Integrated Semantic and Event-based Reasoning for Emergency Response Applications

Anna Hristoskova[1], Wim Boffé[2], Tom Tourwé[3] and Filip De Turck[1]

[1]Department of Information Technology, IBCN - iMinds, Ghent University, G. Crommenlaan 8/201, 9050 Ghent, Belgium, e.mail: {anna.hristoskova, filip.deturck}@intec.ugent.be

[2]IOS International nv, Wetenschapspark 5, 3590 Diepenbeek, Belgium, e.mail: wim.boffe@iosint.be

[3]Software Engineering & ICT Group, Sirris, A. Reyerslaan 80, 1030 Brussels, Belgium, e.mail: tom.tourwe@sirris.be

Abstract

Emergency response applications require the processing of large amounts of data in order to provide an accurate and concise view of the situation at hand. In this context, the adoption of semantic technologies is essential as it allows focusing on the definition of a formal model and intelligent data processing and reasoning.
This paper presents a novel approach to emergency response applications, such as performed by the fire department, integrating a semantic reasoning engine into an event-based system supporting the use of a formal domain definition. A semantic model is used to automatically generate Java objects which are consumed by the rest of the application. Object manipulations automatically update the underlying model. Additionally inference on the model performed by the reasoning engine is dynamically synchronized with the other architectural components triggering events based on predefined conditions. Validation is executed on a fire fighting

scenario where the state of a fire fighter is constantly monitored based on new device and sensor measurements such as body temperature, heart rate, location. While the reasoning engine keeps track of the fire fighter's context, the event-based engine issues alerts to the rest of the team in case his state deteriorates.

1 Introduction

Emergency responders regularly face large amounts of data, generated by a diverse set of sensors and devices that need to be processed in a timely manner in order to form astute decisions during a disaster. For instance, during a fire fighting scenario, the fire fighters are continuously exposed to various risks and dangerous situations. Regular updates of their state to their team are essential. Vital for such settings are context-aware decision support systems able to provide an accurate and concise view of the situation at hand. Relevant information, captured from various devices and sensors, should be pushed pro-actively and presented in a user-specific and context-aware way [Tsiporkova et al(2012)] supporting the situational awareness of the actors involved. Situational Awareness is a field of research defined as the perception of elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future [Endsley(1995)]. It is about being aware of the current context and the future objectives of the user. Situation-aware applications should be able to optimize the available choices while keeping the user in control.

Current emergency response research focuses on two main categories of applications: crisis simulation environments [Indigo(2012)] and decision support systems [Coates et al(2011), Lijnse et al(2012)]. Usually, such efforts construct emergency management systems built on top of crisis databases. These attempts however provide limited information processing and reasoning. The adoption of semantic technologies enables the formal definition of the required domain concepts and their properties and supports the specification of rules and intelligent reasoning on the available data inferring valuable insights on the current context.

The proposed approach in this paper builds on these principles through the seamless integration of a domain-specific semantic model into a decision support system for emergency response by the fire department in the view of the ASTUTE project [ASTUTE(2011-2014)]. This multidisciplinary project aims at the development of an advanced and innovative pro-active HMI (Human-Machine Interface) interface and reasoning engine for improving user efficiency and safety in various industrial sectors such as automotive, avionics, and emergency services, all operating in data intensive and critical environments. The main contribution of the developed application is the integration of semantic and rule-based reasoning. A proof-of-concept implementation of this integration is developed through a Java Beans Code Generator automatically creating Java classes out of the concepts defined in the semantic model. A Context Engine incorporates a semantic reasoner encapsulating the low level generic reasoning on this model. Results of this reasoning are automatically forwarded to a Decision Engine. This engine includes an

event-based system defining the high level application-specific rules responsible for triggering events and responding to alarms. This approach is validated by means of an illustrative example, including the development of an initial semantic model for emergency dispatching.

2 Related Work

Current emergency response applications focus on various issues such as distributed communication between mobile devices, simulation environments for training purposes, decision support systems processing events from numerous sources (devices, sensors, social media), and formal domain modelling.

