# Modelling emergency response processes: Comparative study on OWL and UML

**Wei Xu**
Delft University of Technology,
Jaffalaan 9, 2628 BX Delft, the Netherlands
Wei.Xu@tudelft.nl

**Arta Dilo**
Delft University of Technology,
Jaffalaan 9, 2628 BX Delft, the Netherlands
A.Dilo@tudelft.nl

**Sisi Zlatanova**
Delft University of Technology,
Jaffalaan 9, 2628 BX Delft, the Netherlands
S.Zlatanova@tudelft.nl

**Peter van Oosterom**
Delft University of Technology,
Jaffalaan 9, 2628 BX Delft, the Netherlands
P.J.M.vanOosterom@tudelft.nl

## ABSTRACT

The increasing number of natural and man-made disasters, such as earthquakes, tsunamis, traffic accidents, has demonstrated the importance of disaster management. Disaster management is complex; it involves many actors and large amount of information. It is important to understand the interrelations between the DM components – tasks, processes – and the responsible parties – actors, sectors – as well as the flow of information. The present research is a part of an ongoing work on building ontology for disaster management in the Netherlands. The ontology captures the structure and relations by modelling the process, the tasks, the actors and the information. It represents relations between processes and tasks, relations between sectors and actors, tasks and information. By simple inference, it becomes clear which actor needs what information. Modelling of disaster management processes can be done is several ways. In this paper the work is carried out both in OWL-DL and in UML. The OWL-DL and UML have many similarities in that they both provide the same elements, e.g., classes, attributes, relations (UML use the term associations and OWL uses the term properties). However the two languages are designed for different purposes and thus provide different facilities for modelling. This paper presents a comparative study on the applicability of the two approaches to the modelling of disaster management processes.

### Keywords

Disaster management, ontology, data modeling, OWL-DL, UML

## INTRODUCTION

In the Netherlands, four sectors are mainly responsible for disaster management: the fire brigade, the police, the medical service, and the municipality. Disaster Management (DM) is performed by a fixed set of processes (Borkulo et al 2006, Diehl et al 2006, Snoeren 2006), each having a well-defined objective and a responsible *sector*. A *process* is defined as a series of connected assistance and management *tasks*, performed by several *actors*, i.e. people from the sectors or organisations involved in DM, fulfilling certain role(s) and responsibilities. Knowing which *information* should be available in a given tasks and who is performing the task, the needed data can be specified and appropriately delivered to the actor (Zlatanova et al 2007, Zlatanova 2007).

Having this background in mind, a conceptualisation is required to make the knowledge of the relations among the DM processes, actors, tasks, sectors and information explicit. Unified Modelling Language (UML) and OWL-DL (Web Ontology Language - Descriptive Logic) are two of the most used modelling languages. Both of the languages are used to represent a domain (information) by showing its classes and relations among them. But the question is which of the two is more appropriate for this task.

UML is standardised modelling language for object modelling widely used by domain experts for expressing their domain knowledge. On the other hand, the conceptualisation, which is represented in OWL, is usually considered as an ontology. The ontology is also used to express and share the domain knowledge. UML is developed for the purpose of general domain modelling, application system design, database design, business modelling and so on; OWL is developed for representing semantic information for the World Wide Web. There are differences between the two modelling languages and it is not immediately clear which one is more favourable for the purpose of disaster management.

In this paper, we present our comparative study on the differences between the two languages by a case study for the Dutch emergency management procedures. Processes, actors, tasks and information in the disaster

management. The paper starts by introducing ontology and two modelling languages - UML and OWL - and summarises the major difference and abilities of the two languages in section 2. Section 3 introduces classes and relations among them by explaining what these concepts are in the disaster management. The comparison of OWL and UML language elements are done based on the model introduced in section 3. Section 4 summarises the work.

## OWL AND UML

Ontology and object-modelling have developed independently from each other but both with the idea to allow conceptualisation of real world phenomena,

Ontology has been widely used within computer science, e.g. Artificial Intelligence, Computational Linguistics and Database Theory. Guarino has made a summary of the current research on ontology within computer science (Guarino 1998). The most accepted definition of ontology is 'an explicit specification of a conceptualisation' (Guarion and Giaretta 1995). A *conceptualisation* is 'an intentional semantic structure which encodes the implicit rules constraining the structure of a piece of reality' (Gunarion 1995 and Genesereth and Nilsson 1999). We can consider a 'conceptulisation' as a model, which simply represents our thoughts over a certain problem. This model could be words describing things, drawings, network diagrams, logical expressions and so on. However, the meaning of ontology is ambiguous and many interpretations exist (Guarino and Giaretta 1995). Guarino has distinguished the meaning of ontology and suggests more terms should be used for disambiguating the meaning of ontology (Guarino and Giaretta 1995). No matter how the meaning of ontology is interpreted, the goal of ontology is to 'reduce the conceptual and terminological confusion among the members of a virtual community who need to share electronic documents and information of various kinds' (Navigli and Velardi 2004) and it is a way of 'conceptualisation' (Genesereth and Nilsson 1999).

