Finland",
      },
      "highPrice": "2.10",
      "lowPrice": "1.50",
   }
}
```

Figure 5. RCOS's subscription query model

This approach allows defining nested relation between entities. In this manner, we can express how an ontology relates to another ontology.

According to Campailla et al. [3], Elvin's and our approach are Simple Subscription Language (SiSL) since non-defined attributes results in accepting any value for them. This allows us to keep the query representation as small as possible.

The way we distinguish our approach from the one presented in Elvin is that we have a defined subscription language that does not contain operators. In our subscription language, the comparisons are done according to defined semantics from the schema.org effort [42], which is an effort to create schemas for structured data on the Internet by ontologies, and our semantic modeling ontologies. JSON-LD definition allows us to directly embed in, through defined attributes, logic operators interpreted due to their involvement as attributes in a given ontology.

For example, an ontology containing the properties *maxPrice* and *minPrice*, as defined in schema.org, involves that the broker compares the property *price* for a similar ontology, so that it defines a result for *minPrice* > *price* > *maxPrice*.

In Elvin, this would have been defined in the subscription language in the following way: (PRICE >= *minPrice* && PRICE <= *maxPrice*). Moreover, Elvin's extension for context-based consideration relies on agents [32]. In our approach the broker takes care of both interpreting and spreading messages.

Our contribution brings a subscription language which allows enabling publish/subscribe systems' brokers to semantically interpret a context by incorporating ontologies. We modelled these ontologies through JSON-LD standard, which allows our subscription language to easily integrate with current semantic web services. This integration is not taken into consideration in Elvin. Our contribution is also compatible with JSON. JSON has become a de-facto standard in the last years when it comes to REST APIs, the latest recently expanding to mobile phones and allowing interoperability with services since this technology involves using standard web requests. Compared to XML, used by several publish/subscribe systems, the JSON-LD standard is minimalist and includes lighter overheads which also allows a more efficient interpretation. Interested readers may want to read more in [43].

In case of a public publication on a webpage, a direct integration with applications gathering publicly available JSON-LD declarations according to the schema.org effort is possible. A known application of such a gathering is Google's Knowledge Graph [44], which is used in their commercial search engine and Google Now assistant. In the future, this approach could also allow a publish/subscribe system to be able to automatically match content discovered across the semantic web [6].

2.3. Support for mobile publish/subscribe systems

Mobile publish/subscribe systems are introduced in the research community as early as 2000 by Cugola et al. [45]. In 2001, Huang et al. [46] argue that a distributed broker on stable non-mobile networks is a safer and better approach for mobile publish/subscribe system, since a centralized broker may introduce a performance bottleneck and a single point of failure. While we agree on this point, the distribution and replication of the broker is beyond the scope of this thesis, as it would require a dedicated study involving performance analysis regarding the time of distribution as well as implementing the ontologies distribution in an optimal way (i.e.: favor local node to include locally related contexts and ontologies).

A first introduction, to the best of our knowledge, mixing publish/subscribe mobile system with context awareness was made by Fiege et al. [47] as they analyze the problem of mobility in such systems. Resulting to this analysis, they implement the consideration of physical and logical mobility in the already existing content-based Rebeca [16]. Cugola et al. later introduced an efficient algorithm directed to enable location awareness in publish/subscribe systems [48]. Salvador et al. did further work on location awareness [49] and introduced a protocol for seamless client mobility in publish/subscribe systems [50]. Recently researches have introduced mobile brokers [51] into the mobile publish/subscribe paradigm. This approach allows energy efficiency over the network and reduces the energy waste induced by the previous approaches when it comes to including sensors into such a system. Both [51] and [52] are papers involving mobile brokers. Antonić et al. in CUPUS [52], however, add a cloud broker. This allows efficient mobile brokers to process, with concerns on energy efficiency, the local sensors' data while enabling a context aware distribution of it over the internet. Soldatos et al. involved in the OpenIoT platform [53] integrate the W3C Semantic Sensor Networks (SSN) ontology in addition to CUPUS. Antonić et al. also developed a mobile crowd sensing ecosystem enabled by CUPUS [5].

The common point of the previously mentioned context-aware mobile publish/subscribe systems is that they only consider location as a context or integrate a specific ontology. [51], [52], [53] and [5] are oriented towards the Internet of Things and meant for a sensor integration but none of them is actually directly meant to take into consideration different context types based ontologies defined, or to be defined, for the semantic web effort [6] which we want to introduce in this thesis.

2.4. Ontology representation

Since more than two decades, different ways have been used to describe ontologies in a machine-readable format. The most notable ones are RDF [54], RDFS [55], OWL [56] and recently JSON-LD [57].

While the first public draft was released in 1997 [41], the Resource Description Framework (RDF) became a W3C Recommendation in 1999 [54]. Its primary purpose, according to the first press release, is "to allow different application communities to define the metadata property set that best serves the needs of each community".

Other standards such as TriX [58], TriG [59] and Turtle [60] are containers for the RDF specification. While TriX is used for serializing named graphs and RDF datasets as a XML alternative to RDF/XML, TriG is a compact alternative to TriX. Turtle, on the other hand, is based on a subset of N3 discussed below. It allows representing data in the RDF data model with a syntax similar to SPARQL [61], which is a RDF query language.

A major revision was made in 2004 [62]. During the same year, the Resource Description Framework in attributes (RDFa), allowing to embed rich metadata within web documents through a set of attribute-level extensions, becomes a W3C recommendation [63].

While RDF allows you to represent a collection of triples, each consisting of a subject, predicate and object. Notation3 (or N3) [64], being another ontology representation format, also extends it in order to add features from first-order logic. RDFS (RDF Schema) [55] was published in 2004. It gives a more expressive vocabulary by allowing classifying resources through classes and subclasses, to set restrictions on proprieties in a domain knowledge or using ranges. In 2004, then revised in 2009 and standardized by the W3C, the Web Ontology Language as known as OWL [56], adds more possibilities of restriction to the knowledge representation by adding the possibility to have proprieties into object and data proprieties. It also allows you to add restriction proprieties definition via cardinalities and logic operators. It is possible to perform reasoning on OWL through programs such as Pellet [65]. SPARQL language [61] defines a way to query any representation that is convertible to RDF or OWL.

The recent JSON-LD (JavaScript Object Notation for Linked Data) [57] is designed to provide a possibility to map JSON to the RDF format. It uses a context embedded in the JSON document or pointed to through a URL to link its object proprieties to concepts of an ontology. It became a W3C recommendation in 2010 [57] and was revised in 2014 [66]. Figure 6 gives an example of such a document.

```
{
    "@context": "http://schema.org/",
    "@type": "FoodEstablishment",
    "name": "Joe's Pizza",
    "location": {
        "@type": "PostalAddress",
        "@id": "http://example.com/address",
        "streetAddress": "123 Main Street",
        "addressLocality": "Cambridge",
        "addressRegion": "MA",
        "postalCode": "02142"
    },
    "makesOffer": {
        "@type": "Offer",
        "priceSpecification": {
            "@type": "DeliveryChargeSpecification",
            "appliesToDeliveryMethod":
"http://purl.org/goodrelations/v1#DeliveryModeOwnFleet",
            "eligibleTransactionVolume": {
                "@type": "PriceSpecification",
                "price": "20.00",
                "priceCurrency": "USD"
            },
            "eligibleRegion": {
            "@type": "GeoCircle",
            "address": {
                "@id": "http://www.example.com/address"
            },
            "geoRadius": "5000"
            }
        }
    }
}
```

Figure 6. JSON-LD document example reproduced from [42] under CC BY-SA 3.0 license

This document represents a food establishment called “Joe’s Pizza” through the FoodEstablishment ontology type of schema.org’s [42] vocabulary definition. Such an ontology can carry proprieties like a name, a location that is represented by a "PostalAddress" ontology, an offer made by the establishment, which is carried by the "Offer" ontology.

In RCOS, we use ontologies as knowledge base to represent contexts, as well as for our subscribing language. These ontologies are also modelled through JSON-LD. A part of ontologies we use come from the schema.org effort [42], however, for those which are not yet standardized, we have defined them, as well as new attributes.

OpenWines [67] is an example of external actor working on modelling new ontologies in JSON-LD. They have modelled the ontology presented in figure 7 which we reproduced from [68] in order to semantically represent winemakers.

```
{
    "@context": [
        "http://schema.org/",
        { "ow": "https://github.com/OpenWines/Open-
Data/tree/master/Ontologies/1.0/" }
    ],
    "@type": "Winemaker",
    "ow:isLandowner": true,
    "address": {
        "@type": "PostalAddress",
        "addressLocality": "Sainte Lumine de Clisson",
        "addressRegion": "Pays de la Loire",
        "postalCode": "44190",
        "streetAddress": "26 les Défois"
    },
    "memberOf": {
        "@type": "Organization",
        "name": "Syndicat Défense Des AOC Muscadet",
        "url" : "http://www.muscadet-grosplant.fr/",
        "telephone": "+33 2 40 80 14 90"
    },
    "businessRegistration": "RCS Nantes 514582691",
    "isicV4": "11.02",
    "name": "Durand Vigneron",
    "openingHours": [
        "Mo-Sa 11:00-14:30",
        "Mo-Th 17:00-21:30",
        "Fr-Sa 17:00-22:00"
    ],
    "telephone": "+33 2 40 54 70 03",
    "fax": "+33 2 40 54 70 03",
    "email": "mailto:durand.verteprairie@wanadoo.fr",
    "url": "http://www.durand-vigneron.com"
}
```

Figure 7. OpenWines "Winemaker" ontology representation reproduced from [68] under MIT License

OpenWines' winemaker ontology represents a wine producer since schema.org proposes a "Winery" ontology, which is a subtype of "FoodEstablishment". They also add a non-existing attribute "ow:isLandowner". This attributes gives the information on whether or not a wine producer owns land. This attribute could also include a tuple composed of whether or not a land is owned and a land ontology. In the next paragraph, we compare the previously mentioned publish/subscribe systems.

2.5. Comparison of the implemented publish/subscribe systems

In the table 2, we have reviewed the previously mentioned publish/subscribe systems which are implemented. We have reviewed in which type they can be classified, their subscription protocol and/or format, their interoperability, whether they consider mobile technologies or not and the programming languages for which API are provided for the broker.

Table 2. Publish/subscribe systems comparison in terms of interoperability

System	**Model used / type**	**Subscription protocol / format**	**Interoperability**	**Mobile technology considered**	**Programming language API (broker)**
The information bus [8]	Topic-based / Research	Remote Method Invocation	RMI (CORBA compatible)	No	C++
SCRIBE [10]	Topic-based / Research	Specific over TCP	None	No	Java, C#
CORBA [11]	Topic-based / Research	Specific over TCP	None	No	C++, Java
JEDI [13]	Content-based / Research	Specific over TCP	None	Yes - not in the first version	Java
LeSubscribe [14]	Content-based / Research	Remote Method Invocation	RMI (CORBA compatible)	No	Java
Ready [15]	Content-based / Research	Specific over TCP	CORBA	No	C++
Rebeca [16]	Content-based /	RMI / SNMP /	3 protocols	No	Java

	Research	HTTP			
Hermes [17]	Content-based / Research	XML over TCP	None	No	Java
Elvin [18]	Content-based / Research	Specific	None	Yes	Java, C
MundoCore [19]	Content-based / Research	Specific	None	Yes	Java, C++, Python
XNET [20]	Content-based / Research	XML over TCP	None	Yes – incl. data flow limitation	No information
Eugster et al. [25]	Type-based / Research	Specific	None	No	Java
ECA [26]	Context-based / Research	XML over SOAP	SOAP	Yes	Java
ActiveMQ	Topic-based / Commercial	AMQP, MQTT, OpenWire, REST, STOMP, XMPP	6 protocols	Yes	Java
Apollo	Topic-based / Commercial	AMQP, MQTT, OpenWire, REST	4 protocols	Yes	Java
ActiveMQ Artemis	Topic-based / Commercial	AMQP, MQTT, OpenWire, REST	4 protocols	Yes	Java
Qpid	Topic-based / Commercial	AMQP	Yes, AMQP	Yes	Java / C++
RabbitMQ	Topic-based / Commercial	AMQP, MQTT, REST, STOMP, XMPP	5 protocols	Yes	Erlang
ZeroMQ	Topic-based / Commercial	AMQP	Yes, AMQP	Yes	C++
Redis	Topic-based / Commercial	RESP	None, only RESP clients	Yes	C
FIWARE - Orion	Context-based / Commercial	REST	Yes, REST	Yes	C++

The information bus [8] is one of the early works on publish / subscribe systems. It uses Remote Method Invocation in order to publish and subscribe information. At the time of the publication, this system provides a perfect tool for distributed systems, as it allows bringing the publish/subscribe paradigm to different machines communicating together. SCRIBE [10] and CORBA [11] are two publish/ subscribe research following the information bus. SCRIBE, being based on Pastry [69], brings reliable and scalable alternative to IP multicasting through the publish/subscribe paradigm on application level, balancing the load between nodes and being focused on a peer-to-peer configuration, while CORBA is designed to facilitate the communication of systems deployed on different platforms. This standard is implemented in C++ and Java and has standard mappings in Ada, C, C++, C++11, COBOL, Java, Lisp, PL/I, Object Pascal, Python, Ruby and Smalltalk. To the best of our knowledge, none of these systems has been designed considering mobile integration. In terms of early content-based systems, neither JEDI, LeSubscribe, Ready, Rebeca or Hermes provides a mobile device consideration. While JEDI is later introduced mobile nodes consideration in [70] and Rebeca extended to this possibility by Andreas et al. [71], they do not provide a "mobile-friendly" subscription language for developers. LeSubscribe uses RMI, Ready has a specific communication protocol over TCP while Hermes has as well but includes XML on an application level. Rebeca, however, can communicate through RMI, SNMP and HTTP, which brings a better interoperability. In terms of systems from the research community, the later ones all consider nodes mobility in their first implementation. Elvin, MundoCore and Eugster et al. [25] all have a specific way of subscribing and publishing, which limits their interoperability. XNET uses TCP connections and includes XML on an application level, while ECA transmits XML over SOAP. The latest brings a better interoperability since it follows the SOAP protocol. SOAP is, however, an information rich protocol, and in terms of performance, it is less oriented towards mobile devices than REST for example. As for the commercial publish/subscribe systems, all of them can be integrated into mobile technology, Apache ActiveMQ projects being the ones that support the biggest number of protocols. FIWARE Orion brings novelty in the commercial publish/subscribe systems by being context-based. It still, however, lacks a subscription language and handles queries via REST URLs. RCOS embeds a subscription language which is oriented towards the semantic web.