Interdroid [Bal et al(2012)] aims at building a platform for distributed applications on Android devices. An example is Raven [Palmer et al(2012)] which supports the use of smart phones for collaborative disaster data collection and sharing. Applications are shared by sharing the database and corresponding schema. The current disaster management application consists of a user interface for editing records in a database used to track lost and found people. Similarly SocEDA [SocEDA(2012), Paraiso et al(2012)] enables the exchange of contextual information between heterogeneous services according to social network information. Its DiCEPE platform supports the interaction of different complex event processing engines simultaneously, while enabling communication among them through a distributed system. Currently, it supports an emergency response scenario during a nuclear disaster simulating virtual events from the involved partners.

The suite of adaptive search methods applied by REScUE [Coates et al(2011)] constructs in real time a near-optimal plan with an associated response team consisting of individuals together with their equipment and vehicles. Input for the methods includes detailed information about the environment such as the location, availability, and capabilities of resources. In addition, up-to-date information of casualties and the tasks needing to be undertaken are reported to the decision support system via emergency response agents at the scene of the event. Its agent-based simulation environment, STORMI, evaluates the emergency response to hypothetical major incidents. The INDIGO FP7 EU project [Indigo(2012)] also specializes in simulation technology for crisis management and training. A whiteboard is utilized to share the Common Operational Picture and associated information between the crisis centre and the mobile devices in the field.

Incidone [Lijnse et al(2012)] consists of incident info, loose dynamic action lists, data-based suggestions, and task composition. Intended users are coast guard watch officers in a centralized command centre. During incidents information about the situation and what is being done to resolve it is collected. All knowledge of the area comes from contacts outside, or from automated systems like Automatic Identification System. Additionally, Incidone disposes of plans in the form of hierarchical to-do lists that can be used to coordinate actions to resolve incidents.

WeKnowIT [WeKnowIt(2012), Diplaris et al(2011)] extracts information in emergency response scenarios (e.g. flooding or fire) from user-generated content.

Data is gathered from sources, such as emergency response workers at the scene, the general public observing or through other parties publishing information. The information is geo-located, either from metadata provided with the images or through an analysis of textual or visual data generating tags for the information and displayed on a map. An Emergency Alert service [Ovelgönne et al(2010)] works as an emergency call agent and informs social contacts and public authorities about the emergency situation. All the events and user interactions are represented by means of the WeKnowIt core ontology, CURIO[1], which defines resources holding user generated content. Likewise, the FP7 EU project PRONTO [Pronto(2012), Moi and Marterer(2012)] focuses on event recognition for intelligent resource management. It supports the extraction of relevant data from fire fighter radio chatter. Its semantic data store has an ontology of events defining the status of the system. Events are gathered from various kinds of sources, such as hardware devices (e.g. GPS) and user interaction with the system as well as from audio and video data streams. After aggregation, relevant information is filtered out for the users calculating events as output.

The ontology described by [Dilo and Zlatanova(2010), Fan and Zlatanova(2011)] presents a data model for the organization of dynamic data (operational and situational) for emergency response developed within the RGI-239 project 'Geographical Data Infrastructure for Disaster Management' (GDI4DM). It is early stage work on the possibility of applying ontologies to resolve the semantic interoperability inherent in emergency management. The model is derived from the organization of emergency response in the Netherlands investigating the information flow from processes performed by first responders: fire brigade, paramedics, police and municipality. It captures the type of disaster, the involvement of response sectors including their locations, consequences of the disaster for people, animals and infrastructure. The main objective of the model is to enable the extraction and processing of the information from spatial data sets and its distribution to different response units.

The novelty of the described approach in this paper is the seamless combination of a semantic reasoner and an event-based system. Incoming real-time data from device and sensor measurements during an emergency is updated into a semantic domain model. The reasoner automatically derives new knowledge from a formal definition of an emergency response model. Based on the inferred context the event-based system triggers events and alarms forwarded to the right units.