In this paper, the word ontology is referred to as a conceptualisation which is represented by ontology representation language. OWL, which is 'designed for use by applications that need to process the content of information instead of just presenting information to humans' (OWL 2004), is to provide a standard language for the representation of ontologies on the World Wide Web. It is one of the most used ontology representation languages (Badder et al. 2003). OWL has three increasingly-expressive (what can we express using the language) but decreasingly-decidability (the time spent on the inference of the represented knowledge is finite) sublanguages: OWL Lite, OWL DL and OWL Full. OWL-DL, of which the semantics is based on Description Logic (DL) (Badder et al. 2002), supports the maximum expressiveness without losing the computational completeness and decidability of reasoning. The following of the paper will simply use OWL to refer to OWL DL. More of the semantics of OWL can be found OWL Web Ontology Language Semantics and Abstract Syntax (Patel-Schneider et al. 2004)

Object-oriented modelling has slightly different roots. It has been developed with the idea to help engineering projects by building a model about the system. A model allows for hiding or masking details, revealing 'the big picture', or alternatively making possible to focus on different aspects of the prototype. In the last two decades many object-oriented modelling languages have been proposed of which UML has been the most accepted and developed and currently approved as a standard in many domains.

UML is a general object-oriented language for representing application structures, behaviours, architecture, business process and data structures (www.uml.org). It is a visual design notation, which is designed to integrate competing proposals for modelling languages in the area of software engineering (Falkovych 2003). The language is power enough to show a detailed view of an application, visualise links to other applications or even to other domains. Last developments of UML make it possible to do conceptualisation of business process or a business rules view. There are nine types of UML diagrams (http://dn.codegear.com/article/31863) of which one is *class diagram* (Bell 2004). A class diagram gives an overview of a system by showing its classes and the relations among them and as such it is static. UML class diagram is widely used by domain experts (or verified by them) for expressing their knowledge of a certain domain, it is also a way of *conceptualisation* (Baclawski et al. 2002, Brockmans et al. 2006 and Evans 1998). In the rest of the paper, when we talk about UML, it is always referred to as UML class diagram.

Clearly, OWL and UML are developed for different purposes. UML has a visual design notation, which is much more human-readable and OWL is derived from Description Logics, which is meant for inference (Hart et al. 2004). Apart form that, OWL and UML differ in expressing concepts. Before we further explain, we will recall the definitions necessary and sufficient conditions (a modified definition which more suits our purpose,the original can be found at http://www.sfu.ca/philosophy/swartz/conditions1.htm#section2):

- Necessary conditions: A necessary condition N states that an individual m is a member of a class C only if it satisfies the condition N. Thus, if m does not satisfy condition N, then m is not a member of class C (But, m satisfies condition N does not tell anything about m being a member of C or not).

- Sufficient conditions: A sufficient condition S states that an individual m is a member of class C if it satisfies condition S. Thus, if m satisfies condition S, then m is a member of class C.

In OWL, class is defined in terms of necessary conditions or necessary and sufficient conditions. A class is called partial or primitive if it is defined only by necessary conditions. A class is called a complete or defined if it is defined by both necessary and sufficient conditions. With that OWL allows us to represent knowledge like 'The Centralist, is a type of Actor, who registers the incident'. In our conceptualisation, this is a partial definition of the centralist, because we know there might be others (e.g. officer of duty) who can also register the incident. By given the fact that one registers the incident, we still do not know if he/she is a centralist or not. On the other hand, the knowledge 'Information that is involved in disaster management consists of the existing information and operational information', which is a complete definition of information, because information only consists of the existing information and the operational information (no more and no less).

UML does not distinguish between the defined class and the partial class. The way that a class is described in UML is by listing the attributes and operators that are in a class. UML is based on *close world assumption.* The close word assumption states that what is not currently known to be true is false. The counterpart is *open world assumption* (this is what OWL is based on*)*, which asserts that a system's knowledge is incomplete, so that if a statement cannot be inferred from what is represented in the system, it cannot be inferred to be false.

Moreover, since OWL is rooted in Description Logics, it allows the set operators intersection, union and complement to be used for describing a class. OWL allows us to define hierarchy of a property (property is the term used in OWL and it is referred to as the relations between two classes) (for instance, the hierarchy of isAncestor and isParent). OWL also allows users to specify the characteristics of a property as transitive, symmetric, functional, inverse and inverseFuntional in order to support the inference (the inference ability of OWL can be found at Bechhofer 2003). Inferences are usually done through a reasoner and the most commonly OWL reasoners are RacerPro (http://www.racer-systems.com/), Fact++ (http://owl.man.ac.uk/factplusplus/) and Pallet (http://www.mindswap.org/2003/pellet/).

In a word, the open world assumption and inference ability makes OWL different from UML. Besides that the language elements (constructors) of both languages are different. Before we illustrate the language elements differences, we will introduce the case, which will be used for demonstration.

## THE CONCEPTUALISATION OF DISASTER MANAGEMENT PROCESSES

The conceptualisation is done for the purpose of developing a system that will provide the most appropriate data to each actor, who is involved in disaster management. The 'most appropriate data', here is the data that are primarily needed to successfully perform the task an actor is responsible for.

The disaster management in the Netherlands is managed by a series of connected activities, which are indicated as *Process* (Snoeren 2006). There are 25 types of processes defined in the Netherlands (Dieh et al. 2006 and Borkulo et al. 2006), for instance process 'observation and measurement', process 'traffic control', process 'decontaminating affected people and animals' and so on. Observations and measurements is a process that is initiated when release of dangerous substances (in the air, on the ground, in the water) is detected and a number of specialised measurements should be performed or samples have to be collected.

