OpenKnowledge

FP6-027253

# Interaction Models for eResponse

Maurizio Marchese[1], Lorenzino Vaccari[1], Gaia Trecarichi[1], Pavel Shvaiko[1], Juan Pane[1], Nardine Osman[2] and Fiona McNeill[2]

[1] DISI, University of Trento
[2] University of Edinburgh

Report Version: final
Report Preparation Date: January 10th, 2008
Classification: Deliverable 6.7
Contract Start Date: 1.1.2006 Duration: 36 months
Project Co-ordinator: University of Edinburgh (David Robertson)
Partners: IIIA(CSIC) Barcelona
Vrije Universiteit Amsterdam
University of Edinburgh
KMI, Open University
University of Southampton
University of Trento

## INTRODUCTION TO THE DELIVERABLE

In this deliverable we report on the work carried out towards the definition and the development of appropriate interaction models for the eResponse testbed for the OpenKnowledge system. In particular, we adopted, as a real-life scenario, the flooding event in the province of Trento. The main goal of the present work, is to show how an approach based on interaction models, in combination with an appropriate infrastructure, could be used to manage both the sharing of relevant geographical information and the coordination activities between the different actors involved in eResponse events.

More specifically, the report focuses on:

- a P2P formalization of the eResponse scenario based on the emergency organizational model - as used in Trentino, Italy;
- the usage of the Lightweight Coordination Calculus (LCC) to express P2P style interactions and to provide an open framework for coalition formation and Web service composition;
- the provision of a eResponse simulation environment in which to evaluate how suitable interaction models are in coordinating peers in real time;
- the evaluation of the proposed approach to support typical emergency response tasks, requiring both data sharing coordination and peers structured and dynamic coordination.
- an application of the approximate structure preserving semantic matching algorithm, developed in WP3 of the OpenKnowledge project, within the eResponse domain.

The rest of the deliverable is organized as the collection of two papers - and related annexes - that describe the approach, the related work and discuss the results so far achieved. Both papers have been submitted to 5th Intl-Conference on Information Systems for Crisis Response and Management - http://www.iscram.org/ to be held in Washington, DC, USA, May 4-7, 2008.

In the first paper - titled "Interaction models to support peer coordination in crisis management"- we present the OpenKnowledge approach based on interaction models distributed through a peer to peer infrastructure and we show how it can be applied in the context of crisis management to support coalition formation and process coordination in open environments. We first focus on a short description of the specific use case that we are using to ground our approach, i.e. the evacuation plan management during a flooding event. We then, present the developed eResponse environment by describing its main components, namely, the eResponse simulator and the peer network: along with the description of the system's components (simulator, peers, services and interaction models) we present some interaction models detailed examples. Next, we briefly present a preliminary prototype of the overall system and we assess the simulation of initial emergency response coordination activities. In the final sections, we discuss related and future work. The paper is complemented - in this document - by two appendixes, which present and analyse the technical details of the developed interaction models: (i) for the eResponse simulator (Appendix 1) and (ii) for the selected use case, i.e .the evacuation plan (appendix 2).

In the second paper - titled "An Application of Approximate Ontology Matching in eResponse" - we focus on a data sharing coordination among geoservices. The issue here is on how to interpreter structure and terms in interaction models in order to support peers goal coordination. In the proposed approach, we achieve this by dynamically matching terms in the interaction models. This can happen both at design time (i.e., when synthesising different interactions models) and at execution time (i.e., when running them to perform specific tasks). To this end, we use the approximate structure preserving semantic matching algorithm developed in WP3 of the OpenKnowledge project, in order to reduce the semantic heterogeneity problem as required by the scenario. Preliminary evaluation of the approach used is presented together with relevant related work.

# Interaction models to support peer coordination in crisis management

**Maurizio Marchese**
University of Trento
maurizio.marchese@unitn.it

**Lorenzino Vaccari**
University of Trento
vaccari@disi.unitn.it

**Gaia Trecarichi**
University of Trento
gtrecari@disi.unitn.it

**Nardine Osman**
University of Edinburgh
N.Osman@sms.ed.ac.uk

**Fiona McNeill**
University of Edinburgh
f.j.mcneill@ed.ac.uk

## ABSTRACT

All phases of emergency management activities - that we will reference hereafter as eResponse activities - depend on data from a variety of sources and involve a range of different organizations and teams at various administrative levels with their own systems and services. The existence of numerous and different actors, policies, procedures, data standards and systems, results in coordination problems with respect to data analysis, information delivery and resources management. In this paper we present a novel approach based on interaction models distributed through a peer to peer infrastructure and we show how it can be applied in the context of crisis management to support coalition formation and process coordination in open environments. In particular, a prototype eResponse simulation system – built on a P2P infrastructure – has been developed to share and execute interaction models describing common coordination tasks in emergency response domain. Preliminary evaluation of the proposed framework demonstrates its capability to support such eResponse tasks.

## Keywords

Interaction Modeling, P2P Networks, Process Coordination, Knowledge Sharing, Crisis Management, Emergencies Simulation.

## INTRODUCTION

At 23:00 on November 4th, 1966, the river Adige, the main river of the Trentino region in Italy, broke its banks at different sites and flooded the majority of the territory of the Trentino main town, Trento. Moreover a considerable amount of oil, from housing heating systems and fuel depositories and petrol stations, mixed with the mud waters of the river. The majority of the Trento population as well as surrounding areas had been affected. Today, in 2006, the flooding of the Adige river is still the most probable emergency event in the Trentino region. In such situation, knowledge of the state of the emergency is vital. For example: (i) knowledge of the viability structures affected by the flooding event; (ii) knowledge of all public buildings that are contained in the flooding area are critical sites and potential risk factors since they might contain a high number of persons; (iii) knowledge about the service infrastructures - such as the electricity network, the waterworks network, the pipeline network, the telecommunication network - is of uttermost relevance during emergency events. The primary goal of the municipality emergency plan is to evacuate the population (ca. 110.000 persons) effectively and rapidly from the critical area.