3 Layered Approach to Context-aware Event-based Reasoning

The proposed layered approach in this paper supports the advantages of both worlds of event-based and semantic reasoning. It splits up the reasoning on the domain model from the application-specific actions.

[1] http://socsem.open.ac.uk/ontologies/curio/

Figure 1 presents an overview of the architectural layers of the ASTUTE project focusing on the importance of the semantic domain model used by the rest of the application. The lowest layer consists of the semantic model covering the ontological definition of the domain concepts. The layer above, the *Context Engine*, encapsulates the translation of the semantic concepts into Java Bean objects. These objects are queried by the *Decision Engine* which utilizes Drools for the rule-based reasoning. This enables the transparent use of an actual semantic model by Drools resulting in triggering rules on the created objects in a timely manner. The *Data Aggregator* is responsible for capturing data from devices and sensors and formatting it as defined by the semantic model using the encapsulated concepts from the *Context Engine*.

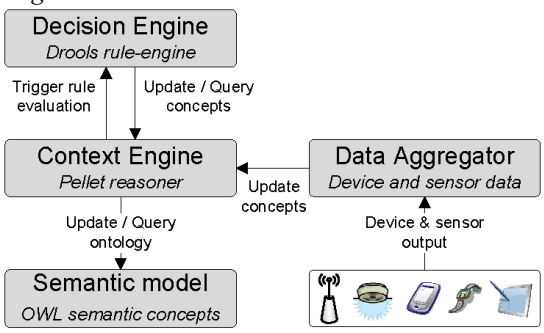


Figure 1: The layers of the reasoning framework of the ASTUTE project focusing on the importance of the semantic model used by the rest of the application.

The main purpose of the *Context Engine* is the low level domain model definition and inference through the use of rules. These rules are responsible for inferring knowledge from new data flowing into the system. The *Decision Engine* captures domain knowledge in the form of rules in order to determine which information needs to be send to whom at what moment. It relies on the *Context Engine* for delivering the interpreted raw context data.

4 Enriching a Rule-based Engine with Semantic Reasoning

This section details the encapsulation of a semantic model into the rule-based reasoning of a Drools engine.

4.1 Motivation for Integrating Semantic Reasoning into Rule-based Engines

Rule-based engines such as Drools [Bali(2009)] use forward chaining providing an integrated platform for rules, workflows and event processing. However, de-

spite its light weight paradigm, Drools lacks support for a formal model definition enabling inference of knowledge at runtime through reasoning on the model.

Semantic technologies support the definition of a domain model by capturing in a formal way the required context between different parties in a machine-processable common vocabulary also known as an ontology. A typical ontology language is OWL (Web Ontology Language) [McGuinness et al(2004)], which is a well-defined vocabulary for describing a domain having a foundation in description logics. An OWL ontology can be created using the Protégé Editor [Stanford(2011)] which provides support for OWL, RDF and XML Schema enabling the design of ontologies through a graphical interface. Additional knowledge is captured using rules defined through SWRL (SemanticWeb Rule Language) expressions and built-ins (SWRLB) such as comparisons (less/greater than) and math functions [Horrocks et al(2004)].

This formal domain allows the use of existing frameworks, such as the OWL-API [Horridge and Bechhofer(2011)] and Jena [Carroll et al(2004)], which provide a programmatic environment for RDF(S), OWL, SWRL, SPARQL and can be extended with reasoning engines enabling common operations on ontologies. Supported description logic reasoners such as Pellet [Sirin et al(2007)] realize the execution of data transformations, query processing and knowledge inference.

Two main concerns of the semantic approach are that it lacks a definition of temporal constraints in the SWRL rules and the execution of actions as result of a rule. The latter is particularly important in case a specific event should be triggered. Therefore, we aim at integrating the benefits of both approaches, a rule-based engine and a semantic reasoner, into an application supporting the invocation of events based on an underlying semantic domain model.