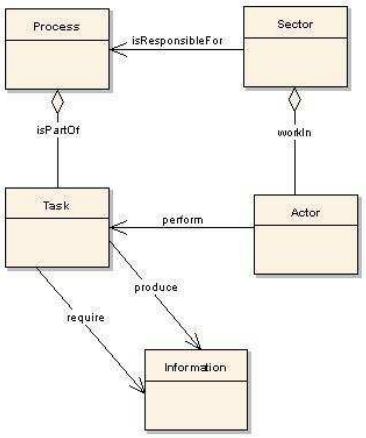
Each emergency response organisation is responsible for a group of processes. There are four primarily emergency response organisations in the Netherlands: the fire brigade, the police, the medical service and the municipality. The four responding organisations belong to the emergency response *Sector*. For instance, the fire brigade is responsible for the processes: 'observation and measurement', 'rescuing and technical assistance', 'decontaminating people and animals' and so on; the medical service is responsible for the process 'medical aid chain', 'psycho-social aid and care' and so on; the police is responsible for 'clearance and evacuation', 'traffic control' and so on; the municipality is responsible for 'advice and information', 'relief and care', 'damage registration of victims' and so on. In other words, the 25 processes are organised into four clusters and each of the clusters has its responsible organisation (sector).

*Actors* are generally the end users of a system for disaster management, who can play different roles, i.e. from emergency operator (centralist) to policy maker. During disaster management, various actors will get involved, e.g., the fire brigade teams, the medical worker for the medical service, the traffic controller from the police, the municipal worker for the municipality. In many incidents, the actors remain restricted to those from the four primarily responding sectors.

Within each process, an actor may need to perform different operations on the system in order to fulfil his responsibilities. The operations that the actor performs on the system are represented as *Task* in our model. Every process consists of a set of tasks, e.g., the process 'observation and measurement' consists of task 'register

the incident', 'identify the incident area', 'prepare the measurement plan', 'obtain the measurement task', 'send the measurement' and 'compute the gasmal'.

Each actor needs data to perform his/her task, but he/she can also deliver data to the system. The data that are either required or produced by the task is indicated as *Information*. Two types of Information are identified as existing information and operational information. Usually a task requires some existing (and/or operational) information and produces some operational information. For instance, the task 'register the incident' creates the information of the incident, which is operational information; the task 'prepare the measurement plan' requires the information about existing features like buildings, roads, surrounding area, but also operational data like location (address) of incident and location at which the measurements hate to be performed.




a). Abstract classes Hierarchy

| Relation Name | Source | Target | Description |
|---|---|---|---|
| isResponsible for | Sector | Process | Express the relationship between the Sector and the Process, specify which sector is responsible for which process |
| workIn | Actor | Sector | Express the relationship between the Actor and the Sector, specify which actor works in which sector |
| isPartOf | Task | Process | Express the relationship between the Task and the Process, specify which task is a part of which Process |
| perform | Actor | Task | Express the relationship between Actor and Task, specify which actor performs which task |
| require | Task | Information | Express the relationship between the task and the Information, specify which task requires what information |
| produce | Task | Information | Express the relationship between the task and the Information, specify which task produces what information |
| isA | | | Generalisation relation between the super class and the sub class |

b). Relations among abstract classes

**Figure 1. Abstract classes Hierarchy and relations in the model**

*Process*, *Sector*, *Actor*, *Task* and *Information* are the most important concepts in our model and therefore they form the top-level classes. Figure 1 shows the overview of the classes and the most important relations between that are included in the model. While Figure 1a illustrates the classes and the relationships using the UML class diagram, Figure 1b explains the meaning of the notated relationships. Furthermore each of the five top-level classes has its subclass, which is modelled as 'isA' relation. For instance, each of the 25 processes is a Process; each of the four first responding organisations is a Sector; each of the emergency workers is an Actor and so on. In the following UML diagram, the 'isA' relation will be represented by generalisation and will not be explicitly stated. The relations apply to the super class and so they do with subclasses. We further elaborate model by showing the subclasses. Since the subclasses are quite many (e.g. Process consist of 25 subclasses), we will use the process 'observation and measurement' as an example. The UML class diagram in Figure 2 shows the model.

'Observation and measurement' is one of the seven processes for which the fire brigade is responsible. The class Fire Brigade together with the other three classes, the class Police, the class Medical Service and the class Municipality, which represent the other three responding organisations, are subclasses of Sector. The relation 'isResponsibleFor' between the class Fire Brigade and the class Observation and Measurement, represents the fire brigade is the responding sector for this process.