2.6. CAROMM framework

Sherchan et al. [72] have developed the Context-Aware Real-time Open Mobile Miner (CAROMM) framework, which addresses according to them, "the research challenge of a highly scalable and efficient data collection for mobile crowd sensing". In order to address this challenge, they leverage on-board mobile data stream mining algorithms to reduce the amount of data transmission. This is done while still maintaining needed amount of sent information for extracting contexts. CAROMM general overview is represented in the figure 8 below which was reproduced from [73].

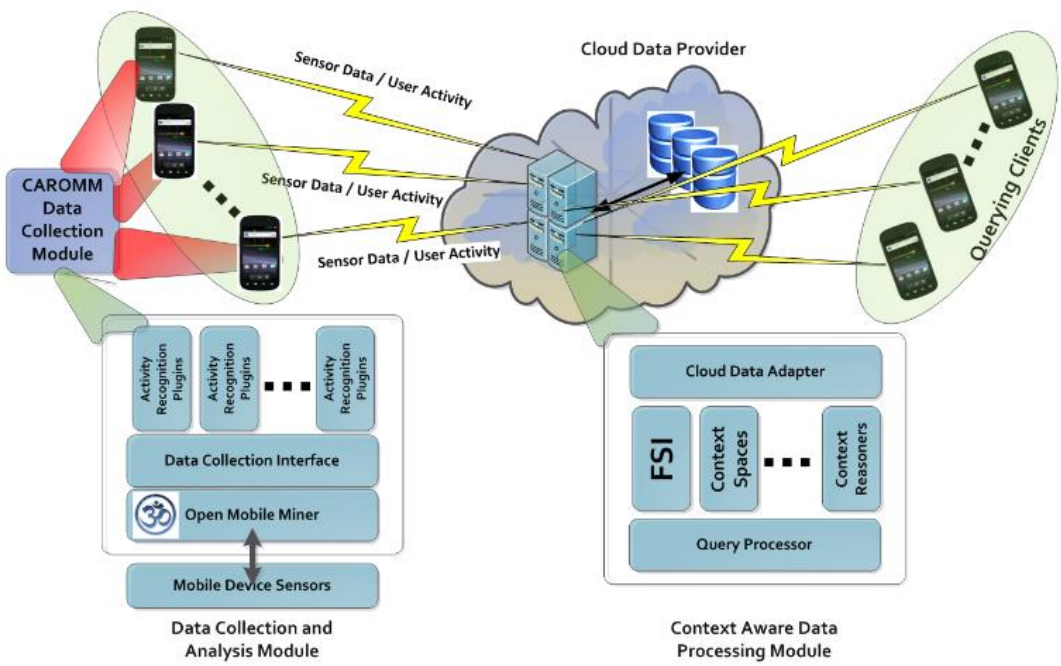


Figure 8. Overview The Here-n-Now Framework (based on CAROMM) reproduced from [73]

CAROMM is composed of two main modules, a *Data Collection & Analysis Module* and a *Data Processing Module*, as described by Jayaraman et al. in [73].

The *Data Collection & Analysis Module* is embedded into the mobile device, it includes the device's sensors, Open Mobile Miner [74], a data collection interface and a set of Activity Recognition Plugins. Once collected, the raw data is analysed and patterns are recognized. A resource-aware clustering technique is used to only send analyzed and useful data to the cloud. More about this and the resulting savings of this data transfer optimization can be found in [72] for interested readers. The cloud, to which the information is sent, contains the *Data Processing Module*. The *Data Collection & Analysis Module* can also be extended with an activity recognition plugin, which for instance could contain an activity recognition model based on neural network. This would allow, using the collected data from the accelerometer, to recognize walking, running, sitting and driving activities.

The *Data Processing Module*, which is in the cloud, is context-aware and will be handling a deeper analysis, management, and fusion of the data streams transmitted by the *Collection & Analysis Module*. This analysis is performed using the real-time sensory data and the activity data collected from the users. A part of the *Data Processing Module* also takes care of social media data collection in order to obtain plain information data that will then be evaluated and aggregated to better extract contexts. In the implementation “Here-N-Now” of Jayaraman et al. [73], a Fuzzy Interference (FSI) model [75] is used for context reasoning. This model integrates fuzzy logic into the probabilistic Context Spaces model [76].

RCOS can interact with CAROMM. It relies on it for retrieving real-time contexts from the cloud which are previously processed from the *Data Collection & Analysis Module*. These retrieved elements are linked to a specific JSON-LD context. More about the involvement of the current CAROMM framework with RCOS are given in the following section.

After going through the current publish/subscribe systems and their model, their subscription language as well as their implementation, we identified a gap of knowledge to be addressed. This gap of knowledge is reflected into the design of a publish/subscribe system that would meet future needs of an expanding volume of information on the Internet with phenomenon such as the Internet of Things and an increasing amount of information sharing. Data used to be treated according to their type (i.e. text, images, audio and video) but we provide a semantic system that will treat information according to its context and meaning within the context. We also went through the current ontology representations and recognize the opportunities offered by JSON-LD, onto which our subscription language is based. Finally, we reviewed the CAROMM framework, developed by Sherchan et al. [72]. RCOS can interact with CAROMM in order to retrieve real time information for ontologies' attributes requiring it and we reflected this possibility in the subscription language we developed.

3. RCOS - Real time COntext Sharing

In order to answer our research questions, we created RCOS. This system is a system that can be included in existing publish/subscribe systems in order to enable context-awareness and history consideration into them. In RCOS, we use ontologies to semantically represent our context information. This approach allows us to provide a system that can be easily integrated with current and future semantic web applications. In this section, we firstly give an overview of the overall architecture of our contribution. Secondly, we present each module and its functionalities.

3.1. Overall architecture

Figure 9 below describes the overall architecture of RCOS. It is composed of three modules which are the *Queue management module*, the *Context & History Aware Broker module* and *Ontology Model Storage module*. Each of these modules interact programmatically together. The *Context & History Aware Broker module* interacts with the existing CAROMM framework for real-time contexts retrieval. The *Ontology Model Storage module* is external to the Publish/Subscribe System in the sense that it does not interact directly with the publishing and subscribing processes, but is invoked by them for ontology modeling and storage.

The following paragraphs give more details on the internals of each module.

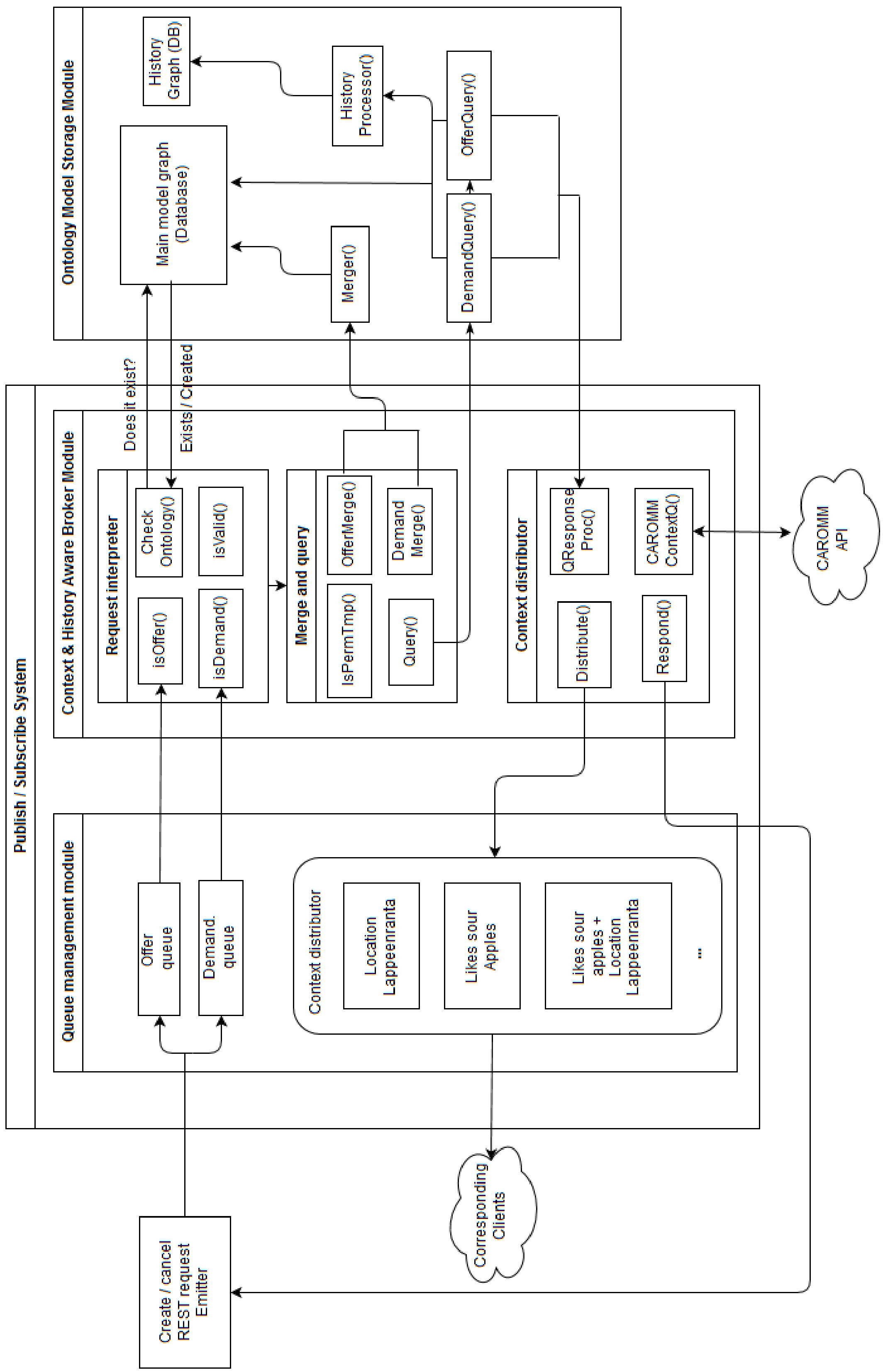


Figure 9. RCOS overall architecture

3.2. Queue management module

In RCOS, an entity publishes or subscribes to information about another entity. We model this concept through our ontology representation and subscription language. An entity emitting an offer offers information about another entity; this can be considered as a publication. An entity emitting a demand requests information about another entity; this can be considered as a subscription. Our approach can also allow, for instance, someone to requests a person's information in the case where we know that the person is looking for a certain entity's information (e.g.: an apple and its offers). Offers and demand can be permanent (until they are cancelled) or discrete.

The *Queue management module* handles two queues to which the offers and demands are posted (publications and subscriptions) via a REST interface. These REST requests can be of two types. Create requests imply that the *queue management module* will request for an addition to the ontology base during the merging operation, while a cancel requests implies that the *queue management module* will request for a deletion to the ontology base during the merging operation.

For example, in order to send a subscription query to the demand queue, one will need to send a file via a REST POST method to the address http://example.com:8081/queues/demand/create. If the query is discrete and should not be stored in the knowledge base graph, the address http://example.com:8081/queues/demand/request can be used.

The content of the file must follow the RCOS subscription language. The subscription language we define in RCOS brings together two ontologies on the base of information offer/demand relationship. In the figure 10 below, we represented the person of our scenario "Apple distribution". Tero wants to find sour apples (context attribute 1) in the Lappeenranta area (context attribute 2). The subscription he emits from the RCOS's client and his interests are linked by the type "Demands". The ontology, as it is represented in the knowledge base, follows the relationship linking both ontologies. More information on the modeling of ontologies in our knowledge base will be given in the *Ontology Model Storage module* paragraph.