Flood response provides an example of an important and potentially devastating type of emergency in which there can be enormous impact on people, property and infrastructure. Such an emergency includes a range of potential uses for geo-informatics services, links to large scale sensor grids, potential for linking to automated or semi-automated response systems, and the needs for coordinating processes with many organizations, and agencies. Timely decisions and properly executed processes can make an enormous impact on the final outcome.

The scenario above aims at showing some of the key elements needed in eResponse situations. In particular emergency management activities are developed and implemented through the essential analysis of information and the coordination of the involved peers (both institutional, like emergency personnel, army, volunteers, etc as well as common people involved in the crisis). The existence of numerous and different actors, policies, processes, data standards and systems, results in coordination problems with respect to data analysis, information delivery and resources management, all critical elements of emergency response management.

In our current research, we want to explore the flexibility and adaptability of an interaction-driven mechanism

for knowledge sharing that relies on a distributed - Peer to Peer - infrastructure. At the core of the approach is a specific view of the semantics of both service and agent coordination (as proposed in Robertson at al, 2007), where peers share explicit knowledge of the "interactions" in which they are engaged and these models of interaction are used operationally as the anchor for describing the semantics of the interaction. Instead of requiring a universal semantics across peers, the approach require only that semantics is consistent (separately) for each instance of an interaction. This is analogous to the use in human affairs of contracts, which are devices for standardising and sharing just those aspects of semantics necessary for the integrity of specific interactions.
In this paper, we apply the above approach in the context of crisis management. The nature of the eResponse domain is such that process-aware systems are beneficial to prevent chaotic and uncontrolled conditions. Nevertheless, taking into account adaptability is fundamental to handle unexpected situations (i.e. sudden road blockage, fast and unpredicted events, etc.) which most likely happen in emergency situations. While the general vision of interaction protocols accounts for the "structured coordination" requirement of the problem, the adoption of models specifically designed to explicit interactions in a P2P fashion and passed through an underling open infrastructure accounts for the support to flexibility and dynamicity.
The main contributions of the paper are:

- the usage of a protocol language that, expressing P2P style interactions and relying on a distributed infrastructure, provides a mechanism for coalition formation and Web service composition (Robertson at al, 2006; Robertson at al, 2007);
- the provision of a simulation environment in which to evaluate how suitable interaction models are in coordinating peers in real time;
- the evaluation of the proposed approach to support typical emergency response tasks, requiring both structured and dynamic coordination.

In what follows, we first focus on a short description of the specific use case that we are using to ground our approach, i.e. the evacuation plan management during a flooding event. We then, present the eResponse system by describing its main components, namely, the eResponse simulator and the peer network: along with the description of the system's components (simulator, peers, services and interaction models) we present some interaction models detailed examples. Next, we briefly present a preliminary prototype of the overall system and we assess the simulation of initial emergency response coordination activities. In the final sections, we discuss related and future work.

## FLOODING EVACUATION USE CASE

For the analysis and simulation of the flooding crisis emergency, we have grounded our work on the actual emergency plan in the Trentino region (Provincia Autonoma di Trento, 2005; Municipality of Trento, 2002). We have focused on the evacuation of the people from the probable flooding areas to the refuge centers, outside these areas. This specific use case assumes that the institutional emergency peers have been alerted to be prepared to face a possible flooding of the river Adige in the next six hours. In the emergency room, the emergency coordinator is present together with the coordinators of the main different agencies involved in the evacuation plan. Namely, in our case: the firefighters coordinator, the police officers coordinator, the medical personnel coordinator. All other involved institutional peers are in the concerned area and they include:

- Fire fighters (~50): supervision of evacuation plan at meeting points and at refuge centers;
- Police officers (~22): control of road gates;
- Medical staff (~88): supervision of people health conditions;
- Bus/Ambulance Drivers (~44).

A number of "system" peers are also present in the scenario: they represent a possible set of digital services that support the evacuation plan, among others: - The emergency coordination data service: it is based on the emergency database and it maintains the current status of the enacted emergency plan, - SDI Services for map information: they mainly provide gazetteer services, map services, download (feature) services as in (Vaccari, Ivanyukovich and Marchese, 2005); - The weather forecast service; - The sensor network: it provides the water level of the river; - The route service: it provides routing information including updates on blocked roads.

Figure 1 shows a schematic view of the involved peers together with the flow of the main interactions (arrows) during the evacuation plan in the selected eResponse coordination activity.