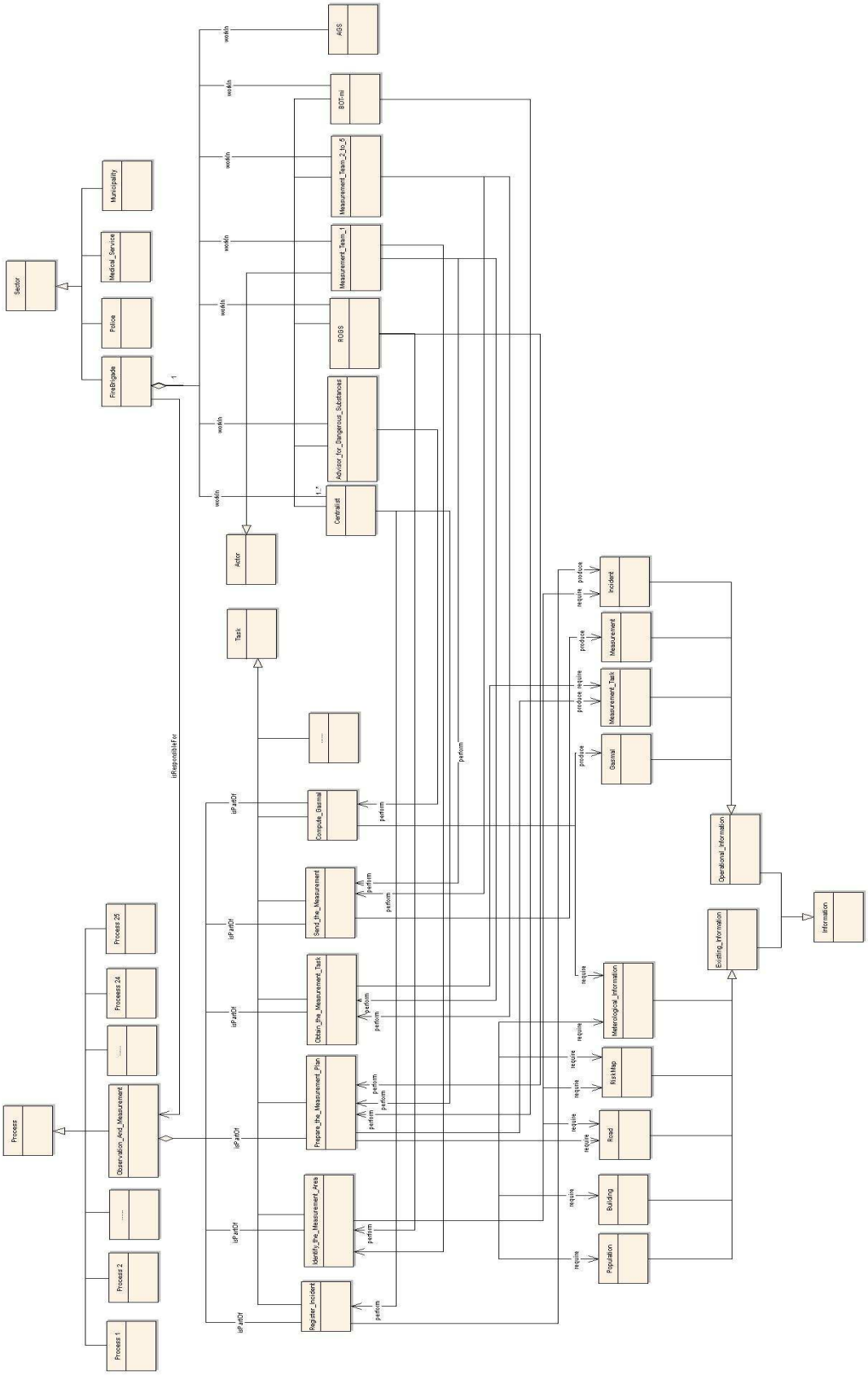



**Figure 2. UML class diagram for the process 'observation and measurement'**

Several emergency officers, such as the centralist, the first measurement team, the second to the fifth measurement team, the advisor for dangerous substances, the ROGS (regional officer for dangerous substances) and the BOT-im (policy support team for environmental incidents), are responsible for the tasks in this process. These emergency officers are made subclasses of the class Actor and relation 'workIn' relates them with the class Fire Brigade to represent that they all from the fire brigade.

This process consists of six tasks: 'register the incident', 'identify the measurement area', 'prepare the measurement plan', 'obtain the measure plan', 'send the measurement' and 'compute the gasmal', which are made subclasses of the class Task. The relation 'isPartOf' connects each of the tasks to process 'observation and measurement' to represents what tasks this process consists of.

Each task is carried out by one or more actors, thus the relation 'perform' connects each task with the actor, who performs the task. The Centralist performs the task Register_the_Incident. The Centralist also performs task Identify_the_Measurement_Area and the Prepare_the_Measurement_Plan. The other actors are connected to the tasks which they perform.

Information is identified as Existing Information and Operational Information, which are represented as two subclasses of the Information. More specific information are further identified and represented as either the subclasses of the Existing_Information or the Operational_Information. The information about the Population, the Building, the Road, the RiskMap and the Meterological Information are subclasses of the Existing_Information. The information about the Incident, the MeasurementTask, the Measurement and the Gasmal are represented as the subclasses of the Operational_Information.

## COMPARISON BY THE CASE STUDY

This section compares the modelling approaches of UML and OWL. In order to make the comparison of case study meaningful, we need to restrict the contents from the two languages we are going to consider. As mentioned previously we focus on UML: classes, classes' attributes, relationships between classes, and multiplicities, which will be compared with classes and properties used in OWL. The conceptualisation for the process 'observation and measurement' as given in figure 2 will be used to discuss the differences in the two modelling approaches.

### Classes

Classes are used to represent the concept in a domain, which is a set of instances that belong to that class. Both UML and OWL use classes to represent concepts of a domain. UML describes the class by defining the name of the class, listing the class' attributes (and operators) and defining associations with other classes. OWL describes the class by defining the name of the class and giving the necessary conditions (NC) or the necessary and sufficient conditions (NSC) of the class in terms of properties (the word property is used in OWL to refer to the relation between two classes).

The figure 3 and figure 4 defines two classes Sector and Actor (we omit the classes' attributes for simplicity reasons when we do not consider the class' attributes) in UML class diagram and OWL. OWL declares classes by owl:Class language element and rdf:ID is used to give the class a name (rdf:ID are RDF tag, which is used to give the class a unique identifier (Beckett 2004)). After the class has been defined, the class' name can be used to define instances.