```
{
    "@context":
    ["http://schema.org/",
    {
        "lfd": "http://example-localfood.org/",
        "crm": "http://example-caromm.org/"
    }],
    "@type": "Person",
    "@id": "http://example.org/profile/Tero5872",
    "seeks": {
        "@type": "Demand",
        "itemOffered":{
            "@type": "lfd:Apple",
            "description": "",
            "name": "",
            "lfd:species": "",
            "lfd:taste": "sour",
            "lfd:origin": "",
            "offers": {
                "@type": "lfd:AggregateOffer",
                "offers": [
                {
                    "@type": "Offer",
                    "price": "*",
                    "offeredBy": {
                        "@type": "lfd:Producer",
                        "@id": "",
                        "name": "",
                        "crm:busyState": ""
                    },
                    "priceSpecification": {
                        "@type": "DeliveryChargeSpecification",
                        "eligibleRegion": {
                            "@type": "GeoCircle",
                            "address": {
                                "@type": "PostalAddress",
                                "addressCountry": "",
                                "streetAddress": "",
                                "addressLocality": "Lappeenranta",
                                "name": "",
                                "postalCode": ""
                            },
                            "geoMidpoint": {
                                "@type": "GeoCoordinates",
                                "latitude": "",
                                "longitude": ""
                            },
                            "geoRadius": ""
                        }
                    }
                }
                ]
            }
        }
    }
}
```

Figure 10. Subscription query for our scenario "Apple distribution"

In the subscription presented in figure 10, all attributes of the ontologies that are to be returned from the knowledge base by the *Ontology Model Storage* module, relayed by the *Context & History Aware Broker module*, are specified. The wildcard "*" indicates to RCOS that the history for a certain attribute is requested. Such a history will be extracted from the history graph of previously cancelled offers by the *Ontology Model Storage*. This extracted information is the same context as the returned results for the demand itself. The "crm:" attribute will be described in the next paragraph. In this case, it allows us to know if the offer emitters are currently busy or not, which CAROMM framework will handle.

Sending a query to the offer queue follows the same logic in terms of REST URLs but the "demand" is replaced by "offer". For cancelling an offer, http://example.com:8081/queues/demand/cancel should be used with the corresponding file. As far as the file being sent is concerned, we created a sample for our scenario given in the below figure 11.

```
{
    "@context":
    ["http://schema.org/",
    {
        "lfd": "http://example-localfood.org/"
    }],
    "@type": "Person",
    "@id": "http://kotiomena-lpr.fi",
    "name": "KotiOmena T. Kakuunen"
    "offers": {
        "@type": "Offer",
        "itemOffered":{
            "@type": "lfd:Apple",
            "description": "Nice Red and Yellow Apple",
            "name": "Apple Fuji Bio",
            "lfd:species": "Fuji",
            "lfd:label": "FI-BIO-01",
            "lfd:taste": "sweet",
            "lfd:origin": "Finland"
        },
        "priceCurrency": "EUR",
        "price": "3.50",
        "priceSpecification": {
            "@type": "DeliveryChargeSpecification",
            "appliesToDeliveryMethod":
"http://purl.org/goodrelations/v1#DeliveryModeOwnFleet",
            "eligibleTransactionVolume": {
                "@type": "PriceSpecification",
                "price": "10.00",
                "priceCurrency": "EUR"
            },
            "eligibleRegion": {
                "@type": "GeoCircle",
                "address": {
                    "@type": "PostalAddress",
                    "addressCountry": "Finland",
                    "streetAddress": "Nuijamaantie 494",
                    "addressLocality": "Lappeenranta",
                    "name": "KotiOmena T. Kakuunen",
                    "postalCode": "53300"
                },
                "geoMidpoint": {
                    "@type": "GeoCoordinates",
                    "latitude": "61.038698",
                    "longitude": "28.362005"
                },
                "geoRadius": "10000"
            }
        }
    }
}
```


Figure 11. Publication query for our scenario "Apple distribution"

In this publication, the “person” offers Fuji apples for 3.50 euros per kilogram. The price specification attribute indicates that the delivery can be handled ten kilometers radius from their production site, and that a minimum price for delivery is ten euros. This context will be merged into RCOS’s knowledge base by the *Ontology Model Storage* module, previously handled by the *Context & History Aware Broker module*, which is in charge of analyzing it.

The context distribution for permanent demanders (subscribers) is handled through the context distributor, the study of which is outside the scope of this thesis. However, results are sent to clients in the JSON-LD format, following our subscription language. These results are then interpreted and can be visualized in each client’s application.

3.3. Context & History Aware Broker module

The *Context & History Aware Broker module* consists of three sub-modules. It serves requests from the *Queue management module*. We designed the sub-modules as follows:

Request interpreter sub-module

This sub-module first checks that the ontology format is valid through the isValid() function. If it is not, the request is rejected. The ontology is then handled as an offer or a demand. If it is an offer, it then queries the main model graph to know if the ontology exists through ChckOntology(), and if it does not, it is created. An offer or a demand object is then passed on to the Merge and Query sub-module.

Merge and query sub-module

The m*erge and query sub-module* behaves according to the object type it obtains from the *request interpreter sub-module*. In the case of an offer or a demand which is permanent and marked as such by the IsPermTmp() function, it will be merged in the ontology model graph by the *Ontology Model Storage module* described in the next paragraph. A demand or an offer cancellation will then be handled by the query module. This query function also relies on the *Ontology Model Storage module*.

Context distributor sub-module

This sub-module handles the *Ontology Model Storage module*'s response through QResponseProc(). If there is a need to call the CAROMM API to retrieve context attributes concerning the queried ontology, it will use the CAROMMContextQ() which will retrieve real-time contextual data. Such attributes start with the "crm:" as mentioned previously. As a final functionality, it will handle the context to distribute to the context distributor in the *Queue management module* via Distribute(), and if the response was for demand request, it will return currently existing results to the sender via Respond(). Otherwise, a simple acknowledgement is delivered to the sender.

3.4. Ontology Model Storage module

The *Ontology Model Storage module* handles the ontologies' knowledge base and history graph, which are loaded in the RAM memory, and the database to store them. It serves the *Context & History Aware Broker module* for verifying ontologies, merging them to the knowledge base graph and handling queries to issue responses.

When a merge request is issued from the *Context & History Aware Broker module*, the Merger() function will merge the ontology into the knowledge base graph. For queries, it will accordingly serve it as a demand by matching the corresponding sub-graph and returning it (contextual matching), or as a cancelled offer by deleting the corresponding ontology from the graph, and calling the HistoryProcessor() function with this given ontology. The matching is not solely based on attributes but also on their interpretation. For instance, the system can take into consideration radius information for a location, as well as the coordinates. If a demand query requests for the history of an attribute, it will also be handled by HistoryProcessor() function, which will take care of querying the history graph for the requested ontology's attributes present in the history graph linked to a date. For instance, for the history of price of a given product, each price returned from the history graph is linked to a date on which the offer started, this date being an attribute of the offer.

As previously mentioned in section 2, our effort and contribution aim to bring context-awareness respecting a semantic approach into publish/subscribe systems. To the regards of this effort, we created a knowledge base graph model based on ontologies modeled through JSON-LD. We also extended schema.org's vocabulary in order to answer our scenario "Apple distribution". Figure

12 on the next page gives an overview on how RCOS models ontologies for the knowledge base graph it uses.

Schema.org’s standard ontology vocabulary allows modeling a product and its attributes, such as name, description and type. A product can also include nested ontologies containing their own attributes. This is the case in RCOS; an apple ontology has an offer ontology that is associated with it. This offer ontology has attributes such as the highest price, the lowest, the average of these prices, and how many offers are included. The offer ontology also includes the individual offer, here represented as an array of offers in our knowledge base. An individual offer includes information such as the availability, the seller, the minimum ordering price and the eligible area for the offer.

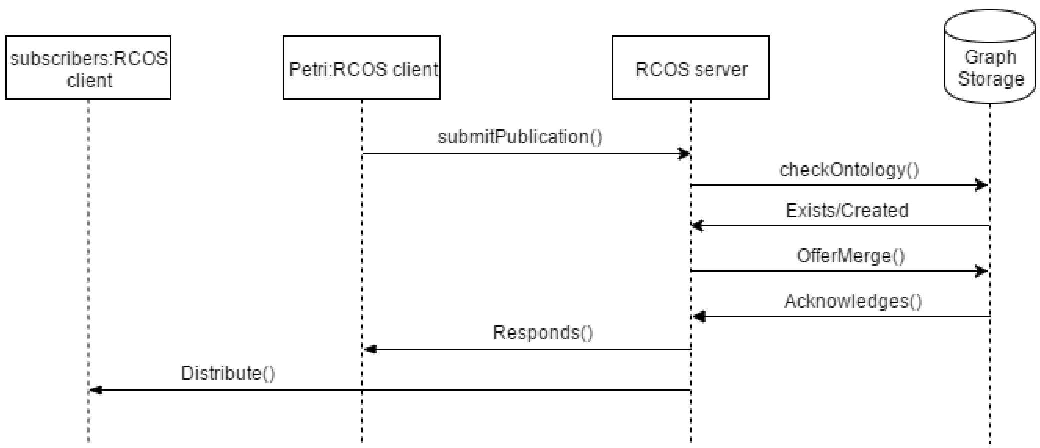
The extension we bring to this vocabulary are attributes starting with “**lfd:**”. The schema.org’s and other similar efforts are, for most, joint commercial effort. The apple ontology does not yet exist as such, to the extent of our knowledge, and has not been standardized either. In this ontology, which we modeled, we include new attributes such as vitamins, label, species and origin of the apple. These attributes, which we have introduced for the apple ontology, would also be valid for any edible product. In this regard, we perceive a need for further research on a “consumable” ontology, which could then be reviewed for standardization by the schema.org consortium.

```
{
  "@context": [
    "http://schema.org/",
    {
      "lfd": "http://example-localfood.org/"
    }
  ],
  "lfd:fruits": [
    {
      "@type": "lfd:Apple",
      "description": "Nice Red and Yellow Apple",
      "name": "Apple Fuji Bio",
      "lfd:species": "Fuji",
      "lfd:label": "FI-BIO-01",
      "lfd:taste": "sweet",
      "lfd:origin": "Finland",
      "nutrition": {
        "@type": "NutritionInformation",
        "calories": "39 calories",
        "carbohydrateContent": "8.16 g",
        "fatContent": "0.04 g",
        "fiberContent": "1.5 g",
        "proteinContent": "0.16 g",
        "saturatedFatContent": "0.01 g",
        "sodiumContent": "0.0 g",
        "sugarContent": "8.07 g",
        "unsaturatedFatContent": "0.03g"
      },
      "lfd:vitamins": {
        "@type": "lfd:Vitamins",
        "lfd:vitaminA": "4.46 ug",
        "lfd:vitaminC": "6.0 mg",
        "lfd:vitaminE": "0.209 mg"
      },
      "offers": {
        "@type": "lfd:AggregateOffer",
        "highPrice": "3.50",
        "lowPrice": "3.00",
        "lfd:averagePrice": "3.25",
        "offerCount": "4",
        "offers": [
          {
            "@type": "Offer",
            "priceCurrency": "EUR",
            "price": "3.50",
            "validFrom" : "2016-04-12T19:30+02:00",
            "validThrough" : "2016-07-12T19:30+02:00"
            "offeredBy": {
              "@type": "lfd:Producer",
              "@id": "http://kotiomena-lpr.fi",
              "name": "KotiOmena T. Kakuunen"
            },
            "priceSpecification": {
              "@type": "DeliveryChargeSpecification",
              "appliesToDeliveryMethod":
"http://purl.org/goodrelations/v1#DeliveryModeOwnFleet",
              "eligibleTransactionVolume": {
                "@type": "PriceSpecification",
                "price": "10.00",
                "priceCurrency": "EUR"
              },
              "eligibleRegion": {
                "@type": "GeoCircle",
                "address": {
                  "@type": "PostalAddress",
                  "addressCountry": "Finland",
                  "streetAddress": "Nuijamaantie 494",
                  "addressLocality": "Lappeenranta",
                  "name": "KotiOmena T. Kakuunen",
                  "postalCode": "53300"
                },
                "geoMidpoint": {
                  "@type": "GeoCoordinates",
                  "latitude": "61.038698",
                  "longitude": "28.362005"
                },
                "geoRadius": "10000"
              }
            }
          },
          {
            "@type": "Offer",
            "priceCurrency": "EUR",
            "price": "3.00",
            "validFrom": "2016-04-12T19:30+02:00",
            "validThrough": "2016-07-12T19:30+02:00",
            "offeredBy": {
              "@type": "lfd:Producer",
              "@id": "http://kuorttasenluomuomena-lpr.fi",
              "name": "Kuorttasen luomuomena"
...