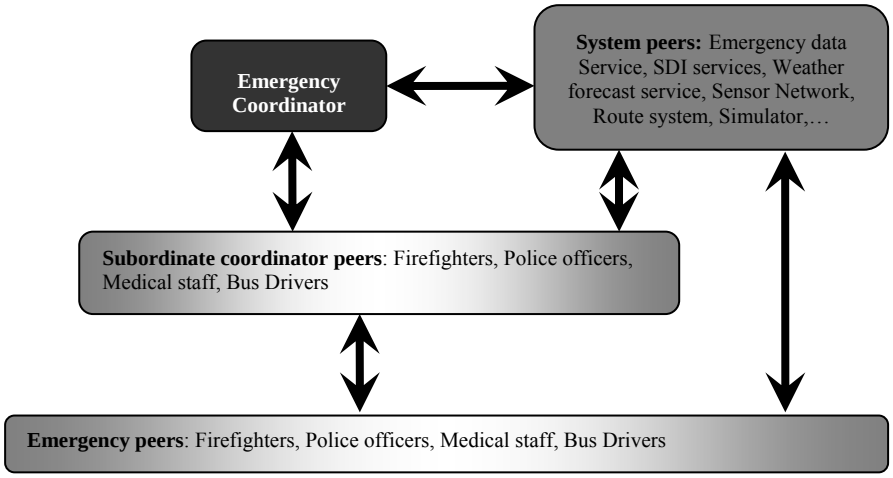


**Figure 1: Schematic view of involved peers and main interactions in the evacuation plan**

The emergency coordinator evaluates the information collected by the peers and the system services, and propagates its plans to the subordinate coordinators, that are responsible to act the plan by distributing plans/sub-plans to each organization or group involved in the crisis management. In our use case, the evacuation plan is enacted by the emergency coordinator, by (i) propagating the alarm to the bus drivers and assign them the appropriate destination; (ii) propagating the alarm to the subordinate coordinators; (iii) sending the evacuation alarm and information to the citizens; (iv) continuingly monitoring the crisis information from all available sources (e.g. sensors, institutional peers, volunteers, citizens) and taking appropriate actions.

Each subordinate peer propagates the directives to its group peers. As an example the firefighters coordinator peer assigns to each firefighter (supplied with a vehicle) its destination point (e.g. meeting point, refuge center, etc). Each emergency peer can execute some specific tasks, related to its location and to the tasks assigned by its coordinator.
For example, the firefighter, after receiving the assignment of its destination point, has to reach it. It can use its own knowledge, or some digital services provided by a system peer like e.g. a route service. Once the firefighter arrives to the final destination it can perform its planned tasks, like manage census information, evaluate criticality of the meeting point, handle the evacuation plan etc.

## THE E-RESPONSE SYSTEM

The eResponse system we are developing is used: (1) to model the behaviour of each peer involved in an e-response activity, (2) to execute predefined interaction models within a P2P infrastructure and (3) to visualize and analyze a simulated coordination task through a Graphical User Interface (GUI). It is composed of two major components: the eResponse simulator and the peer network. Figure 2 sketches the overall architecture of the system. Examples of peer types have been discussed in the previous section (i.e. Figure 1). In the next two subsections, we will focus on the eResponse simulator and the peer network, respectively.

**The eResponse Simulator**

The simulator is composed of three peers: the controller, the flood sub-simulator, and the visualiser (see Figure 2). The controller is the core of the simulator: it drives the simulation cycles. The controller has one main goal: to keep track of the current state of the world. In order to achieve that, the controller needs to know what changes are happening to the world and update its state accordingly. Of course, after updating its state, it should inform the relevant peers of these changes as well. The goal of the flood simulator is to simulate the flood. It is composed of a set of predefined equations that define how the flood evolves with time. The goal of the visualiser is to simply visualise what is happening in the flood area. Simulation is then divided into cycles, which are driven by the controller. The steps of one simulation cycle are:

1. The controller shares the initial topology of the world with others simulator peers: the flood sub-simulator and the visualiser

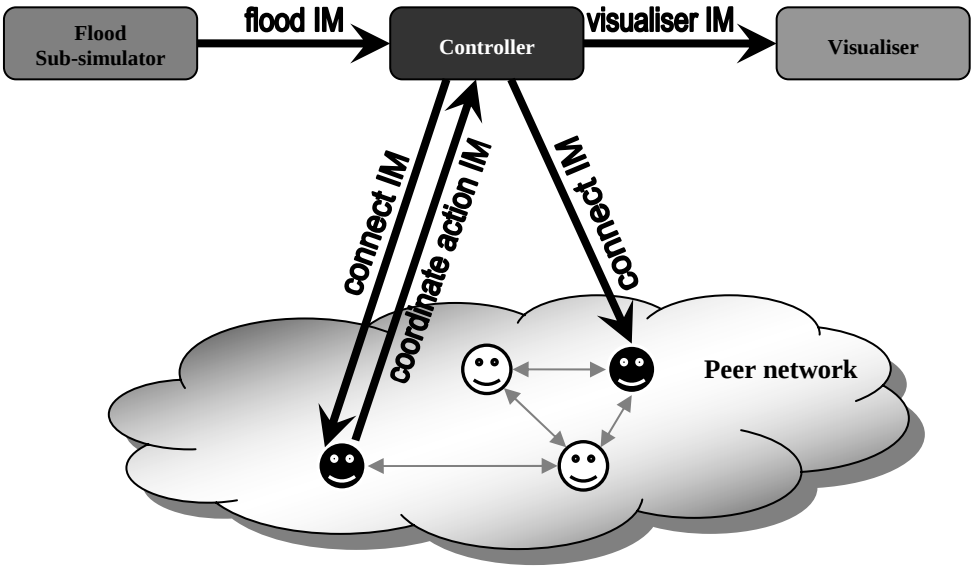


**Figure 2: The eResponse system's architecture**

The controller receives information about the changes that happened to the world:

a. It receives the flood changes from flood sub-simulator
b. It receives other changes from the peers in the peer network that caused these changes (and verifies their validity)

2. The controller sends information about the changes that happened in the world:
   c. It sends a list of all the changes to the simulator's visualiser
   d. It sends changes that occurred in a peer's vicinity to each peer in the peer network
3. The controller updates the time step, and repeats by going back to step 1

The flow of the interactions between the controller and other peers is also depicted in Figure 2. Each black arrow represents a different interaction model, which also represents the flow of information between peers. For example, the flood interaction model "flood IM" is used to allow the flood sub-simulator to communicate with the controller, sending it flood changes at each time step.