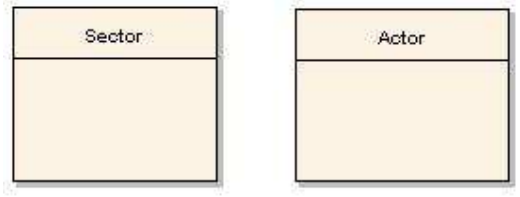



**Figure 3. UML class diagram of Sector and Actor**

```
<owl:Class rdf:ID="Sector"/>
<owl:Class rdf:ID="Actor"/>
```

**Figure 4. OWL representation of Sector and Actor**

Both of UML and OWL allow users to declare class by giving it a name. However, they differ in describing the classes. The next section explains how they differ in describing the classes' attributes.

### Classes' attributes

In OWL, there is not a direct way to define the classes' attributes as UML does. In OWL, classes' attributes are represented through properties. OWL uses the word 'property' to represent the relationship between two classes. In OWL, the class's attribute is regarded as the relationship between the class and the attribute's type. The are two types of property in OWL: owl:datatypeProperty and owl:objectProperty (OWL 2004). If a class is related to the primitive types, which are defined by RDF literals or XML Schema (such as string, integer, long and ect.), owl:datatypeProperty is used (OWL 2004). Otherwise owl:objectProperty is used. Similar constructs exist in


**Figure 5. UML's Attributes**

```
<owl:ObjectProperty rdf:ID="location">
    <rdfs:domain rdf:resource="#Sector"/>
    <rdfs:range rdf:resource="#Location"/>
</owl:ObjectProperty>
```

**ObjectProperty**

```
<owl:DatatypeProperty rdf:ID="sectorCode">
    <rdfs:domain rdf:resource="#Sector"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="sectorID">
    <rdfs:domain rdf:resource="#Sector"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

**DataTypeProperty**

**Figure 6. OWL's Properties**

UML as well. In the next example in Figure 5 and Figure 6 Location is data type, which has to be additionally defined, for example as a triple of floats (for X, Y and Z), or as String (Address).

In figure 5 the UML class diagram of Sector defines the sector class and its three attributes: location (where the sector is located) and location is of type Location, which is a user defined class, sectorCode (a unique string code for the sector) and the sectorID (the identification of a sector). Figure 6 represent the same class in OWL. In OWL, owl:ObjectProperty and owl:DatatypeProperty are used to declare properties in OWL; rdf:ID is to give the property a unique name; rdfs:domain and rdfs:range for a property corresponds to source and target for directed relationships in UML class diagram. If a class has been defined using <owl:class rdf:ID='className'>, it can be referenced as rdf:resource='#className'. The owl code <rdfs:domain rdf:resource='#Sector'> means the domain is Sector class (because Sector has been defined before rdf:resource='#Sector' is used to reference the class). Three owl properties are created, location (owl:objectProperty, Location is not of primitive types, thus use owl:objectProperty), sectorID (owl:datatypeProperty) and sectorCode (owl:datatypeProperty) (string and long are primitive types, thus use owl:datatypeProperty). The three properties relates Sector to Location, String (&xsd;string, xmlshema of String) and Long (&xsd;long, xmlshema of Long). This is the OWL's way of expressing attributes for a class. Apart from representing a class's attributes, owl properties are also used to represent the relationships between classes.

### Relationships

Relationships in UML class diagram are generalisations, aggregations, associations, compositions, dependencies and interfaces. OWL does not provide dependencies, interface aggregation and compositions. Thus we only consider generalisation and associations.

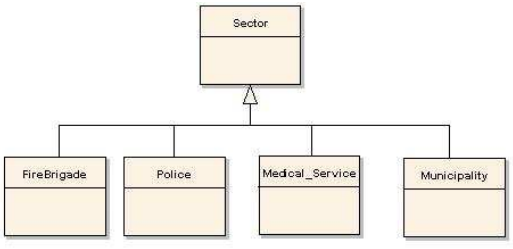



**Figure 7. UML class diagram of sector and its subclasses**

```
<owl:Class rdf:ID="FireBrigade">
    <rdfs:subClassOf rdf:resource="#Sector"/>
</owl:Class>
<owl:Class rdf:ID="MedicalService">
    <rdfs:subClassOf rdf:resource="#Sector"/>
</owl:Class>
<owl:Class rdf:ID="Municipalicity">
    <rdfs:subClassOf rdf:resource="#Sector"/>
<owl:Class rdf:ID="Police">
    <rdfs:subClassOf rdf:resource="#Sector"/>
</owl:Class>
```

**Figure 8. OWL representation of generalisation**

Generalisation is 'is_a' relationship between two classes (the super class and its subclass), which can easily be represented in both UML and OWL. The UML in figure 7 represents four subclasses of class Sector: the MedicalService, the Municipality, the Fire Brigade, and the Police. The OWL code in figure 8 uses the property rdfs:subClassOf to declare that a class is a subclass of another. Note that the declaration of the relationship with the super class is done while declaring the class FireBrigade.

Associations are relations between classes in UML, which describe the relationship and the rules (the multiplicity and the direction of the relation) that regulate the relationship. In OWL, owl:objectProperty is used to represent the relation between classes, with the domain indicating the source and the range indicating the target. The UML in Figure 9 defines the association 'perform' to represent the Centralist performs the task of Register_the_Incident. In Figure 10, OWL creates an owl:objectProperty perform and restricts its domain and range to be the Centralist and the Initiate_the_Incident.


**Figure 9. association between two classes**