}
```

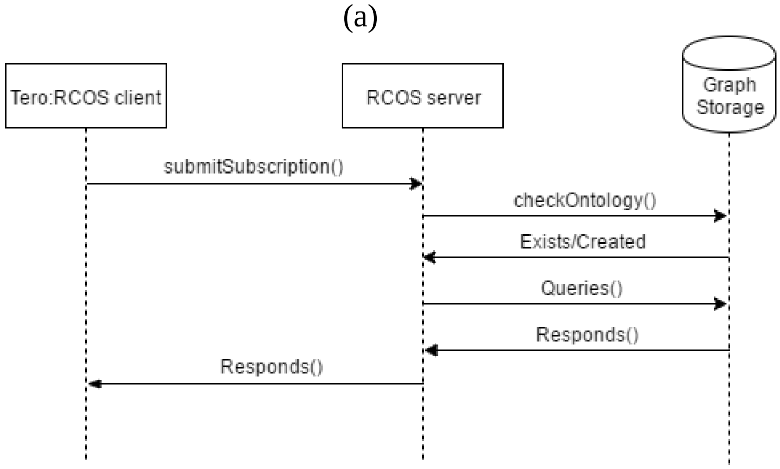
Figure 12. Ontologies representation for RCOS's knowledge base graph

4. Proof of Concept – Prototype implementation

Our proof-of-concept implementation is divided into two parts, and it is constituted of a prototype of RCOS, and a mobile prototype client communicating with RCOS. We focused our effort for this proof of concept on the possibility to emit a demand to RCOS, interpreting it and receiving the answer which we format visually in the mobile client application. The whole prototype accounts for an approximate amount of 1000 lines of code. Both the client and server are described in the following paragraphs. The below figure 13 relates our prototype to our use case, (a) represents a publication to the system, while (b) represents a subscription.



(a)

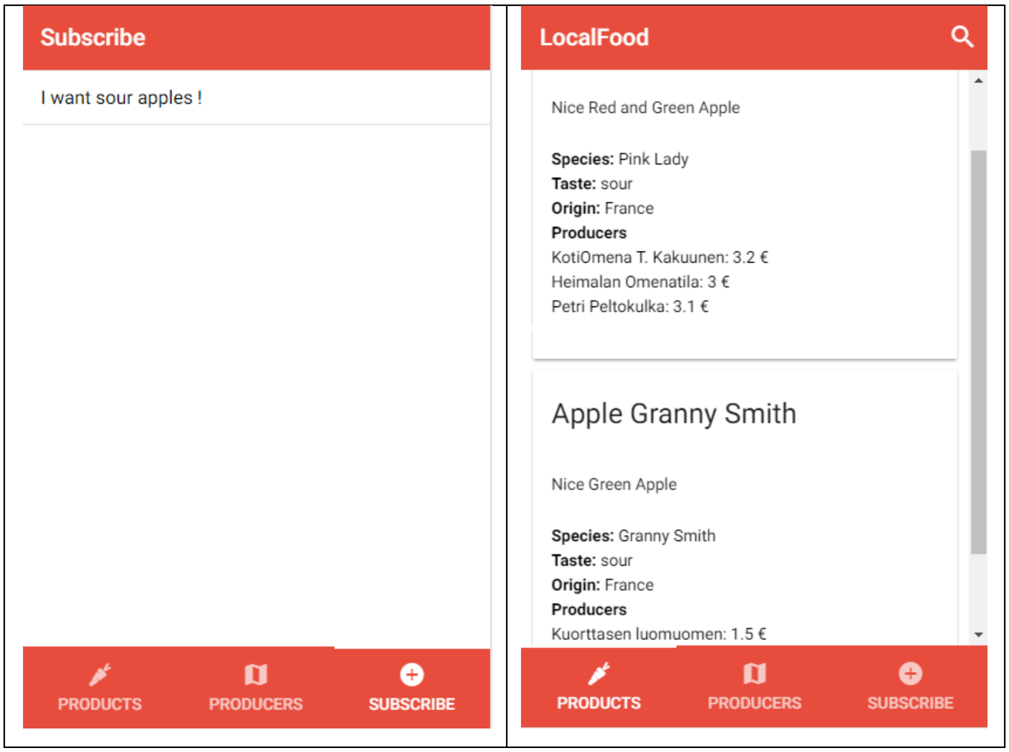


(b)

Figure 13. Sequence Diagram of RCOS's Proof of Concept

4.1. The RCOS client

We developed the RCOS client using the Ionic2 framework, which allows to use the Angular web technology to develop mobile applications. Our client allows us to emit demands (subscriptions) and display the ontologies resulting from the sub-graph transmitted by the RCOS server as a response. Figure 14 shows the mobile application client that we developed.



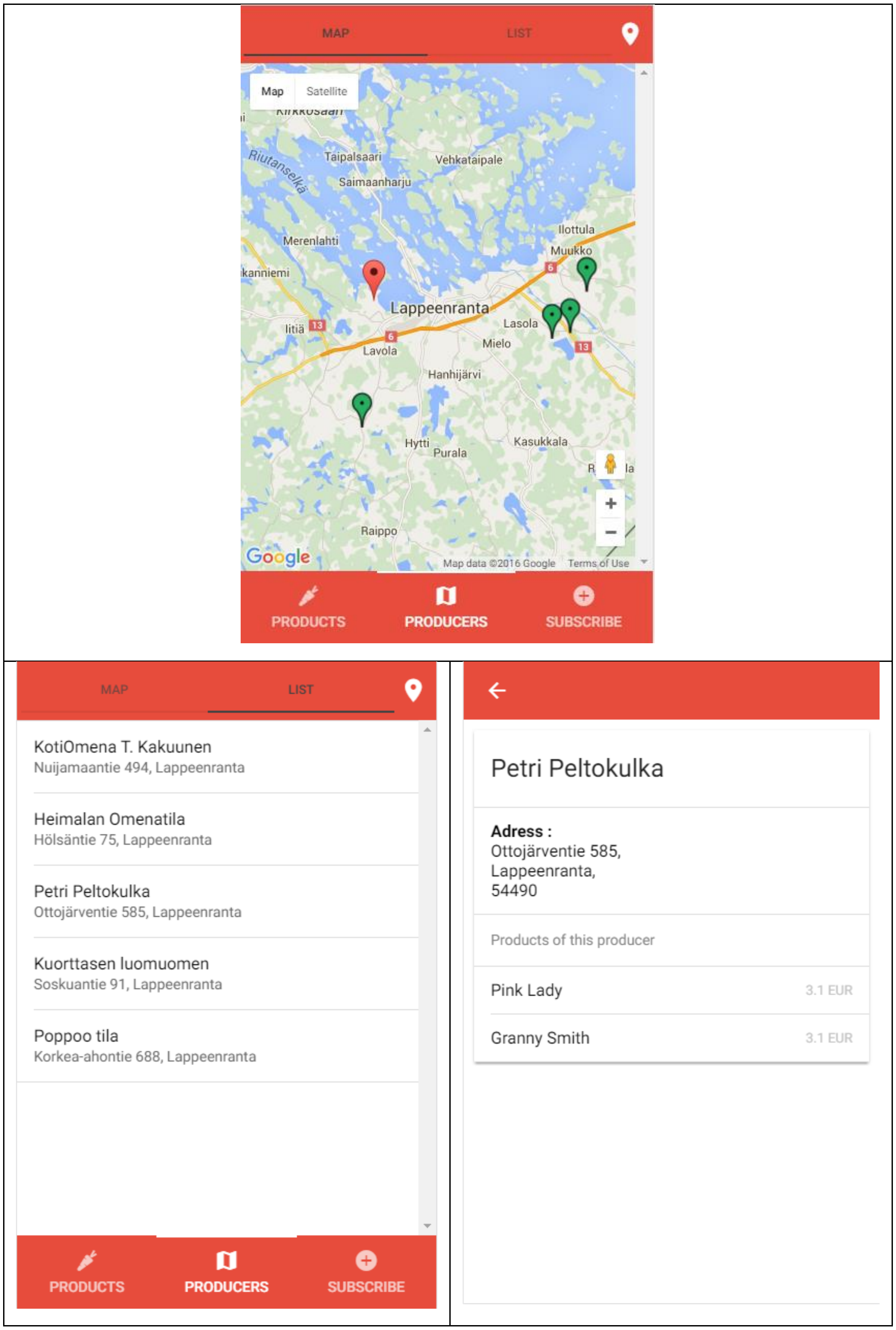


Figure 14. RCOS Proof-of-Concept's client

It follows the Model-View-Controller programming paradigm and is constituted of the following components:

- The *connectivity provider*

This component provides a way to check if the smart mobile device is connected to the Internet. This component is present to avoid errors in case of the lack of connectivity.

- The *data provider*

The *data provider* is the core component of our application as far as the data management is concerned. It handles the REST requests to RCOS server and provides an SQLITE local storage. This storage is used as local cache for our application, and allows us to save network load when transiting through the application. The RCOS server is only queried in case of a subscription and a manual refresh from the client. We consider including a push system in the next versions, but since this is a proof of concept, it relies on simple REST requests.

- The *Google maps provider*

This provider is responsible for handling the communication with Google maps and provides an API to our *producers view and controller*, which includes a map of the surrounding products.

- The *products view and controller*

The *products view* is the component that displays the apples resulting from our subscription. It relies on the *products controller*, which handles the needed communication invoking the *data provider* to query our local SQLITE storage.

- The *producers view and controller*

The *producers view* handles both the map view of the producers who have products matching our subscription, and the list. It relies on the *producers controller*, which handles the needed communication invoking the *data provider* to query our local SQLITE storage in order to retrieve the necessary data.

- The *subscription view and controller*

The *subscription view* is the component that displays the subscription view in our application. Since it is a proof of concept, it only handles subscriptions for sour apples in the Lappeenranta area at the moment, but this can be extended to a more generic set of choices. It relies on the *subscription controller*, which handles the needed communication invoking the *data provider* to in order to send the appropriate REST request to the RCOS server.

4.2. The RCOS server

The RCOS server relies on using Apache ActiveMQ Artemis 1.3 as an external tool. However, the component we add can directly be embedded into the broker of a publish/subscribe system. The history and CAROMM communication are not handled in the proof of concept. Currently, our prototype implementation handles demands on an ontology knowledge base. This ontology knowledge base is stored in a JSON-LD file and is loaded in memory as a graph by RCOS.

In order to create ontologies and place them in a graph for our knowledge base, we use the software tool Apache Jena 3.0.1. We have chosen this software due to its high interoperability and compatibility with the JSON-LD format. The main graph is loaded from a JSON-LD file and the extracting of the sub-graph used to answer a "demand" request is processed through the Jena SPARQL interpreter. An example of SPARQL query, which represents our case scenario "Apple distribution", is presented in the below figure 15.

```
PREFIX s: <http://schema.org/>
PREFIX lfd: <http://example-localfood.org/>
CONSTRUCT {
?a ?b ?c .
?d s:offers ?j .
?j s:price ?k .
?j s:offeredBy ?l .
?j s:priceSpecification ?o .
?o s:eligibleRegion ?p .
?p s:address ?q .
?q s:addressCountry ?r .
?q s:streetAddress ?x .
?q s:addressLocality ?t .
?q s:name ?u .
?q s:postalCode ?v .
?p s:geoMidpoint ?w .
?w s:latitude ?y .
?w s:longitude ?z .
?p s:geoRadius ?aa}
 WHERE {?a ?b ?c .
?a s:description ?e .
?a s:name ?f .
?a lfd:species ?g .
?a lfd:taste ?h .
?a lfd:origin ?i .
?a s:offers ?d .
?d s:offers ?j .
?j s:price ?k .
?j s:offeredBy ?l .
?j s:priceSpecification ?o .
?o s:eligibleRegion ?p .
?p s:address ?q .
?q s:addressCountry ?r .
?q s:streetAddress ?x .
?q s:addressLocality ?t .
?q s:name ?u .
?q s:postalCode ?v .
?p s:geoMidpoint ?w .
?w s:latitude ?y .
?w s:longitude ?z .
?p s:geoRadius ?aa .
?a lfd:taste "sour" .
?q s:addressLocality "Lappeenranta"}
```

Figure 15. SPARQL query sample used by RCOS

This query with nested elements allows us to obtain the full ontologies that present a sour taste, and whose seller's locality is Lappeenranta. It could be made generic independently to the subscription's attributes formatting by analyzing it and identifying parent nodes. However, we limited our current proof-of-concept to a static query for simplicity reason. We evaluate our prototype performances in the following section for larger scale deployment.

5. Evaluation of RCOS

In this section, we evaluate our prototype in terms of performance variation for a larger scale deployment. We want to know which factors influence the processing time the most when querying our knowledge base graph. In order to do that, we continuously inject ontologies into our knowledge base graph and do measurements while querying it. We have chosen to evaluate this part of our system, because it is the one that will be the most affected as the knowledge base grows in a large deployment case. Current publish/subscribe systems are already able to handle a significant amount of requests efficiently, so it is worth measuring the knowledge base graph querying. This allows us to understand the behavior of the processing time according to the evolution of the number of ontologies in it.