4.2 Automatic Java Beans Generation from Semantic Concepts

A main requirement for working with a rule-based engine such as Drools is the use of Java Beans. It should be possible to define rule conditions on Java objects in the following way: FireFighter(temperature>39). Due to the Java Bean nature of the objects, the background query uses the getTemperature method of the FireFighter object. Our goal is to define concepts such as FireFighter and BodyTemperature through an ontology in order to enable semantic reasoning and at the same time trigger Drools rules.

The querying of a semantic domain model usually requires the configuration of a reasoner such as Pellet and the manual encapsulation of the necessary concepts into Java classes in order to update and query their properties in a more general domain independent way. This manual work may be feasible for limited models but for larger use cases such as an emergency dispatching scenario the amount of manual work becomes difficult to maintain. Apart from the necessary testing, one needs to manually update the Java classes each time the ontology changes in order to keep the system synchronized with the domain model.

We automated this process through the adaptation of the code generation library of the Protégé Editor[2]. This tool automatically generates Java classes from semantic concepts translating their properties into class methods. The translation of the FireFighter concept having a BodyTemperature property to a Java object with corresponding convenience methods is presented in Table 1.

Table 1: Translation of semantic concepts and properties to Java classes and convenience methods.

Semantic Concept	**Concept property**
FireFighter	hasBodyTemperature(FireFighter, BodyTemperature)
Java Class	**Convenience method**
FireFighter()	List<BodyTemperature> getHasBodyTemperature() boolean hasHasBodyTemperature() addHasBodyTemperature(BodyTemperature) removeHasBodyTemperature(BodyTemperature)

Clearly the translation is nowhere near a Java Bean format and needs additional tweaking. The requirements from the proposed emergency response approach are:

1. Automatic generation of Java Bean-compliant classes, so that i) the code is easily updated when the model is updated and ii) the instances of these classes are consumed by the other architectural components, in particular the Drools implementation of the **Decision Engine**.
2. Integration of the **Decision Engine** into the generated code, so that its rules are evaluated whenever a Java Bean is updated or the model changes due to inference by the Pellet reasoner in the **Context Engine**.
3. Handling automatic classification of the domain concepts as result of necessary conditions or SWRL rules defined in the ontology.

In order to meet these needs, we adapted the code generation as follows:

1. Insertion of additional methods to the code generation template in accordance to the bean formatting. For example the following methods are automatically generated for the property hasCommander(FireFighter, Commander) which specifies that a FireFighter has a Commander:

```
public interface FireFighter extends User {
    /*
     * Property http://localhost/ASTUTE.owl#hasCommander
     */

    // Gets all values for the hasCommander property.
    Collection<? extends Commander> getAllCommander();

    // Gets the value for the hasCommander property.
    Commander getCommander();

    // Checks if it has a hasCommander property value.
    boolean hasCommander();

    // Adds a hasCommander property value.
```

[2] http://protegewiki.stanford.edu/wiki/Protege-OWL Code Generator

```
    void addCommander(Commander newCommander);

    // Removes a hasCommander property value.
    void removeCommander(Commander oldCommander);

    // Sets a hasCommander property value.
    void setCommander(Commander newCommander);
    ...
}
```

This also shows that all property definitions starting with has are automatically translated into methods without it. E.g. getBodyTemperature() instead of getHasBodyTemperature() from the property hasBodyTemperature.