```
<owl:ObjectProperty rdf:ID="perform">
  <rdfs:domain rdf:resource="#Centralist"/>
  <rdfs:range rdf:resource="#Initiate_the_Incident"/>
</owl:ObjectProperty>
```

**Figure 10. OWL object property**

OWL is a bit reached in describing the characteristics of a property than UML in relationsips. A user can specify: transitive, symmetric, functional, inverseOf and inverseFunctional Property. Apart from that, OWL allows the users to specify the sub-property of a property. Those two characteristics are not present in UML class diagram. The ontology of disaster management processes so far does not include any involvement of using these properties restrictions, thus we only give abstract inference usage that these properties restrictions can bring. In the Table 1, P, P1, P2 represent OWL object properties, x, y, z represent instances that are derived from class and that are related by properties, '=' means equivalent. The declaration of property's characteristics is done with OWL and the inference is accomplished through OWL reasoners.

| PropertyName | Condition | Inference |
|---|---|---|
| TransitiveProperty | P(x,y) and P(y,z) | P(x,z) |
| SymmetricProperty | P(x,y) | P(y,x) |
| FunctionalProperty | P(x,y) and P(x,z) | y = z |
| InverseOf | P1(x,y), inverseOf(P1,P2) | P2(y,x) |
| InverseFunctionalProperty | P(y,x) and P(z,x) | y = z |
| | | |

**Table 1. OWL inference properties**

Furthermore, in OWL, the property can be defined without defining either its domain or its range. The property defined in this way can be applied to any classes (domain) and the range can be set to any classes. In other words, OWL allows users to define the owl:property globally and use it locally.

OWL uses owl:Restriction to put constraints on the property within a class. The owl:Restriction is included inside a class, which mean this restriction is only valid for this class. UML can consider constraints on the classes only by attaching a text describing the constraint to it.

Owl:onProperty specifies to which property the restriction is on and someValuesFrom specifies the range of the property. The Figure 11 shows how OWL representation defines a owl:objectProperty perform (whitout specifying its domain and range). Inside MeasurementTeam_01, the range of perform is defined to be Obtain_Measurement_Task. Inside the Centralist, the range of the property perform is defined to be Register_Incident.

```
<owl:ObjectProperty rdf:ID="perform"/>                          define property

<owl:Class rdf:ID="MeasurementTeam_01">

    <owl:Restriction>
      <owl:onProperty rdf:resource="#perform"/>
      <owl:someValuesFrom rdf:resource="#Obtain_Measurement_Task"/>
    </owl:Restriction>

  </owl:Class>                                  restrict it to MeasurementTeam_01

<owl:Class rdf:ID="Centralist">
  <owl:Restriction>
    <owl:onProperty rdf:resource="#perform"/>
    <owl:someValuesFrom rdf:resource="#Initiate_the_Incident"/>
  </owl:Restriction>

 </owl:Class>                                              restrict it to centralist
```

**Figure 11. Restriction on properties within a class**

Apart from owl:someValuesFrom, OWL also uses owl:allValueFrom to restrict a property within a class. When owl:allValueFrom is used with a property, it restricts the range and no other values are allowed. While owl:someValueFrom is used with a property, it restricts the range but other values are allowed. Since OWL has a deep grounding on Description Logics and Description Logics has formal descriptions of allValuesFrom and someValuesFrom, interested readers can refer to Baader et al. 2003.

Aggregation and composition, which represent the part-whole relations in UML, are not supported by OWL language. Thus owl:objectproperty with key words, such as 'is_part_of', 'consist_of' should be used instead. However, OWL provides more possibilites to represent relations between two classes, such as owl:equivalent, owl:disjoint, owl:unionOf, owl:intersectionOf and owl:complementOf, which can be used to define relations between classes more precise than in UML.

### Multiplicity

UML allows the user to specify the multiplicity for the association's source and target. OWL achieves this by specifying the cardinality of an owl:property in a class. OWL allows users to define the owl:cardinality: exactly 1 (exists exactly one occurrence), min (the minimal number of occurrence) and max (the maximal number occurrence). However, the multiplicity of a relation's two ends in OWL should be specified through two properties (one property and its inverse property). The Figure 12 shows UML class diagram of multiplicity of workIn and the multiplicity represents many to one relation between the Centralist and the FireBrigade. The OWL in figure 13 uses <owl:cardinality rdf:datatype="&xsd;int">number</owl:cadinality> or <owl:mincardinality rdf:datatype="&xsd;int">number</owl:cadinality> to specify the cardinality (number is the numbers of occurrence, owl:mincardinality declares the minimal occurance and owl:cardinality declare the exact occurance) and uses <owl:onProperty rdf:resource="propertyName"> to restrict to which property the cardinality is applied.

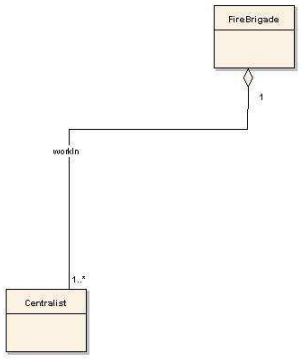



**Figure 12. Multiplicity in UML**

```
<owl:Class rdf:ID="Centralist">
      <owl:Restriction>
        <owl:onProperty rdf:resource="#workIn"/>
        <owl:cardinality rdf:datatype="&xsd;int">1
          </owl:cardinality>
      </owl:Restriction>
  </owl:Class>
  <owl:Class rdf:ID="FireBrigade">
      <owl:Restriction>
        <owl:onProperty rdf:resource="#inverse_of_workIn"/>
        <owl:minCardinality rdf:datatype="&xsd;int">1
          </owl:minCardinality>
      </owl:Restriction>
  </owl:Class>
```

**Figure 13. Cardinality in OWL**

### Instance

Normally instances will not appear in UML class diagram. In OWL, the term 'individual' is used to refer to instances. The instance is created also using the language tag <owl:Thing rdf:ID="instanceID"> and it is declared as an instance of a class by specifying <rdf:type rdf:resource = "className">, where instanceID is the unique identifier (such as the name) of the instance and className is the class to which the instance belongs. For instance, the class Centralist is used to represent all the actors who play the role as centralists, as there might be several persons working as centralists. In Figure 14 OWL uses <owl:Thing rdf:ID = "instanceID"> to declare an instance and use <rdf:type rdf:resource="ClassName"> (instanceID is the unique identified given to the instance by the user and ClassName is the class from which the instance is derived from). In this way, the class description as well as the instance can be present in the same file.