Note that the controller (or the simulator in general) does not interfere or help coordinate peer's actions in the peer network. It is simply used to simulate the real world. For instance, peers cannot ask the simulator questions about the world, such as where the location of something is or what the flood level at a given point is. Nevertheless, in the real world, peers (whether humans or sensors) are able to sense certain things in their vicinity. Therefore, in our simulation, this information will be provided by the controller. At every time step, the controller sends sensory info for each connected peer through the "connect" interaction model. Also, the real world usually prevents humans (or even sensors) from performing certain unrealistic actions. For example, one person may try to drive a car in a flooded road, but will fail even if he insisted. Usually, the rules of Physics of the real world prevent such unrealistic actions. Again, in our simulation, the controller will verify whether certain actions are legal or not before they are performed, and if a certain action is illegal, the peer is informed of the reason of failure. This is done through the "coordinate action" interaction model (see Figure 2). Note that any peer that either needs to perform an action (such a police man closing a road) or needs to receive sensory info about its vicinity (such as a sensor) needs to be connected to the simulator. We call these peers physical peers. They are represented as black faces in the peer network of Figure 2. Of course, not all peers need to connect to the controller. Just like in real life, non physical peers, such as a web service that provides information about the weather, does not need to communicate with the controller but with other peers in the peer network (such as a sensor, a human user, ect.). Non physical peers are represented as white faces in the peer network of Figure 2.

**The Peer network**

In this section we first describe the Lightweight Coordination Calculus (LCC), that is, the communication language employed to implement the interactions among peers acting in our Peer-to-Peer (P2P) network. After briefly introducing it, we then report a selected coordination problem taken from the scenario depicted in the previous section, and explain in some details the interaction models which can be derived to describe it. Finally, we show a preliminary prototype which has been implemented to execute and analyze the evolution of the interactions within a peer-network.

*Introduction to LCC*

LCC is a protocol language used to describe interactions among distributed processes, e.g., agents, web services (Robertson, 2004-1; Robertson, 2004-2). LCC was designed specifically for expressing P2P style interactions within multi-agent systems, i.e., without any central control; henceforth, it is well suited for modeling coordination of software components running in an open environment. Its main characteristics are the flexibility, the modularity and the neutrality to the distributed communication infrastructure.
Interactions in LCC are expressed as the message passing behaviours associated with roles. The most basic behaviours are to send or receive messages, where sending a message may be conditional on satisfying a constraint (precondition) and receiving a message may imply constraints (postcondition) on the agent accepting it. As an example, a basic LCC interaction is shown in Figure 3 where double and single arrows indicate respectively message passing and constraints to satisfy:

$$
\begin{aligned}
&a(r1, A1) :: \\
&\qquad ask(X) \Rightarrow a(r2, A2) \leftarrow need(X)\, then \\
&\qquad update(X) \leftarrow return(X) \Leftarrow a(r2, A2) \\
&a(r2, A2) :: \\
&\qquad ask(X) \Leftarrow a(r1, A1) \\
&\qquad return(X) \Rightarrow a(r1, A1) \leftarrow get(X)
\end{aligned}
$$

**Figure 3: Basic LCC interaction**

According to the interaction shown in Figure 3, the agent A1 playing the role r1 verifies if it needs the info X (precondition need(X)); if yes, A1 asks for X to the agent A2 playing the role r2 by sending the message ask(X). A2 receives the message ask(X) from A1 and then get the info X (precondition get(X)) before sending back a reply to A1 through the message return(X). After having received the message return(X), A1 updates its knowledge (postcondition update(X)).

The constraints embedded into the protocol express its semantic and could be written as first-order logic predicates (i.e., in Prolog) as well as methods in an object-oriented language (i.e., in Java). Furthermore, these constraints could hide simple functionalities (i.e., provided by web services) as well as very complex AI algorithm. This is the characteristic of modularity previously mentioned that allow to separate the protocol from the agent or service engineering.

While performing the protocol, peers can therefore exchange messages, satisfy constraints before (after) messages are sent (received) and jump from one role to another so that a flexible interaction mechanism is enabled still following a structured policy, this being absolutely necessary for team-execution of coordinated tasks.

*Flooding evacuation use case and selected examples of Interaction Models*

We designed and implemented a sub-scenario where different participating actors, having different roles, coordinate each other and rely on existing service components. This scenario is currently running on a Prolog based LCC parser engine and on a local P2P network. It constitutes a simulated scenario where peers are either emergency peers, that is, agents acting on behalf of human emergency personnel, or system peers, that is, digital services providing information that are essential for coordinated task execution.