2. Insertion of arguments for each object and data property. E.g. the FireFighter has an argument Commander additional to the generated methods which can be queried and updated by the application.
3. Insertion of the Pellet reasoner into the generated code. This enables the querying of the inferred axioms as result of the automatic reasoning on the domain model whenever object methods are invoked.
4. Integration of the **Decision Engine** into the code generation essential during object updates. This supports the automatic synchronization between the reasoner and the rule engine. Whenever an object is updated not only the corresponding semantic concept in the model is updated, but the Drools engine is also triggered to evaluate its rules. In this way the changes in the semantic model are propagated all the way up to Drools.
5. Automatic creation and removal of objects due to classification of concepts by the Pellet reasoner as result of SWRL rules or necessary conditions.
    - A concept is classified by the reasoner through inheritance from another concept as result of a predefined rule. For example, a SWRL rule can be stating that in case of several conditions such as high temperature and heart rate a FireFighter is a StressedUser. This means that the concept FireFighter should also be of the type StressedUser. As the reasoner automatically infers this in the ontology, the **Decision Engine** is also notified through the creation of a StressedUser out of the FireFighter object.
    - Necessary conditions are conditions that a semantic concept must fulfill and thus inherent to his domain specification. An example of a necessary condition is hasSolution (FaultyLightSensor, DoNotUseDataValue) defining that a possible solution to a faulty light sensor is simply not using its output data. Due to this, the moment there is an object FaultyLightSensor, the concept DoNotUseDataValue is automatically created if not present. This enables the **Decision Engine** to query for solutions to this event in a similar way the reasoner performs inference.

    In case the necessary conditions or classifications disappear, the classified individuals are removed automatically propagating up to the **Decision Engine**.
6. Java classes not only extend a specific class but also implement several interfaces allowing for multiple concept inheritance as supported by an ontology.

7. All data and object properties lacking a referenced property subject are automatically added to a main super class Thing extended by the rest of the classes. This resembles the equivalent Java Object class.

The result is the automatic translation of the semantic domain concepts into Java Beans that are used by Drools just like regular Beans. However, there is a slight change in the mentioned query: FireFighter(temperature>39). This is not possible as Temperature is also defined as a semantic concept and thus also translated into a Java object. In order to retrieve the specific temperature value, one should query its data property value hasValue and thus use: FireFighter(bodyTemperature.getValue()>39). On the other hand, as data properties specify value type definitions, it is possible to use them in the following way: BodyTemperature(value>39).

5 Illustrative Example to an Emergency Response Application

In order to validate the proposed approach, a scenario is designed capturing the state of a person such as a fire fighter walking through a building on fire. For this purpose, we developed a basic ontology, partially visualized in Figure 2(a), defining concepts such as users, devices, medical measurements and the relationships between them. Figure 2(b) presents the description of a fire fighter that can be a user having several properties such as location, activities (extinguishing fire), and medical measurements among which temperature, heart rate, oxygen level. Due to the specification of several types of context, such as physical, task, medical, one can define the criticality of a task or the level of a medical measurement. For example the activity ExtinguishingFire that is performed by fire fighters is defined as a Task with High Criticality as specified by the necessary condition below:

```
class ExtinguishingFire
    Superclasses: Task
        hasCriticality value High
```

Using these definitions one can define SWRL rules inferring the thresholds for high temperature or heart rate. The following rule defines that a BodyTemperature above 38 degrees is High.

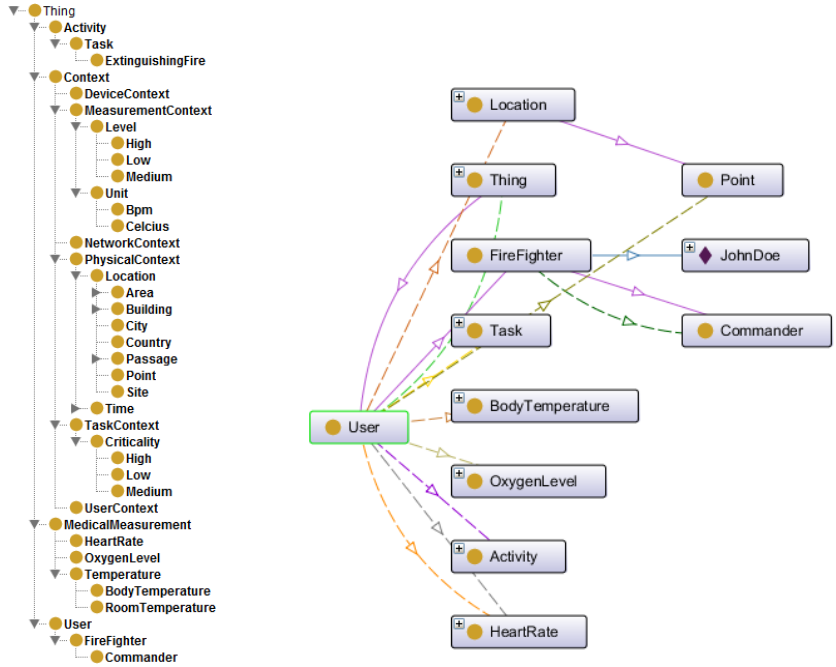
```
BodyTemperature(?bt), hasValue(?bt, ?btv),
greaterThan(?btv, 38) -> hasTemperatureLevel(?bt, High)
```