```
<owl:Thing rdf:ID="John">
  <rdf:type rdf:resource="#Centralist"/>
</owl:Thing>
<owl:Thing rdf:ID="Peter">
  <rdf:type rdf:resource="#Centralist"/>
</owl:Thing>
<owl:Thing rdf:ID="Tom">
  <rdf:type rdf:resource="#Centralist"/>
</owl:Thing>
```

**Figure 14. Example of OWL instances for a Centralist**

## DISCUSSION

In this paper, we introduced our conceptualisation of DM processes and our model consisting of Process, Actor, Task, Information and Sector. The concepts were modelled using the two modelling languages UML and OWL as only the UML was taken into account.

As immediately obvious, UML is better readable by humans since it gives a visual overview of classes and relationships. In this respect, UML is more appropriate in the process of conceptualisation, when deciding which classes and relationships should be considered. Our modelling process has showed the same pattern. There some attempts to overcome this disadvantage of OWL by providing plug-ins for visualising the classes and their relationships (Brockmans et al. 2006).

Classes and attributes can be defined in both languages approximately the same way. However, OWL allows for stricter description of attributes. Since the attribute (i.e. property) is considered as a kind of relationships between class and attribute's type, it can be defined independently from the class and repetitively used. This ensures that the meaning of the attribute is exactly the same. In contracts, UML allows attributes with identical names, which belong to different classes to differ. This property of OWL can be used in DM to represent specific terms used in the different emergency response sectors. OWL is lacking the notation of operations, which are part of the UML class description. Operations should be modelled as a special type of properties. Since we have not considered operations in our model, our experience in the way they can be represented in OWL is limited.

The large difference between the two languages is in the description of the relationships (in UML) or properties in OWL. Some relationships are clearer to be expressed by UML (aggregation, composition, dependency, interface), while others with OWL (transitive, symmetric, functional, inverseOf and inverseFunctional). In OWL it is possible to define sub-properties and restrict them. UML would require for this switch from class diagram for example to state chart diagram. Our conceptualisation did not require the definition of sub-property, either. OWL and UML differ with respect to the notation of equivalent or disjoint. UML lacks the semantics of 'equivalent' (being based on close word assumption), even though an 'equivalent' or a 'disjoint' association can be created explicitly in UML and two classes can be related with these two relations.

The general conclusion is that the two languages are compatible in the modelling part of static events (as what UML class diagram is designed for) and when simple cases are considered. The advantages of OWL would become more apparent when logic and reasoning should be applied. For example, the search for the most appropriate data would be more flexible and efficient using OWL. First implementations (not discussed in this paper) have shown that concepts described in OWL facilitate selective search and extraction of items (e.g. 'buildings') from particular data set (e.g. large topographic map, small scale map). It should not be forgotten also that UML has eight more diagrams which are not investigated in this study.

As mentioned above, users can also specify inference rules in OWL. For instance, if we want to derive the relation between the actor and the information, we need to specify a rule like 'require (Task, Information) ^ perform (Actor, Task) -> require (Actor, Information)', which means if there is a 'require' relation between a certain task and a certain type of information and there is also a 'perform' relation between a certain actor and this task, then (it can be inferred) there is a 'require' relation between this actor and this type of information. Having the relations between Actor-Task and Task-Information and this rule defined, we can infer the relation between the Actor and the Information. However, as mentioned earlier, this inference will be accomplished through reasoners.

The implementation of the model in UML and OWL also differs. UML class diagram may change when implemented for different applications. For example if the model is going to be used as a data model in Database Management System (i.e. classes should be mapped to relational tables), some classes may be omitted or subclasses (with same attributes) can be organised as attributes of super classes. Generally, a class can be mapped to a table, but variations can exist. The decisions depend very much on the type of attributes and relationships that have to be persistent in the data model. For example, the five tasks of process 'measurement and observation'(as given is figure 2) may be organised as instances of the process by defining appropriate attributes to distinguish between them. The super class Process may not be considered for mapping in a table. On the other hand if the UML class diagram is used to develop a C++ application, it can be implemented as it is.

Since OWL incorporates the instances (usually UML class diagram does not work with instances), it is possible to have instances and classes in the same file. But when the number of instances grows large, it is more appropriate to store these instances into database management system, such as Oracle 11g semantic database (http://www.oracle.com/technology/tech/semantic_technologies/index.html). When OWL is implemented in Java, it is not mapped directly because the relations can not be directly applied to classes (Kalyanpur et al. 2004).