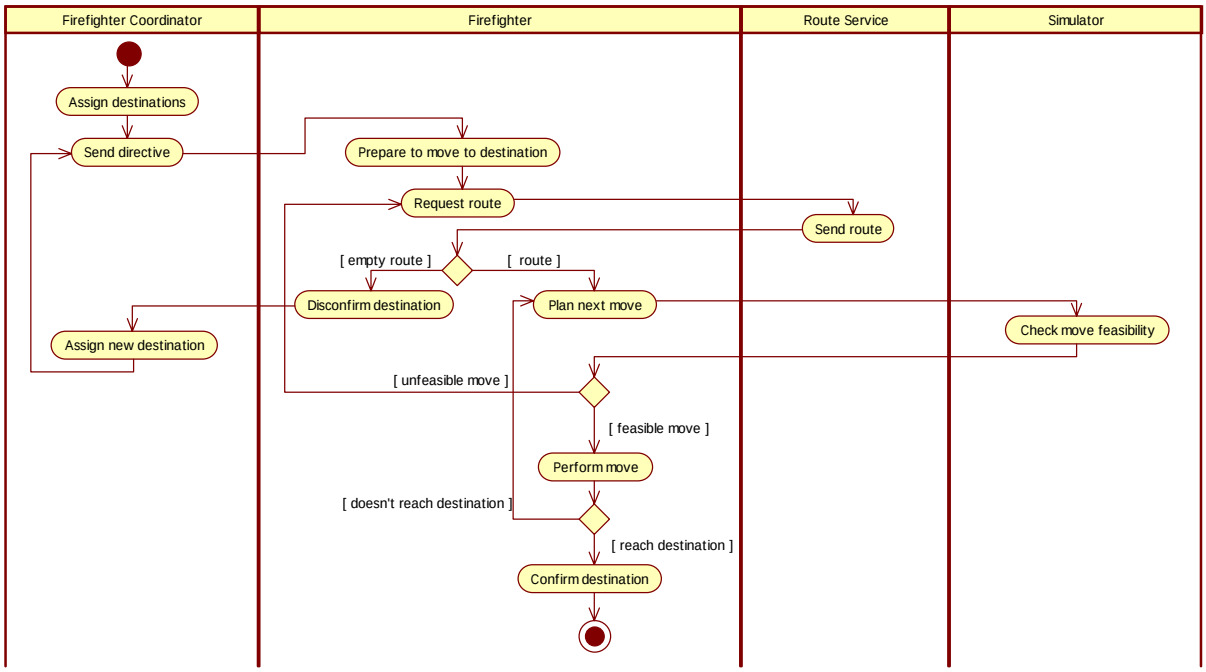


**Figura 4: Activity diagram of the coordination task "Go to Destination"**

Our sub-scenario - of the overall scenario flooding evacuation use case - is related to the phase where an emergency coordinator (the firefighter coordinator in this case) sends a directive to its organizational peers (firefighters) that have to accomplish it. The coordination problem expressed above can be described in a verbose form as follow:

> *After having found out all the available firefighters, the firefighter coordinator (ffc) assigns to each of them a specific destination; in order to find a route, the firefighter will rely on the route service component whose aim is to provide a suitable path to the requester; the firefighter communicates each action to the simulator which check the feasibility of the action (i.e., the action of moving could be impossible because of road blocked); in case of blocked road the firefighter asks the route service an alternative path (which doesn't contain the road blocked) and keep trying alternatives till either there are free paths connecting the destination or this latter is unreachable; if the firefighter reaches the previously assigned destination he/her will confirm it to the ffc otherwise he/her will inform the cheaf about the negative result; the ffc will reassign a destination to the firefighter who couldn't reach the previous place.*

This description can be sketched in a graphical representation such as the activity diagram shown in Figure 4. The diagram shows the workflow related to the coordination task "Go to Destination". An interaction model can be derived either from the narrative description above or from the activity diagram; it is expressed in a compact, formal language ( in our case, in LCC).
As example of the implementation and usage of the interaction model language, Figures 5, 6 and 7 show some excerpts of LCC code related to the firefighter coordinator, firefighter and route service roles respectively. It is worth to notice here that the LCC code presented in this paper is part of the code needed to run the interaction. The whole code is more lengthy since it entails also the role of the simulator peer and its interactions with the network peers. However, for our purposes here - to indicate how to model the possible interactions among the network peers – we restrict our description to these excerpts

i. Fire-fighter coordinator

The firefighter coordinator, *FFC*, initiates the coordination task by entering the role *firefighter_coordinator*. According to this role a list of available firefighters, *FFL*, must be retrieved in order to send an alert message to each of them. The role involves a recursion over the list *FFL* so that a destination meeting point (*MP*) can be assigned and an alert message can be sent to each firefighter *FFL_H*, in *FFL*. After having sent the messages to all the firefighters, the *FFC* assumes the role *firefighter_coordinator3*. Here the *FFC* receives a confirmation (or disconfirmation) of reached (not reached) destination *MP* from a given firefighter *Id* (in Figure 5 *'action_performer2'* and *'replanner'* are roles assumed by the firefighter role) and the role recurses. If a disconfirmation is received, the *FFC* assigns to the firefighter Id a new destination – *NewMP* - which is then embedded in the alert message sent back.

---

$$a(\ firefighter\_coordinator(\ FFL\ ),FFC\ )::$$
$$\left(\begin{array}{l}\left(\begin{array}{l} alert(MP)\ \Rightarrow\ a(firefighter,FFL\_H)\ \leftarrow\ FFL=[FFL\_H\mid FFL\_T]\ and \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad assign(MP,FFL\_H) \\ then\quad a(\ firefighter\_coordinator(FFL\_T),FFC) \end{array}\right)\ or \\ null \leftarrow FFL=[\ ] \end{array}\right)\ then$$
$$a(\ firefighter\_coordinator3,FFC\ ).$$

$$a(\ firefighter\_coordinator3,FFC\ )::$$
$$\left(\begin{array}{l} confirm(Id,MP)\ \Leftarrow\ a(action\_performer2(\_,\_,\_,\_),Id)\ or \\ \left(\begin{array}{l} disconfirm(Id,MP)\ \Leftarrow\ a(replanner(\_,\_,\_),Id)\ \ then \\ \\ alert(NewMP)\ \Rightarrow\ a(firefighter,Id)\leftarrow assign\_newDest(MP,NewMP,Id) \end{array}\right)\ then \end{array}\right)$$
$$a(\ firefighter\_coordinator3,FFC\ ).$$