Evaluation setup

The tests in this section are made on a Lenovo G505S laptop running Windows 8. We are interested in understanding the performance variations, and not in measuring the performances themselves. Moreover, better performances may be obtained from a Linux system, and a better hardware. Each test is constituted of five series of a hundred measurements, which we averaged together. We inject ontologies that might or might not be matching the attribute restriction we include in our query request. We use the current timestamp in order to know how long the processing takes.

Evaluation of the matching process

In figure 16, we are interested in the performance of the matching process's behavior. The matching process is handled via SPARQL [61] CONTRUCT operation. This operation allows us to extract a sub-graph from an existing graph, according to restrictions we define in the query. During this evaluation, our knowledge base graph sample includes ten ontologies, two of which match lfd:taste = "sour" and eight lfd:taste = "sweet". The blue curve corresponds to the case where the SPARQL CONSTRUCT will create a sub-graph out of n-8 matching ontologies, n being the number of ontologies in the queried knowledge base graph and 8 being a static number of results used for realizing this test. The green curve corresponds to the case where SPARQL CONSTRUCT will create a sub-graph out of 8 matching ontologies for any size of the queried knowledge base graph. This measurement allows us to know, whether or not, the processing time difference between the two cases grows as the knowledge base grows.

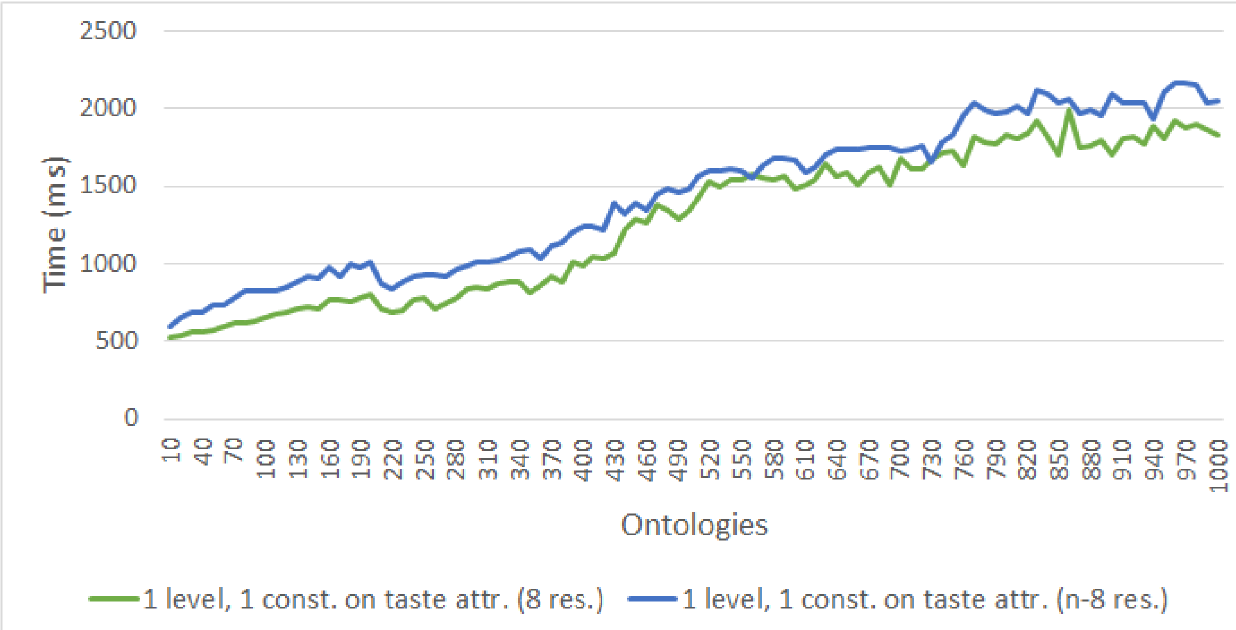


Figure 16. Processing time and number of results against number of ontologies

From these results, we consider the CONSTRUCT operation processing time to be negligible against the time it takes for the SPARQL processor to go through the knowledge base graph, in order to find the right ontologies matching the query. Due to this fact, and in order to obtain clear data on the impact of the number of ontologies in the knowledge base graph, the following tests (when they have a constraint) are made with a static n-8 ontologies result matching to the query.

We are firstly interested in knowing the actual influence of introducing a single constraint into the SPARQL query that is represented in figure 17. In this first case, we do not involve any nested elements (ontologies included into ontologies) in our knowledge base graph. Only the first level attributes of our ontologies are queried.

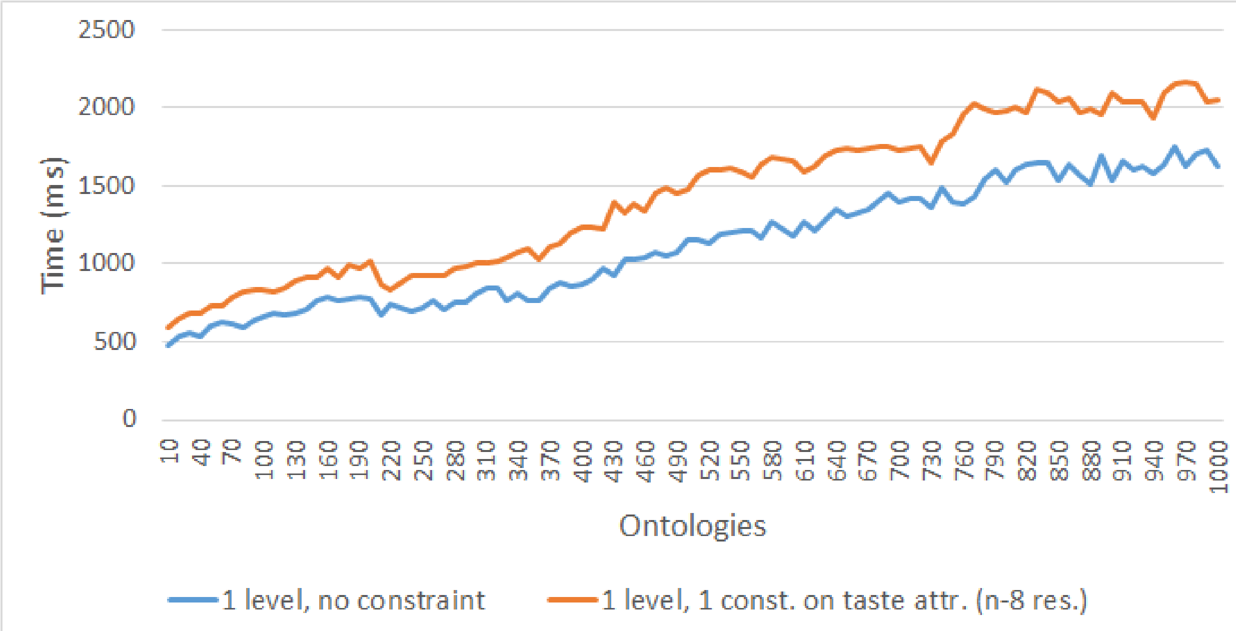


Figure 17. Processing time introducing a constraint on the taste attribute against number of ontologies

The blue line represents the case where we do not introduce a constraint in the query (n results), and the orange line the case where we limit the results to sour apples (n-8 results). As we can observe when introducing a constraint, the processing time of the query can increase by 20% and we can also observe that this trend is slightly increasing as the number of ontologies increases. This first test involves a single-level type of ontologies. Such ontologies are constituted of attributes, and cannot include any other ontology.

In figure 18, we observe the behavior of the processing time as we introduce new ontologies including six level of complexity. These ontologies are of the type represented in the figure 11 of section III. They involve complex computing operations for the SPARQL engine, resulting in a bigger processing task.

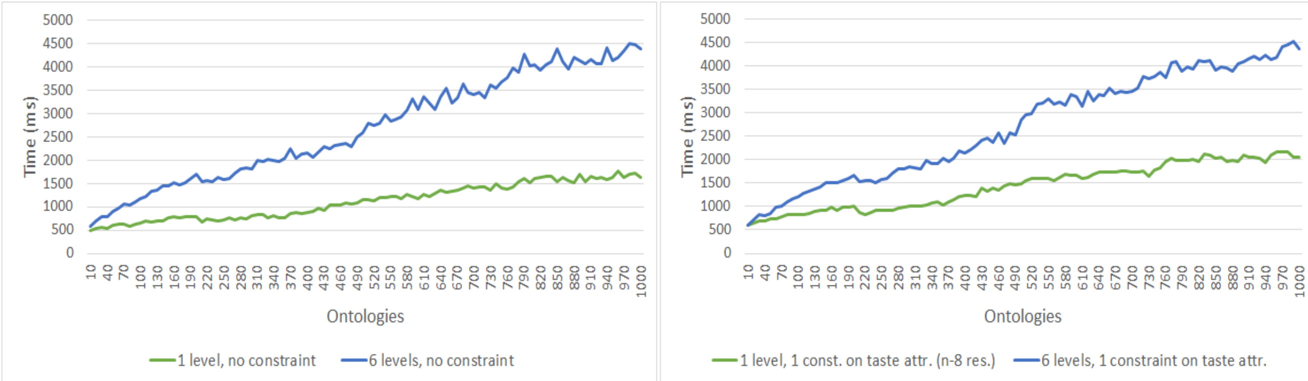


Figure 18. Processing time for one and six levels ontologies against number of ontologies

In the case where we do not have any constraint, the processing time of an ontology with six levels can be three times the one of an ontology which has a single-level. In the case of a constraint being introduced, the gap is lower but still significant.

From these measurements, we can conclude that the processing time variation follows a linear trend against the number of ontologies in our knowledge base graph. However, it grows significantly more as we introduce complex ontologies. Ontologies' complexity is the most affecting factor on the processing time as the number of ontologies grows. In a real case deployment, one might want to limit the complexity of ontologies, and favor two separated ontologies, which are linked together programmatically after the query, in a case of expecting a high number of ontologies in the knowledge base graph.

6. Conclusion and future work

In this thesis, we propose, develop and implement RCOS and its subscription language. RCOS is a real time context sharing system that aims to address a key problem currently experienced by food producers and customers. It fills the gap of knowledge on the design of a real time and contextual publish/subscribe system for the future Internet and Internet of Things, and brings a novel subscription language for it. This type of system enables producers to directly communicate with targeted consumers bypassing the need for an intermediate actor.

RCOS takes advantage of semantic web technologies, in particular an ontological representation, allowing producers and consumers to exchange context about various products. The context supported by RCOS includes location-based and personalization context. RCOS also incorporates a novel history feature, which allows to request for the history of an attribute, such as the price of a product.

To validate and evaluate the system, RCOS was implemented with our subscription language, based on JSON-LD, and the context-aware broker was integrated with the widely used publish/subscribe system Apache ActiveMQ Artemis. RCOS also incorporated the development of a mobile application that allows seamless exchange of context between the entities. Experiments evaluating the performance of the context-aware request matching indicate that the performance of RCOS is linear with the increasing number of requests. However, as the complexity of ontology representations increases, the processing resource usage follows the same trend.

Integrating context-classification (e.g. fuzzy logic) in order to interpret the weight of contextual information is an area in need of research. Currently, RCOS does not attribute weights to attributes and uses static interpretation of them. However, we consider researching how to interpret similar attributes' meaning within different context, and their consequent ontological classification, as being an essential point for future research in the field that needs to be investigate a priori to the context-classification. The optimal physical distribution of the context is also outside the scope of this thesis, but we acknowledge the need to investigate this area as well for the future. Our current assumption is an important security of the network communication. We, however, acknowledge the need for a full study of this area including authentication, encryption and privacy. End to end encryption would be an interesting paradigm to explore, Pallickara et al. [77] propose a framework for secure end-to-end delivery of messages in publish/subscribe systems, which could be implemented alongside with RCOS. Lastly, we acknowledge that such a broker combined with a web scraper could enable analysing the semantic web. It would also instantly share, in an automated way, new contexts to the relevant peers. Such an integration is another area to be researched.

References

[1] Villarreal, M. (2006, October 15). What is agrobiodiversity? - Module 1: Introduction of key concepts. Retrieved December 5, 2016, from http://www.fao.org/docrep/009/y5956e/Y5956E03.htm#ch1.1

[2] Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. M. (2003). The many faces of publish/subscribe. ACM Computing Surveys (CSUR), 35(2), 114-131.

[3] Campailla, A., Chaki, S., Clarke, E., Jha, S., & Veith, H. (2001, July). Efficient filtering in publish-subscribe systems using binary decision diagrams. In Proceedings of the 23rd International Conference on Software Engineering (pp. 443-452). IEEE Computer Society.

[4] Loke, S. W., & Zaslavsky, A. (2003, October). Communicative acts of Elvin-enhanced mobile agents. In Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on (pp. 446-449). IEEE.