OWL is for modelling the classes and the relations but when applications are built, it should be used with reasoners. With the help of reasoners, the consistency of a model represented in OWL can be checked, e.g. if the relation is used with proper classes (checking its domain and ranges), if the definition of a class is correct (checking the property's owl:someValueFrom and owl:allValueFrom). The inference can be also accomplished based on the rules defined in OWL.

The study can also be used as a staring point to define rules for model transformation between the two languages. Currently there are many tools (RacerPro, Pellet, FACT), which are used for checking the consistency and doing the inference on OWL. However, there no tools to automatically convert UML to OWL and vice versa. This would be very useful as for some purposes the same concept might benefit from the properties of one particular language.

Next shortly coming step will be building ontology for several processes and an application that will demonstrate the reasoning power of ontology (OWL-DL).

## REFERENCES

1. Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel- Schneider, P. F., Eds (2003) The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, UK.
2. Baader, F., Horrocks, I., and Sattler, U. (2003) Description logics as ontology languages for the semantic web. In Lecture Notes in Artificial Intelligence, p. 21.
3. Baclawski, K., Kokar, M. M., Kogut, P. A., Hart, L., Smith, J. E., Letkowski, J., and Emery, P. (2002) Extending the unified modeling language for ontology development. Software and System Modeling, pp. 142-156.
4. Bechhofer, S (2003) OWL reasoning examples, available at http://owl.man.ac.uk/2003/why/latest/, last visited in April 2008.
5. Bell, D. (2004) UML basics: The class diagram, available at http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/, last visited in April 2008.
6. Beckett, D. (2004) RDF/XML Syntax Specification, available at http://www.w3.org/TR/rdf-syntax-grammar/, last visited in April 2008.
7. Borkulo, E, V. Barbosa, A. Dilo, S. Zlatanova and H. Scholten, 2006, Services for emergency response systems in the Netherlands, Second Symposium on Gi4DM, 25-26 September, Goa, India, CD ROM, 6 p.
8. Brockmans, S., Colomb, R. M., Haase, P., Kendall, E. F., Wallace, E. K., Welty, C. A., and Xie, G. (2006) A model driven approach for building OWL DL and OWL FULL ontologies. International Semantic Web Conference, pp. 187-200.

9. Diehl, S, J. Neuvel, S. Zlatanova and H.Scholten, 2006, Investigation of user requirements in the emergency response sector: the Dutch case, Second Symposium on Gi4DM, 25-26 September, Goa, India, CD ROM, 6p.
10. Evans, A. S. (1998) Reasoning with UML class diagrams, Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques IEEE Computer Society, p. 102.
11. Falkovych, K. and M. Sabou and H. Stuckenschmidt (2003) UML for the Semantic Web: Transformation-Based Approaches, Knowledge Transformations for the Semantic Web, pp. 92 – 106.
12. Genesereth, M. R., and N. J. Nilsson(1999) Logical Foundation of Artificial Intelligence, Morgan Kaufmann Publishers.
13. Goodchild, M. F. (2007) Recognition of GIS critical, available at http://www.gim-international.com/issues/articles/id892-Recognition_of_GIS_Critical.html, last visited in April 2008.
14. Guarino, N., and P. Giaretta(1995) Ontologies and knowledge bases: Towards a terminological clarification, Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, IOS, pp. 25-32.
15. Guarino, N. (1998) Formal Ontology and Information Systems, Proceedings of FOIS'98, ISO, pp. 3 – 15.
16. Hart, L., P. Emery, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, and Y. Ye (2004) OWL full and UML 2.0 compared, available at http://www.itee.uq.edu.au/~colomb/Papers/UML-OWLont04.03.01.pdf, last visited in April 2008.
17. Kalyanpur, A. and D. Jimenez, S. Battle and J. Padget (2004), Automatic Mapping of OWL Ontologies into Java, in Proceesings of SEKE. p. 6.
18. Kwon, J., and C.J. Moon(2007) Visual modeling and formal specification of constraints of RBAC using semantic web technology. Know.-Based Syst. 20, 4, 350-356.
19. McGuinness, D. L., and F. van Harmelen(2004) OWL web ontology language overview, available at http://www.w3.org/TR/owl-features/, last visited in April 2008.
20. Navigli, R., and P. Velardi (2004) Learning domain ontologies from document warehouses and dedicated web sites. Computational Linguistics, vol. 30, no. 2.
21. OWL (2004) Ontology web language, available at http:///www.w3c.org/2004/owl, last visited in April 2008.
22. Patel-Schneider, P. F., P. Hayes, and I. Horrocks(2004) OWL web ontology language semantics and abstract syntax, available at http://www.w3.org/TR/owl-semantics/, last visited in April 2008.
23. Rector, A. and C. Welty (2005) Simple part-whole relations in OWL Ontologies, available at http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html, last visited in April 2008.
24. Snoeren, G. (2006) Rampbestrijdingsprocessen (in Dutch), available at http://ivm10.ivm.vu.nl/intranets/GeoDI4DM/WorkPackage2/Work%20Documents/Rampbestrijdingsprocessen%20Gineke.pdf, last visited in April 2008
25. Unified modeling language (UML), available at http://www.uml.org/, last visited in April 2008.
26. Xu, W. and S. Zlatanova (2007) Ontologies for Disaster Management, Geomatics Solutions for Disaster Management, Lecture Notes in Geoinformation and Cartography, Springer-Verlag Berlin, Heidelberg, pp. 185-200
27. Zlatanova, S., 2007, Interoperability in Disaster Management, GIMasters and Disasters, GIM International, October 2007, 21 (10), p. 69
28. Zlatanova, S., D. Holweg and M. Stratakis, 2007, Framework for multi-risk emergency response, in: Tao&Li (Eds.) Advances in Mobile Mapping Technology, Taylor&Francis, London, ISPRS Book Series, pp. 159-171