---

**Figure 5: LCC fragment for the firefighter coordinator role**

ii. Fire-fighters

Each firefighter, *FF*, receives from the firefighter coordinator, *FFC*, an alert message stating that the destination *MP* has to be reached. Once the message is received, the firefighter, *FF*, assumes the role of *route_finder*. If the firefighter, *FF*, is not yet at the location *Dest* that must be reached, a vehicle will be needed together with the name of a digital service able to provide a route. A message for requesting a route between two locations (*Pos* and *Dest*) is then sent to the route service *RS*. Once the firefighter *FF* receives the requested path together with its sub-paths through a message sent by the route service *RS*, he/her stores the path in its local memory and assumes the role *action_performer* in order to communicate the move action to the simulator which in turn will check its feasibility (more details in the activity diagram in figure 4).

iii. Route Service

The route service, *RS*, after having received a route request from a route finder *Id*, selects a path from node *From* to node *To*, split the path in sub-paths and sends them back to the route finder *Id*. The route service role then recurses to be able to accept other requests. The route service can receive two types of messages from a route finder:

- A message asking for a path from A to B (a vehicle is also specified);
- A message asking for a path from A to B (a vehicle is also specified) not including certain sub-paths (i.e., the blocked subpaths).

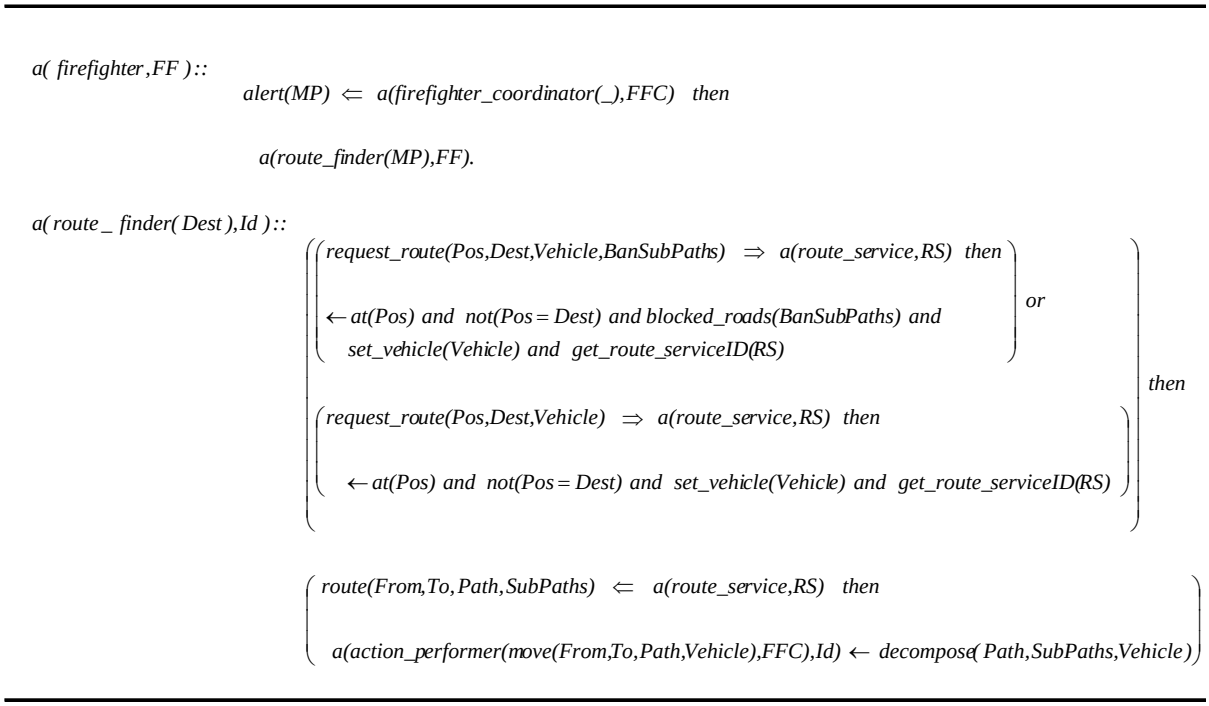


**Figure 6: LCC fragment for the firefighter role**

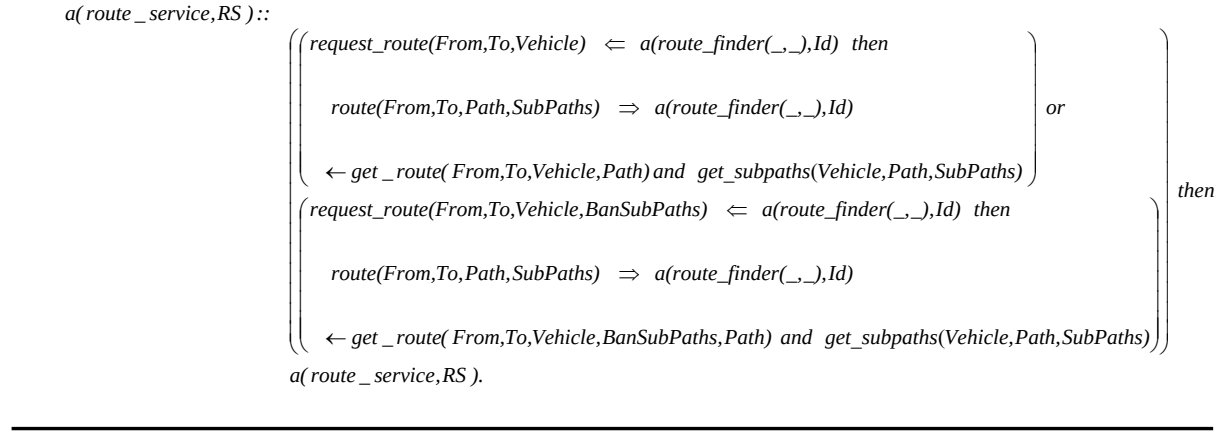


**Figure 7: LCC fragment for the route service role**

*Preliminary Prototype*