[5] Antonić, A., Marjanović, M., Pripužić, K., & Žarko, I. P. (2016). A mobile crowd sensing ecosystem enabled by CUPUS: cloud-based publish/subscribe middleware for the internet of things. Future Generation Computer Systems, 56, 607-622.

[6] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. Scientific american, 284(5), 28-37.

[7] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., & Lindström, N. (2014). JSON-LD 1.0. W3C Recommendation (January 16, 2014).

[8] Oki, B., Pfluegl, M., Siegel, A., & Skeen, D. (1994, January). The Information Bus: an architecture for extensible distributed systems. In ACM SIGOPS Operating Systems Review (Vol. 27, No. 5, pp. 58-68). ACM.

[9] Altherr, M., Erzberger, M., & Maffeis, S. (1999, October). iBus-a software bus middleware for the Java platform. In Proceedings of the International Workshop on Reliable Middleware Systems (pp. 43-53).

[10] Castro, M., Druschel, P., Kermarrec, A. M., & Rowstron, A. I. (2002). SCRIBE: A large-scale and decentralized application-level multicast infrastructure. Selected Areas in Communications, IEEE Journal on, 20(8), 1489-1499.

[11] Object Management Group. (1995). The Common Object Request Broker (CORBA): Architecture and Specification. Object Management Group.

[12] Rosenblum, D. S., & Wolf, A. L. (1997). A design framework for Internet-scale event observation and notification (Vol. 22, No. 6, pp. 344-360). ACM.

[13] Cugola, G., Di Nitto, E., & Fuggetta, A. (1998, April). Exploiting an event-based infrastructure to develop complex distributed systems. In Software Engineering, 1998. Proceedings of the 1998 International Conference on (pp. 261-270). IEEE.

[14] Pereira, J., Fabret, F., Llirbat, F., Preotiuc-Pietro, R., Ross, K. A., & Shasha, D. (2000, September). Publish/subscribe on the web at extreme speed. In VLDB (pp. 627-630).

[15] Gruber, R. E., Krishnamurthy, B., & Panagos, E. (1999, May). The architecture of the READY event notification service. In icdcs (p. 0108). IEEE.

[16] Parzyjegla, H., Graff, D., Schröter, A., Richling, J., & Mühl, G. (2010). Design and implementation of the rebeca publish/subscribe middleware. In From active data management to event-based systems and more (pp. 124-140). Springer Berlin Heidelberg.

[17] Pietzuch, P. R., & Bacon, J. M. (2002). Hermes: A distributed event-based middleware architecture. In Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on (pp. 611-618). IEEE.

[18] Segall, B., & Arnold, D. (1997). Elvin has left the building: A publish/subscribe notification service with quenching. Proceedings of the I997 Australian UNLX Users Group (A UUG'I997), 243-255.

[19] Aitenbichler, E., Kangasharju, J., & Mühlhäuser, M. (2007). MundoCore: A light-weight infrastructure for pervasive computing. Pervasive and Mobile Computing, 3(4), 332-361.

[20] Chand, R., & Felber, P. (2004, October). XNET: a reliable content-based publish/subscribe system. In Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on (pp. 264-273). IEEE.

[21] Chand, R., & Felber, P. (2005). Semantic peer-to-peer overlays for publish/subscribe networks. In Euro-Par 2005 Parallel Processing (pp. 1194-1204). Springer Berlin Heidelberg.

[22] Sivaharan, T., Blair, G., & Coulson, G. (2005). Green: A configurable and re-configurable publish-subscribe middleware for pervasive computing. In On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE (pp. 732-749). Springer Berlin Heidelberg.

[23] Tarkoma, S., & Nokia, N. R. C. (2009). Distributed event routing in publish/subscribe communication systems. Middleware for Network Eccentric and Mobile Applications, Springer, 219-244.

[24] Eugster, P. T. (2001). Type-based publish/subscribe (Doctoral dissertation, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE).

[25] Eugster, P. T., Guerraoui, R., & Damm, C. H. (2001, October). On objects and events. In ACM SIGPLAN Notices (Vol. 36, No. 11, pp. 254-269). ACM.

[26] Buchmann, A. P., & Moody, K. An Active Functionality Service for Open Distributed Heterogeneous Environments.

[27] Cugola, G., Cote, D., & Munoz, J. E. (2005, June). On introducing location awareness in publish-subscribe middleware. In Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on (pp. 377-382). IEEE.

[28] Fiege, L., Gärtner, F. C., Kasten, O., & Zeidler, A. (2003, June). Supporting mobility in content-based publish/subscribe middleware. In Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (pp. 103-122). Springer-Verlag New York, Inc..

[29] Eugster, P., & Holzer, A. (2008, January). Design and implementation of the pervaho middleware for mobile context-aware applications. In e-Technologies, 2008 International MCETECH Conference on (pp. 125-135). IEEE

[30] Loke, S. W., Padovitz, A., & Zaslavsky, A. (2003, January). Context-based addressing: The concept and an implementation for large-scale mobile agent systems using publish-subscribe event notification. In Distributed Applications and Interoperable Systems (pp. 274-284). Springer Berlin Heidelberg.

[31] Padovitz, A., Loke, S. W., & Zaslavsky, A. (2008). The ECORA framework: A hybrid architecture for context-oriented pervasive computing. Pervasive and mobile computing, 4(2), 182-215

[32] Matthew Phillips. (2010, July 29). Home - Avis. Retrieved March 10, 2016, from http://avis.sourceforge.net/

[33] Tarkoma, S., Lindholm, T., & Kangasharju, J. (2005). Collection and object synchronization based on context information. In Mobility Aware Technologies and Applications (pp. 240-251). Springer Berlin Heidelberg.

[34] Cugola, G., Margara, A., & Migliavacca, M. (2009, July). Context-aware publish-subscribe: Model, implementation, and evaluation. In Computers and Communications, 2009. ISCC 2009. IEEE Symposium on (pp. 875-881). IEEE.

[35] Zahariadis, T., Papadakis, A., Alvarez, F., Gonzalez, J., Lopez, F., Facca, F., & Al-Hazmi, Y. (2014, December). FIWARE lab: managing resources and services in a cloud federation supporting future internet applications. In Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on (pp. 792-799). IEEE.

[36] The Apache Software Foundation. (2016, Januay 8). Apache ActiveMQ - ActiveMQ Atemis. Retrieved March 10, 2016, from https://activemq.apache.org/artemis/

[37] The Apache Software Foundation. (2015, Ocober 16). Apache ActiveMQ - REST. Retrieved March 10, 2016, from http://activemq.apache.org/rest.html

[38] The Apache Software Foundation. (2015, April). Apache ActiveMQ - Apache ActiveMQ Board Report - 2015.04 (April). Retrieved March 10, 2016, from http://activemq.apache.org/apache-activemq-board-report-201504-april.html

[39] The Apache Software Foundation. (2015, May 29). Apache ActiveMQ - Migration to ActiveMQ Atemis. Retrieved March 10, 2016, from https://activemq.apache.org/artemis/migration.html

[40] Carzaniga, A., Rosenblum, D. S., & Wolf, A. L. (2001). Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems (TOCS), 19(3), 332-383.

[41] Oracle and/or its affiliates (2013. January 11). The Java EE 6 Tutorial – Overview of JMS API. Retrieved on April 25, 2016, from https://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html

[42] Barker, P., & Campbell, L. M. (2014). What is schema.org?. LRMI. Retrieved April, 21, 2015.

[43] Gil, B., & Trezentos, P. (2011, July). Impacts of data interchange formats on energy consumption and performance in smartphones. In Proceedings of the 2011 workshop on open source and design of communication (pp. 1-6). ACM.

[44] Singhal, A. (2012). Introducing the knowledge graph: things, not strings. Official Google Blog, May.

[45] Cugola, G., Nitto, E. D., & Picco, G. P. (2000). Content-based dispatching in a mobile environment. Proceedings of WSDAAL, 2000.

[46] Huang, Y., & Garcia-Molina, H. (2001, May). Publish/subscribe in a mobile enviroment. In Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access (pp. 27-34). ACM.

[47] Fiege, L., Gärtner, F. C., Kasten, O., & Zeidler, A. (2003, June). Supporting mobility in content-based publish/subscribe middleware. In Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (pp. 103-122). Springer-Verlag New York, Inc..

[48] Cugola, G., Cote, D., & Munoz, J. E. (2005, June). On introducing location awareness in publish-subscribe middleware. In Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on (pp. 377-382). IEEE.

[49] Salvador, Z., Lafuente, A., & Larrea, M. (2012). Design and Evaluation of a Publish/Subscribe Framework for Ubiquitous Systems. In Mobile and Ubiquitous Systems: Computing, Networking, and Services (pp. 50-63). Springer Berlin Heidelberg.

[50] Salvador, Z., Larrea, M., & Lafuente, A. (2012, August). Phoenix: A Protocol for Seamless Client Mobility in Publish/Subscribe. In Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on (pp. 111-120). IEEE.

[51] Podnar Zarko, I., Antonic, A., & Pripužic, K. (2013, September). Publish/subscribe middleware for energy-efficient mobile crowdsensing. In Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (pp. 1099-1110). ACM.

[52] Antonic, A., Roankovic, K., Marjanovic, M., Pripuic, K., & Zarko, I. P. (2014, August). A mobile crowdsensing ecosystem enabled by a cloud-based publish/subscribe middleware. In Future Internet of Things and Cloud (FiCloud), 2014 International Conference on (pp. 107-114). IEEE.

[53] Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J. P., Riahi, M., ... & Skorin-Kapov, L. (2015). Openiot: Open source internet-of-things in the cloud. In Interoperability and Open-Source Solutions for the Internet of Things (pp. 13-25). Springer International Publishing.

[54] Lassila, O., & Swick, R. R. (1999). Resource description framework (RDF) model and syntax specification.

[55] Brickley, D., & Guha, R. V. (2014). RDF Schema 1.1. W3C Recommendation, 25, 2004-2014.

[56] Bechhofer, S. (2009). OWL: Web ontology language. In Encyclopedia of Database Systems (pp. 2008-2009). Springer US.

[57] Lanthaler, M., & Gütl, C. (2012, April). On using JSON-LD to create evolvable RESTful services. In Proceedings of the Third International Workshop on RESTful Design (pp. 25-32). ACM.

[58] Carroll, J. J., & Stickler, P. (2004, May). RDF triples in XML. In Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (pp. 412-413). ACM.

[59] Bizer, C., & Cyganiak, R. (2014). RDF 1.1 TriG. W3C recommendation, 110.

[60] World Wide Web Consortium. (2014). RDF 1.1 Turtle: terse RDF triple language.

[61] Prud'Hommeaux, E., & Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, 15.

[62] Beckett, D., & McBride, B. (2004). RDF/XML syntax specification (revised). W3C recommendation, 10.

[63] Adida, B., Birbeck, M., McCarron, S., & Pemberton, S. (2008). RDFa in XHTML: Syntax and processing. Recommendation, W3C, 7.

[64] Berners-Lee, T., & Connolly, D. (1998). Notation3 (N3): A readable RDF syntax. W3C Team Submission: http://www.w3.org/TeamSubmission, (3).

[65] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web, 5(2), 51-53.

[66] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., & Lindström, N. (2014). JSON-LD 1.0. W3C Recommendation (January 16, 2014).

[67] OpenWines working group. (2015, October 2). OpenWines – Open-Data, for vineyards, winegrowers and wines. Retrieved on April 25, 2016, from http://openwines.eu/

[68] Ronan Guilloux. (2015, Jully 19). Github OpenWines – Ontology draft for Winemaker. Retrieved on April 25, 2016, from https://github.com/OpenWines/Ontology/blob/master/1.0/Winemaker.md

[69] Rowstron, A., & Druschel, P. (2001, November). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001 (pp. 329-350). Springer Berlin Heidelberg.

[70] Cugola, G., Di Nitto, E., & Fuggetta, A. (2001). The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. Software Engineering, IEEE Transactions on, 27(9), 827-850.

[71] Zeidler, A., & Fiege, L. (2003, May). Mobility support with REBECA. In Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on (pp. 354-360). IEEE.

[72] Sherchan, W., Jayaraman, P. P., Krishnaswamy, S., Zaslavsky, A., Loke, S., & Sinha, A. (2012, July). Using on-the-move mining for mobile crowdsensing. In Mobile Data Management (MDM), 2012 IEEE 13th International Conference on (pp. 115-124). IEEE.

[73] Jayaraman, P. P., Sinha, A., Sherchan, W., Krishnaswamy, S., Zaslavsky, A., Haghighi, P. D., ... & Do, M. T. (2012, June). Here-n-now: A framework for context-aware mobile crowdsensing. In Proc. of the Tenth International Conference on Pervasive Computing.

[74] Haghighi, P. D., Krishnaswamy, S., Zaslavsky, A., Gaber, M. M., Sinha, A., & Gillick, B. (2013). Open mobile miner: a toolkit for building situation-aware data mining applications. Journal of Organizational Computing and Electronic Commerce, 23(3), 224-248.