We could also specify personalized thresholds per user where one can define a maximum heart rate specific for each person. The following rule defines a user HeartRate higher that his personal defined maximum as too High.

```
HeartRate(?mhr), HeartRate(?uhr), User(?u),
hasHeartRate(?u, ?uhr), hasMaxHeartRate(?u, ?mhr),
hasValue(?mhr, ?mhrv), hasValue(?uhr, ?hrv),
greaterThan(?hrv, ?mhrv) -> hasHeartRateLevel(?uhr, High)
```

The results from inferring these rules are combined into additional rules. For example, we can define the rule that a user who performs a highly Critical Task

having High HeartRate and BodyTemperature and located in a Room with High RoomTemperature is Stressed.



(a) Partial ontology defining a user and device context.

(b) Specification of several user properties such as medical measurements, activity, location.

Figure 2: Basic ontology defining concepts such as users, devices, medical measurements and the relationships between them.

```
Activity(?a), hasCriticality(?a, High),
User(?u), hasActivity(?u, ?a),
BodyTemperature(?ht), HeartRate(?hh),
hasBodyTemperature(?u, ?ht), hasHeartRate(?u, ?hh),
hasHeartRateLevel(?hh, High), hasTemperatureLevel(?ht, High),
Room(?r), RoomTemperature(?hr),
hasRoomTemperature(?r, ?hr), hasTemperatureLevel(?hr, High),
hasLocation(?u, ?r) -> isStressed(?u, true)
```

These rules enable the tracking of a fire fighter's state during the fire fighting scenario. The environmental sensors sending various measurements such as temperature, location and heart rate register these values via the **Data Aggregator** in the **Context Engine**. With each new update the **Context Engine** fires the rules in the **Decision Engine**. If the following Drools rule is defined that alerts the commander that the fire fighter is stressed, it will be triggered evaluating the condition of the fire fighter.

```
rule "Firefighter stressed"
no-loop
  when
    $p : FireFighter(isStressed==true)
```

```
  then
    $hmi.sendMessage($p.getCommander().getName(),
    "Firefighter " + $p.getName() + " is stressed", "ALERT");
end
```

The moment the **Context Engine** receives a new update, inferring that the fire fighter is actually stressed because of for instance elevated heart rate, the **Decision Engine** fires this rule and the commander is alerted of his team member's state.

6 Conclusions

This paper presents a novel approach to an emergency response application used in scenarios such as performed by a fire department. The approach implements the integration of a semantic reasoning engine into an event-based system supporting the use of a formal domain definition. The semantic model describing the users and device context is used to automatically generate corresponding Java Bean objects. These objects are consumed by the rest of the application just like normal objects with the exception that the underlying model is updated together with the object updates. Additionally, inference on the model performed by the semantic reasoner is automatically synchronized with the other architectural components resulting in the triggering of events based on scenario specific conditions.

The approach is validated on a fire fighting scenario where the state of a fire fighter is constantly monitored based on new device and sensor measurements such as body temperature, heart rate, and location. While the reasoner keeps track of the fire fighter's context, the event-based engine issues alerts to the rest of the team in case his state deteriorates.

Future work should incorporate an extensive evaluation of the proposed scenario consisting of capturing the state of a complete team of fire fighters using real-time device and sensor readings during an emergency.