We developed an initial prototype in which the coordination activities between the network peers can be executed, visualised and analysed to evaluate the effectiveness of the LCC protocol. In our first prototype, the ongoing simulation and the resulting movements of the emergency peers are visualized through a GUI (see Figure 8). The GUI is a draft version of the control panel used by the emergency coordinators. Through the emergency GUI, users can visualize the map of the emergency situation.

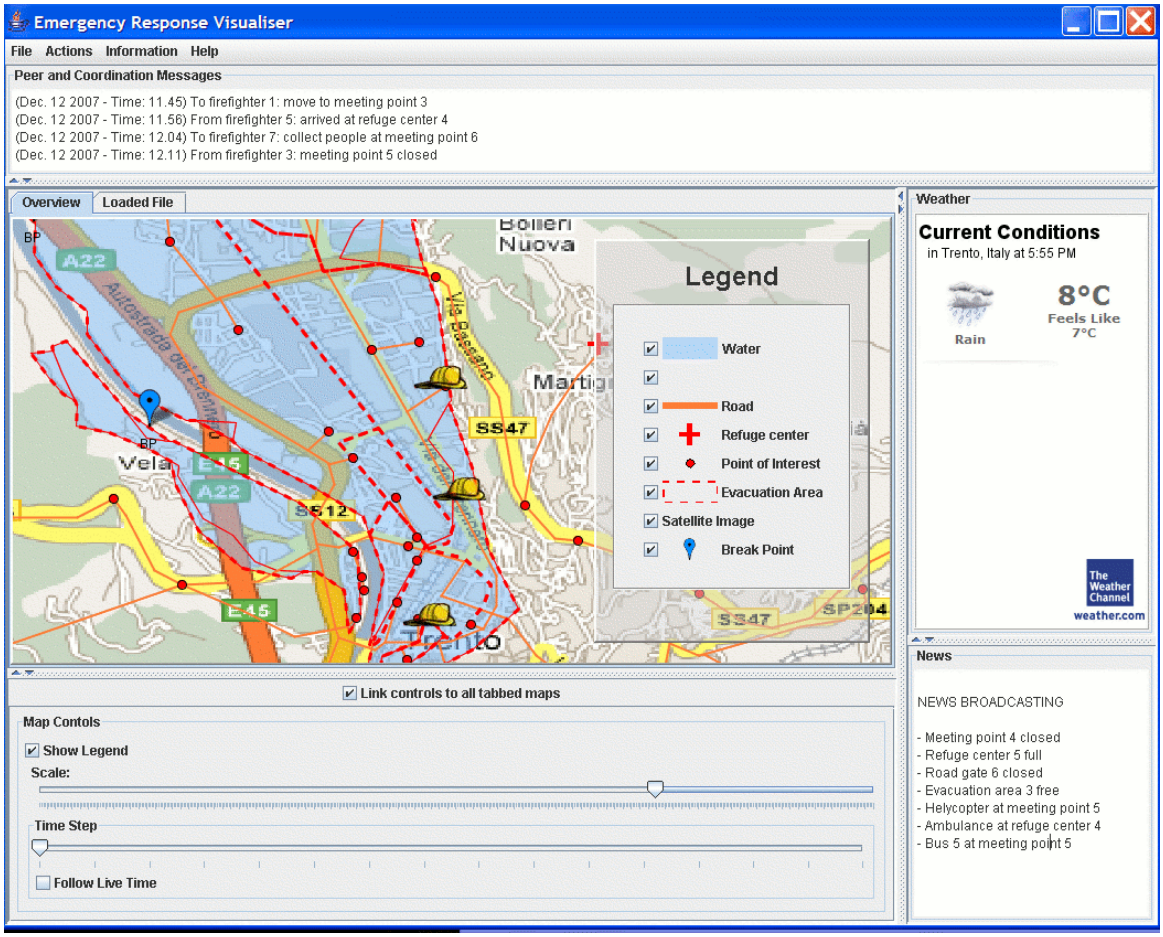


**Figure 8: The emergency GUI**

The map shows static geographic datasets (topographic map, probable flooding areas, escape roads, meeting points, refuge centers, sensor network) as well dynamic datasets (the figure shows the position of the firefighters involved in the simulation). Moreover, through the GUI, the emergency coordinators can perform actions (enact the emergency plan, recall digital services, change the map legend, search for other GIS datasets, send statements to the emergency peers, etc) as well as ask information about the emergency situation (i.e., evacuated people, list of the emergency peers, blocked roads, situation of the meeting points and of the refuge centers, etc). The main components of the GUI are:

- The main menu in the upper part of the GUI
- The peer and coordination messages frame: it shows the messages exchanged between the coordination peer and the emergency peers (different for each peer)
- The weather forecast frame: useful to analyze the current weather conditions
- The news frame: it shows the general information about the emergency situation (identical for all the peers)
- The map frame: it shows, together with a real map of Trento and some static interesting points (coordination center, the meeting points, the refuge points, etc.), the movement of the peers during a simulated coordination task.