[75] Padovitz, A., Loke, S. W., & Zaslavsky, A. (2004, March). Towards a theory of context spaces. In Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on (pp. 38-42). IEEE.

[76] Padovitz, A., Loke, S. W., & Zaslavsky, A. (2008). Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 38(4), 729-742.

[77] Pallickara, S., Pierce, M., Gadgil, H., Fox, G., Yan, Y., & Huang, Y. (2006, September). A framework for secure end-to-end delivery of messages in publish/subscribe systems. In Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (pp. 215-222). IEEE Computer Society.

[78] Klimova, A., Rondeau, E., Andersson, K., Porras, J., Rybin, A., & Zaslavsky, A. (2016). "An international Master's program in green ICT as a contribution to sustainable development, Journal of Cleaner Production

Appendix 1. Raw evaluation results

1 level in ontology – 0 constraint and n results

Number of nodes	Measurement 1	Measurement 2	Measurement 3	Measurement 4	Measurement 5	Average
10	484	483	487	482	461	479.4
20	508	539	542	529	561	535.8
30	534	574	564	568	572	562.4
40	528	527	549	521	547	534.4
50	568	588	639	590	617	600.4
60	652	624	590	612	632	622
70	624	616	598	610	628	615.2
80	563	575	614	596	588	587.2
90	694	626	610	623	612	633
100	667	644	641	670	657	655.8
110	698	697	680	664	684	684.6
120	618	736	676	701	659	678
130	724	678	646	676	712	687.2
140	710	701	702	683	729	705
150	732	742	761	794	778	761.4
160	844	770	757	772	773	783.2
170	834	775	782	744	699	766.8
180	759	725	788	744	858	774.8
190	759	752	899	774	774	791.6
200	751	686	789	853	804	776.6
210	684	709	636	691	625	669
220	661	742	785	723	774	737
230	682	673	705	749	778	717.4
240	668	726	750	695	630	693.8
250	691	746	754	645	766	720.4
260	695	780	824	826	696	764.2
270	691	701	765	706	676	707.8
280	798	730	825	678	753	756.8
290	676	865	722	741	754	751.6
300	698	757	785	801	990	806.2
310	707	910	924	888	775	840.8
320	825	874	929	777	790	839
330	732	764	798	757	760	762.2
340	786	776	807	869	788	805.2

350	735	843	760	759	727	764.8
360	793	762	763	736	790	768.8
370	841	1021	725	800	852	847.8
380	814	950	880	850	912	881.2
390	763	876	977	824	833	854.6
400	850	849	809	855	968	866.2
410	868	884	820	1014	928	902.8
420	929	983	1104	894	927	967.4
430	825	1045	844	1018	919	930.2
440	1069	1059	957	1024	1029	1028
450	1039	1063	1027	934	1084	1029
460	1040	998	1060	1105	991	1039
470	1003	1083	1063	1186	1021	1071
480	988	1082	1043	1092	1041	1049
490	927	1153	940	1254	1109	1077
500	1153	1128	1023	1348	1118	1154
510	1055	1137	1068	1373	1166	1160
520	1132	1087	1228	1081	1108	1127
530	1233	1320	1214	1207	991	1193
540	1174	1023	1459	1091	1280	1205
550	1164	1116	1435	1135	1218	1214
560	1097	1116	1462	1245	1142	1212
570	1039	1161	1471	1181	970	1164
580	1298	1138	1501	1205	1181	1265
590	1104	1179	1565	1210	1078	1227
600	1244	1102	1413	1098	1051	1182
610	1400	1110	1333	1299	1192	1267
620	1175	1244	1408	1130	1107	1213
630	1377	1365	1351	1155	1160	1282
640	1215	1335	1431	1563	1183	1345
650	1410	1462	1404	1067	1184	1305
660	1373	1381	1396	1120	1346	1323
670	1336	1418	1475	1178	1322	1346
680	1594	1487	1534	1073	1345	1407
690	1481	1433	1444	1627	1285	1454
700	1394	1408	1481	1483	1226	1398
710	1441	1490	1321	1474	1377	1421
720	1334	1408	1358	1391	1580	1414
730	1150	1355	1470	1352	1458	1357
740	1464	1597	1394	1353	1619	1485
750	1363	1457	1500	1330	1336	1397

760	1521	1390	1379	1311	1327	1386
770	1466	1388	1456	1520	1313	1429
780	1544	1514	1685	1447	1532	1544
790	1576	1603	1525	1602	1711	1603
800	1639	1402	1476	1558	1524	1520
810	1443	1805	1617	1644	1490	1600
820	1651	1594	1627	1548	1779	1640
830	1611	1718	1760	1561	1573	1645
840	1718	1654	1618	1640	1614	1649
850	1403	1510	1493	1454	1791	1530
860	1656	1538	1588	1730	1665	1635
870	1430	1464	1432	1651	1843	1564
880	1533	1542	1457	1487	1519	1508
890	1785	1548	1648	1739	1768	1698
900	1395	1617	1508	1554	1571	1529
910	1519	1864	1586	1778	1548	1659
920	1770	1481	1430	1764	1555	1600
930	1560	1671	1742	1559	1585	1623
940	1667	1608	1685	1480	1466	1581
950	1766	1564	1742	1651	1444	1633
960	1882	1843	1784	1612	1666	1757
970	1581	1542	1601	1729	1686	1628
980	1399	1813	1696	1848	1773	1706
990	1839	1738	1582	1733	1757	1730
1000	1621	1513	1632	1636	1743	1629

1 level in ontology – 1 constraint on the taste attribute and n-8 results

Number of nodes	Measurement 1	Measurement 2	Measurement 3	Measurement 4	Measurement 5	Average
10	741	740	487	494	503	593
20	827	841	522	526	545	652.2
30	870	878	573	544	557	684.4
40	931	852	549	527	562	684.2
50	889	979	561	601	646	735.2
60	931	941	567	616	591	729.2
70	1018	1019	623	666	599	785
80	1115	1032	659	634	660	820
90	1204	1016	673	647	604	828.8
100	1140	1054	705	633	624	831.2

110	1046	1035	696	666	671	822.8
120	1103	1063	686	730	667	849.8
130	1125	1119	667	727	793	886.2
140	1106	1356	697	707	719	917
150	1140	1213	709	753	726	908.2
160	1175	1250	822	888	744	975.8
170	1146	1221	702	777	728	914.8
180	1319	1276	806	800	778	995.8
190	1349	1318	772	656	781	975.2
200	1337	1423	816	685	801	1012.4
210	1051	1312	660	716	619	871.6
220	1032	1018	827	673	634	836.8
230	1038	1087	864	665	759	882.6
240	1086	1132	898	755	735	921.2
250	1127	1096	956	685	761	925
260	1088	1275	774	806	698	928.2
270	1159	1193	762	751	749	922.8
280	1297	1315	782	751	693	967.6
290	1417	1172	816	788	714	981.4
300	1426	1209	874	772	752	1006.6
310	1423	1238	795	846	736	1007.6
320	1323	1283	816	829	835	1017.2
330	1442	1255	870	855	787	1041.8
340	1471	1396	847	842	817	1074.6
350	1297	1432	893	1000	841	1092.6
360	1386	1356	815	797	784	1027.6
370	1453	1337	890	906	962	1109.6
380	1626	1487	951	807	808	1135.8
390	1372	1505	1161	998	973	1201.8
400	1425	1639	985	1184	956	1237.8
410	1513	1420	954	1429	875	1238.2
420	1586	1521	982	1120	885	1218.8
430	1665	1721	1468	1139	987	1396
440	1703	1486	1403	986	1059	1327.4
450	1680	1803	1307	1049	1110	1389.8
460	1710	1726	1294	1018	971	1343.8
470	2035	1567	1402	1157	1092	1450.6
480	1818	1946	1437	1109	1107	1483.4
490	1840	1703	1468	1280	1001	1458.4
500	1877	1783	1475	1181	1091	1481.4
510	1903	1613	1650	1521	1132	1563.8

520	1960	1997	1589	1467	992	1601
530	2000	2093	1505	1233	1182	1602.6
540	1982	2083	1559	1170	1250	1608.8
550	1960	2072	1600	1191	1146	1593.8
560	2012	1833	1172	1259	1503	1555.8
570	1940	2024	1326	1248	1622	1632
580	2087	2005	1447	1283	1593	1683
590	2031	2091	1505	1188	1557	1674.4
600	2110	1817	1429	1244	1713	1662.6
610	2060	2050	1170	1246	1433	1591.8
620	1985	1913	1254	1377	1584	1622.6
630	1964	1987	1377	1568	1596	1698.4
640	2064	1954	1503	1572	1572	1733
650	2231	2035	1503	1647	1282	1739.6
660	2068	2108	1611	1594	1285	1733.2
670	2295	1954	1559	1433	1478	1743.8
680	1942	2237	1644	1402	1523	1749.6
690	1937	1820	1538	1674	1783	1750.4
700	1979	2042	1475	1584	1557	1727.4
710	2058	2151	1337	1546	1589	1736.2
720	2144	2035	1274	1680	1649	1756.4
730	2074	2042	1206	1483	1466	1654.2
740	2000	2226	1368	1705	1609	1781.6
750	2320	1977	1395	1824	1642	1831.6
760	1925	2317	1929	1893	1730	1958.8
770	2266	2401	1763	1908	1829	2033.4
780	2144	2402	1867	1626	1907	1989.2
790	2355	2328	2007	1280	1899	1973.8
800	2039	2509	1869	1705	1783	1981
810	2433	2248	1856	1679	1834	2010
820	2309	2376	1730	1883	1559	1971.4
830	2180	2633	2031	1791	1957	2118.4
840	2425	2397	1917	1793	1933	2093
850	2397	2316	1831	1890	1755	2037.8
860	2391	2323	1905	1861	1830	2062
870	2235	2331	1813	1825	1635	1967.8
880	2056	2304	1894	1835	1869	1991.6
890	2169	2274	1740	1765	1838	1957.2
900	2436	2419	1849	1893	1875	2094.4
910	2218	2371	1952	1832	1842	2043
920	2355	2022	1936	1967	1929	2041.8

930	2412	2349	1933	1658	1848	2040
940	2037	2464	1623	1817	1712	1930.6
950	2350	2334	1963	1920	1947	2102.8
960	2388	2495	1948	2086	1877	2158.8
970	2362	2562	1965	1918	2005	2162.4
980	2520	2458	1934	1818	2059	2157.8
990	2247	2476	1952	1635	1905	2043
1000	2473	2463	1853	1942	1525	2051.2

1 level in ontology – 1 constraint on the taste attribute and 8 results

Number of nodes	Mewasurement 1	Measurement 2	Measurement 3	Measurement 4	Measurement 5	Average
10	559	522	494	593	478	529.2
20	526	554	537	523	525	533
30	551	578	543	552	563	557.4
40	545	538	565	558	570	555.2
50	545	553	590	600	566	570.8
60	623	579	609	595	584	598
70	609	606	623	615	622	615
80	608	602	650	601	641	620.4
90	660	639	661	611	599	634
100	615	634	652	665	683	649.8
110	661	706	662	681	696	681.2
120	685	656	727	689	706	692.6
130	696	705	691	748	708	709.6
140	750	702	706	736	722	723.2
150	719	663	690	765	700	707.4
160	813	756	781	721	774	769
170	798	784	733	765	737	763.4
180	794	763	778	720	746	760.2
190	849	800	735	769	766	783.8
200	794	790	803	750	874	802.2
210	726	700	705	728	712	714.2
220	747	701	640	636	686	682
230	813	669	654	708	672	703.2
240	785	765	708	781	772	762.2
250	808	749	757	830	775	783.8
260	673	773	698	668	765	715.4
270	687	722	801	765	727	740.4

280	748	770	772	775	838	780.6
290	805	829	804	906	829	834.6
300	861	829	896	765	892	848.6
310	817	920	882	788	782	837.8
320	875	846	860	889	913	876.6
330	843	934	903	878	853	882.2
340	959	934	855	877	780	881
350	863	718	870	844	802	819.4
360	870	920	824	760	925	859.8
370	962	922	949	807	931	914.2
380	875	957	848	990	755	885
390	971	978	907	958	1218	1006.4
400	1045	992	934	816	1126	982.6
410	956	941	970	1048	1289	1040.8
420	940	955	1014	951	1317	1035.4
430	909	1173	960	1009	1284	1067
440	993	1277	1315	1058	1420	1212.6
450	1140	1085	1406	1345	1451	1285.4