Acknowledgement

A. Hristoskova would like to thank the Special Research Fund of Ghent University (BOF) for financial support through her PhD grant. This work is funded by the ASTUTE project from the ARTEMIS Joint Undertaking, Grant agreement no.: 269334.

References

ASTUTE (2011-2014), Pro-active decision support for data-intensive environments. http://www.astute-project.eu/ .

Indigo (2012), Crisis Management Solutions. http://indigo.diginext.fr/ .

Pronto (2012), Event Recognition for Intelligent Resource Management. http://www.ict-pronto.org/ .

SocEDA (2012). SOCial Event Driven Architecture. http://research.petalslink.org/display/soceda/SocEDA+Overview/ .

WeKnowIt (2012). http://www.weknowit.eu/ .
Bal, H., Kielmann, T., Palmer, N., and Kemp, R., (2012). Interdroid. http://interdroid.net/ .
Bali (2009). Drools JBoss Rules 5.0 Developer's Guide. Packt Publishing, Limited.
Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. (2004). Jena: implementing the semantic web recommendations. In: 13th International World Wide Web conference on Alternate track papers & posters, ACM, pp 74–83.
Coates, G., Hawe, G., Wilson, D., and Crouch, R. (2011). Adaptive co-ordinated emergency response to rapidly evolving large-scale unprecedented events (rescue). In: Proceedings of 8th International Conference on Information Systems for Crisis Response and Management–ISCRAM.
Dilo, A., and Zlatanova, S. (2010). Data modelling for emergency response.
Diplaris, S., Sonnenbichler, A., Kaczanowski, T., Mylonas, P., Scherp, A., Janik, M., Papadopoulos, S., Ovelgoenne, M., and Kompatsiaris, Y. (2011). Emerging, collective intelligence for personal, organisational and social use. Next Generation Data Technologies for Collective Computational Intelligence pp 527–573.
Endsley, M.R. (1995). Toward a theory of situation awareness in dynamic systems. Human Factors: The Journal of the Human Factors and Ergonomics Society 37(1): 32–64.
Fan, Z., and Zlatanova, S. (2011). Exploring ontologies for semantic interoperability of data in emergency response. Applied Geomatics 3(2):109–122.
Horridge, M., and Bechhofer, S. (2011). The owl api: A java api for owl ontologies. Semantic Web 2(1):11–21.
Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/.
Lijnse, B., Jansen, J.M., and Plasmeijer, R. (2012). Incidone: A task-oriented incident coordination tool. In: 9th International Conference on Information Systems for Crisis Response and Management.
McGuinness, D., Van Harmelen, F., et al (2004). OWL web ontology language overview. http://www.w3.org/TR/owl-features/.
Moi, M., and Marterer, R. (2012). An architecture for distributed, event-driven systems to collect and analyze data in emergency operations and training exercises. In: Proceedings of 8th International Conference on Information Systems for Crisis Response and Management–ISCRAM.
Ovelgönne, M., Sonnenbichler, A., and Geyer-Schulz, A. (2010). Social emergency alert service-a location-based privacy-aware personal safety service. In: 2010 Fourth International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST), IEEE, pp 84–89.
Palmer, N., Kemp, R., Kielmann, T., and Bal, H. (2012). Raven: Using smartphones for collaborative disaster data collection. In: 9th International Conference on Information Systems for Crisis Response and Management.
Paraiso, F., Hermosillo, G., Rouvoy, R., Merle, P., Seinturier, L., et al (2012). A middleware platform to federate complex event processing. In: Sixteenth IEEE International EDOC Conference.
Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical owldl reasoner.Web Semantics: science, services and agents on theWorldWideWeb 5(2):51–53.
Stanford University (2011). Protégé, Stanford University. http://protege.stanford.edu/
Tsiporkova, E., Tourwé, T., González-Deleito, N., and Hristoskova, A. (2012). Ontology-driven Multimodal Interface Design for an Emergency Response Application. Proceedings of the 9th International ISCRAM Conference. Vancouver, Canada, April 2012.