460	972	1276	1404	1361	1299	1262.4
470	1178	1433	1326	1586	1352	1375
480	1124	1472	1317	1412	1411	1347.2
490	1129	1443	1267	1137	1438	1282.8
500	1154	1375	1336	1678	1174	1343.4
510	1469	1409	1369	1351	1533	1426.2
520	1682	1473	1368	1532	1620	1535
530	1532	1348	1586	1405	1632	1500.6
540	1545	1541	1465	1551	1594	1539.2
550	1794	1497	1548	1493	1369	1540.2
560	1611	1516	1573	1648	1533	1576.2
570	1437	1348	1734	1728	1537	1556.8
580	1557	1497	1425	1571	1656	1541.2
590	1521	1609	1515	1678	1510	1566.6
600	1525	1421	1395	1503	1586	1486
610	1486	1315	1384	1634	1697	1503.2
620	1430	1615	1506	1416	1740	1541.4
630	1516	1880	1751	1691	1377	1643
640	1666	1611	1624	1383	1509	1558.6
650	1570	1646	1638	1543	1531	1585.6
660	1757	1480	1502	1548	1273	1512
670	1665	1501	1606	1658	1525	1591
680	1637	1598	1571	1675	1647	1625.6

690	1559	1348	1740	1432	1453	1506.4
700	1530	1712	1648	1648	1873	1682.2
710	1754	1447	1618	1595	1619	1606.6
720	1434	1749	1634	1572	1679	1613.6
730	1694	1571	1644	1850	1608	1673.4
740	1692	1673	1710	1894	1595	1712.8
750	1794	1488	1900	1725	1707	1722.8
760	1960	1307	1547	1522	1811	1629.4
770	1588	1817	1712	2085	1898	1820
780	2013	1750	1709	1740	1682	1778.8
790	1913	1475	1829	1893	1747	1771.4
800	1841	2005	1871	1678	1739	1826.8
810	1737	1759	1902	2016	1634	1809.6
820	2053	1798	1963	1734	1631	1835.8
830	1849	1992	2108	1893	1772	1922.8
840	1849	1702	2015	1842	1669	1815.4
850	1767	1921	1316	1873	1628	1701
860	1967	2038	1935	2140	1874	1990.8
870	1622	1549	1881	1751	1944	1749.4
880	1689	1516	1875	1953	1751	1756.8
890	1677	1769	1740	1933	1844	1792.6
900	1381	1743	1628	1811	1946	1701.8
910	1662	1930	1759	1714	1996	1812.2
920	1756	1952	1606	1852	1913	1815.8
930	1782	1689	1988	1578	1844	1776.2
940	1966	1854	2044	1626	1956	1889.2
950	1779	1874	1667	1885	1844	1809.8
960	1879	1952	1799	1990	1987	1921.4
970	2060	1885	1919	1840	1693	1879.4
980	1785	1820	2096	1875	1925	1900.2
990	1755	1871	1751	1860	2060	1859.4
1000	1815	1896	1794	1680	1985	1834

Full ontology – 0 constraint and n results

Number of nodes	Mewasuremen t 1	Measuremen t 2	Measuremen t 3	Measuremen t 4	Measuremen t 5	Average
10	581	600	572	582	592	585.4
20	711	673	666	705	720	695
30	761	754	779	790	792	775.2

40	838	721	768	816	841	796.8
50	1005	811	907	870	898	898.2
60	947	922	949	996	1017	966.2
70	1117	1019	1053	1025	1120	1066.8
80	1078	1004	1006	1098	1013	1039.8
90	1080	1145	1126	1076	1100	1105.4
100	1219	1196	1067	1121	1253	1171.2
110	1307	1242	1188	1117	1233	1217.4
120	1424	1283	1183	1361	1371	1324.4
130	1255	1386	1393	1346	1391	1354.2
140	1298	1674	1429	1421	1389	1442.2
150	1502	1330	1489	1594	1325	1448
160	1609	1368	1524	1557	1511	1513.8
170	1539	1579	1571	1292	1406	1477.4
180	1503	1712	1368	1386	1615	1516.8
190	1637	1530	1629	1536	1642	1594.8
200	1777	1762	1613	1730	1604	1697.2
210	1381	1612	1432	1550	1659	1526.8
220	1678	1736	1366	1483	1513	1555.2
230	1676	1534	1528	1559	1379	1535.2
240	1999	1474	1628	1431	1665	1639.4
250	1690	1506	1657	1504	1588	1589
260	1594	1726	1476	1468	1741	1601
270	1857	1762	1613	1565	1745	1708.4
280	2433	1661	1704	1626	1606	1806
290	2209	1789	1919	1588	1702	1841.4
300	2129	1622	1945	1707	1613	1803.2
310	2588	1766	1640	2125	1837	1991.2
320	2248	2000	1859	1978	1734	1963.8
330	2493	1990	1915	1919	1715	2006.4
340	2426	1982	1898	1796	1895	1999.4
350	2553	1863	1712	1864	1868	1972
360	2298	1876	2135	2049	1862	2044
370	3123	2133	2129	1847	1993	2245
380	2372	2176	1848	1842	2010	2049.6
390	2669	2081	1925	1799	2172	2129.2
400	2621	2050	2083	1940	2023	2143.4
410	2712	1971	1898	1831	1950	2072.4
420	2471	1999	2319	1921	2176	2177.2

430	2713	2059	1940	2721	2062	2299
440	2747	2334	2250	1990	1927	2249.6
450	2800	1984	2127	2138	2563	2322.4
460	2825	2291	2436	2046	2122	2344
470	2811	1833	2335	2132	2696	2361.4
480	2899	2057	2076	2119	2294	2289
490	2685	2070	3000	2202	2474	2486.2
500	3079	2021	3044	2241	2558	2588.6
510	3205	2255	2755	2420	3297	2786.4
520	2554	2409	3128	2878	2803	2754.4
530	2826	2131	2687	3177	3083	2780.8
540	3442	2820	2913	2938	2782	2979
550	3000	2613	3271	3144	2111	2827.8
560	3031	2763	2762	3375	2424	2871
570	2754	2531	3538	3460	2343	2925.2
580	3445	3148	3123	3098	2458	3054.4
590	3410	3317	3823	3031	2988	3313.8
600	2934	3098	3024	3200	3202	3091.6
610	3815	3193	3574	2890	3379	3370.2
620	3080	2989	3000	3193	3804	3213.2
630	3230	3392	2966	3032	2791	3082.2
640	4140	3412	3152	3023	3098	3365
650	3665	4045	3018	3705	3313	3549.2
660	3204	3449	3039	3309	3092	3218.6
670	3132	3331	3361	3293	3517	3326.8
680	3023	4286	3741	3336	3813	3639.8
690	3652	3241	3294	3833	3187	3441.4
700	3299	3014	3543	3525	3601	3396.4
710	3235	3386	3686	3227	3693	3445.4
720	3411	3042	3397	3483	3364	3339.4
730	4409	3186	4035	3064	3350	3608.8
740	3387	3342	3485	4070	3410	3538.8
750	3614	4065	3473	3399	3799	3670
760	3764	3669	3475	4272	3617	3759.4
770	4288	3793	3591	4020	4213	3981
780	3998	3812	3901	3789	3895	3879
790	4688	4225	3878	3808	4783	4276.4
800	3727	4136	4328	3900	4061	4030.4
810	4186	4015	3824	3916	4214	4031

820	4242	3759	4086	3837	3756	3936
830	3530	4060	3718	3934	4930	4034.4
840	4238	3805	4064	4565	3864	4107.2
850	3719	4677	4776	4363	4388	4384.6
860	4098	4255	4033	3659	4511	4111.2
870	3994	3901	3720	4023	4145	3956.6
880	4251	4594	4035	4334	3813	4205.4
890	4265	4110	3649	4159	4452	4127
900	4078	4262	3813	4432	3756	4068.2
910	4134	3730	4169	4350	4437	4164
920	3960	4139	4212	3958	4026	4059
930	4233	4148	4201	3846	3879	4061.4
940	4336	4048	5105	4386	4182	4411.4
950	4681	4036	3893	3711	4291	4122.4
960	3729	4351	4454	4195	4324	4210.6
970	4824	4200	4392	4035	4187	4327.6
980	5199	3944	4130	4150	5043	4493.2
990	4574	4553	4538	4389	4325	4475.8
1000	4796	4291	3979	4398	4410	4374.8

Full ontology – 1 constraint on the taste attribute and n-8 results

Number of nodes	Mewasurement 1	Measurement 2	Measurement 3	Measurement 4	Measurement 5	Average
10	593	570	673	576	543	591
20	678	680	863	683	665	713.8
30	837	768	939	765	778	817.4
40	790	808	861	797	816	814.4
50	818	869	895	885	837	860.8
60	960	993	974	1024	994	989
70	1000	961	1027	958	1047	998.6
80	986	1058	1253	1081	1097	1095
90	1034	1104	1344	1240	1138	1172
100	1193	1125	1308	1222	1205	1210.6
110	1215	1267	1302	1348	1320	1290.4
120	1289	1382	1331	1362	1302	1333.2
130	1349	1382	1393	1311	1385	1364
140	1496	1362	1391	1403	1442	1418.8
150	1473	1403	1510	1568	1552	1501.2

160	1417	1553	1587	1519	1492	1513.6
170	1514	1435	1364	1520	1687	1504
180	1562	1589	1581	1453	1606	1558.2
190	1552	1485	1682	1549	1743	1602.2
200	1804	1716	1553	1764	1475	1662.4
210	1634	1317	1600	1521	1599	1534.2
220	1464	1734	1466	1616	1440	1544
230	1672	1562	1459	1692	1398	1556.6
240	1438	1698	1480	1467	1478	1512.2
250	1465	1720	1657	1644	1417	1580.6
260	1492	1627	1795	1519	1575	1601.6
270	1766	1696	1499	1791	1832	1716.8
280	1796	2074	1620	2016	1501	1801.4
290	1664	1969	1915	1849	1588	1797
300	1656	2025	1810	1859	1898	1849.6
310	1694	2185	1606	1683	1963	1826.2
320	1686	1671	1820	2018	1797	1798.4
330	1842	2363	1927	2022	1814	1993.6
340	1949	1841	1758	2017	2025	1918
350	1750	2048	2144	1782	1859	1916.6
360	1826	1964	1989	2200	2180	2031.8
370	1811	2179	1898	2074	1862	1964.8
380	2366	1982	2087	1830	1862	2025.4
390	2022	2132	2547	1887	2361	2189.8
400	2415	2096	2112	1882	2206	2142.2
410	2227	1985	2459	2091	2267	2205.8
420	1937	2289	2482	2399	2386	2298.6
430	2332	2461	2658	2308	2356	2423
440	2374	2999	2520	2187	2225	2461
450	2262	2805	2647	1965	2186	2373
460	2384	2712	2773	2566	2437	2574.4
470	2149	2231	2665	2236	2491	2354.4
480	2253	2644	2420	2638	2926	2576.2
490	2281	2511	2184	2586	3056	2523.6
500	3201	2736	2500	3219	2526	2836.4
510	3246	3020	2239	3227	3059	2958.2
520	3105	3266	2536	2766	3190	2972.6
530	2906	2870	2925	3744	3437	3176.4
540	2995	3416	3295	3094	3293	3218.6
550	3528	3214	3278	2968	3547	3307
560	3234	3301	3272	3047	3046	3180

570	2997	3226	3280	3413	3279	3239
580	3142	3008	3300	3376	2993	3163.8
590	3454	3691	3154	3349	3286	3386.8
600	3447	3108	3461	2975	3783	3354.8
610	3095	3370	3090	3184	3013	3150.4
620	3131	3408	3718	4021	2986	3452.8
630	3088	3161	3496	3093	3466	3260.8
640	3171	3269	3273	3473	3749	3387
650	3228	3365	3363	3646	3248	3370
660	3588	3513	3613	3745	3114	3514.6
670	3295	3488	3031	3513	3712	3407.8
680	3282	3735	3471	3478	3322	3457.6
690	3304	3138	3651	3344	3733	3434
700	3665	3596	3667	3139	3235	3460.4
710	3552	3906	3791	3441	2881	3514.2
720	3708	3754	3780	3776	3835	3770.6
730	3756	3501	4053	3710	3630	3730
740	3596	3806	3757	3881	3847	3777.4
750	3643	4139	3676	4147	3754	3871.8
760	3983	3637	3880	3767	3529	3759.2
770	3855	3939	4007	4139	4437	4075.4
780	4337	3482	3897	4711	4010	4087.4
790	3572	4193	3936	3552	4168	3884.2
800	4350	3991	3671	3865	4013	3978
810	3932	4105	3923	3585	4100	3929
820	4079	4387	4316	3849	3936	4113.4
830	3499	4176	4119	4566	4084	4088.8
840	4393	4110	4148	4000	3915	4113.2
850	3920	3731	3629	4126	4201	3921.4
860	3966	3903	3836	4259	3977	3988.2
870	3735	4140	3879	4092	3958	3960.8
880	3871	3408	4054	3941	4198	3894.4
890	3756	4251	3910	3931	4337	4037
900	4193	4271	3893	4082	4051	4098
910	3885	3847	4362	4267	4456	4163.4
920	3904	4062	4429	4294	4387	4215.2
930	4156	4373	3871	4147	4195	4148.4
940	4017	4122	4249	4255	4460	4220.6
950	4498	4047	4065	4242	3888	4148
960	4303	3944	3949	4008	4726	4186
970	4182	4361	4463	4242	4813	4412.2

980	4129	4406	4498	4840	4432	4461
990	4855	4228	4768	4581	4206	4527.6
1000	4583	4237	4110	4555	4285	4354