We have run several simulations involving the peers types and the interactions described previously. In the one visualize in Figure 8, we have set 4 emergency peers (1 firefighter coordinator and 3 firefighters), 1 digital service (the route service) and the simulator peer; the coordination task was to follow a directive sent by the firefighter coordinator to its personnel. The directive consisted in moving to a given destination. We set 3 different destinations (meeting points) for each firefighter and initially fixed some specific roads as blocked. First, all the firefighters were at the same location, that is, the coordination center. By running the simulation we obtained three different behaviours for the three firefighters:

1. one firefighter reached the assigned destination without any problem;
2. one firefighter reached the destination after having tried more than one alternative path;
3. the third firefighter couldn't reach the assigned destination and, as a result, was redirected to a second location.

Figure 8 shows a snapshot of the evolution of the simulated "Go To Destination" process. With this preliminary experiments we give an initial qualitative evaluation of the suitability of the protocol language to handle processes involving both agents and digital services. The modelling of interactions in the LCC language allows to structure the coordination task still maintaining some adaptability, this latter being intended as the capability to entail situations which are not known a priori (i.e., the blockage state of the road can change over time ).

## RELATED WORK

In the present work, we show how an interaction oriented approach might be adopted to handle the coordination problems arising when multiple agencies need to collaborate on emergency response management. Our approach envisages a twofold perspective: one is related to the provision of a controlled but chaotic environment in which to study the effectiveness of interaction models in coordinating agents in real time. Here the emphasis is on a Multi-Agent simulation as a testbed for investigating, along with the validity of interaction models, the impact of a P2P infrastructure in emergency response. The idea to apply multi-agent models for such purposes is not new since a number of complex Multi-Agent Systems (MAS) simulators are under development (Murakami, Minami, Kawasoe and Ishida, 2002; Kanno, Morimoto and Furuta, 2006; Massaguer, Balasubramanian, Mehrotra and Venkatasubramanian, 2006; Robocup-Rescue, 2005).

The second perspective, the one distinctive to our work, is concerned with the peculiarity of an agent protocol language (LCC in our case) specifically designed for expressing interactions in a P2P fashion, which we use to provide a mechanism for knowledge coalition formation and Web service composition, following the approach in (Li and Robertson, 2005; Robertson et al, 2007; Walton, 2005). Here the emphasis is on MAS techniques for inter-operability and coordination tasks employed in P2P architecture. The combination of the two perspectives has the potential to handle the dynamic and distributed aspects of emergency situations: in such scenarios, a P2P architecture is always preferred over a centralised client/server one since it allows the involvement of an large numbers of participants interacting in a distributed and decentralised manner. We describe collaboration between these participants through the specification of a message passing behaviour for each service involved.

Related research works are either specifically devised for the emergency management area or focused more on the architectural aspect. In particular CASCOM[1] , WORKPAD[2], EGERIS[3], EUROPCOM[4] , POMPEI[5], POPEYE[6] , WIN[7] are few such projects. For example, in the CASCOM project (Context-Aware Business Application Service Coordination in Mobile Computing Environments) an intelligent agent-based peer-to-peer (IP2P) environment is under development (Helin, Klusch, Lopes, Fernandez, Schumacher, Schuldt, Bergenti and Kinnunen, 2005). Here, the service coordination mechanism relies on Semantic Web technologies, such as OWL-S and WSMO, rather than an explicit lightweight protocols. Also the WORKPAD project, aims at designing and developing an innovative software infrastructure (software, models, services, etc.) for supporting collaborative work of human operators in emergency/disaster scenarios (Mecella, Catarci, Angelaccio, Buttazzi, Krek , Dustdar and Vetere, 2006). A set of front-end peer-to-peer communities providing services to human workers, mainly by adaptively enacting processes on mobile ad-hoc networks, is part of the system developed (de Leoni De Rosa and Mecella, 2006). Each community is lead by a super-peer, the only peer managing workflow composition and coordination in an adaptive manner. In this case, a mechanism based on our approach would allow each peer to execute, and eventually modify, the workflow, thanks to the sharing of the multi-agent protocol. The work in (Li et al, 2005) also represents an effort in this direction: in this case the approach has been applied directly to business modelling. The method has proved to be promising also in the field of e-Science (Walton and Barker, 2004).

## CONCLUSIONS

The purpose of this paper is to show how an approach based on interaction models sharing through a P2P network can be applied successfully to process coordination problems in the context of emergency response management.

Currently, our eResponse system is running on a Prolog based LCC engine. The current system is basic in the

[1] http://www.ist-cascom.org
[2] http://www.workpad-project.eu/description.htm
[3] http://www.egeris.org
[4] http://www.ist-europcom.org
[5] http://www.pompei-eu.com
[6] http://www.ist-popeye.org
[7] http://www.win-eu.org

sense that it simply provides means to execute LCC interaction models. However, to run an interaction model, the peer should know which interaction model it wants to execute and with which peers it will be interacting. This and more, is under development in the European project OpenKnowledge[8]: the project provide a unifying framework based on interaction models that are mobile among peers.. The developing OpenKnowledge platform (Siebes at al, 2007) provides means for us to move to a more dynamic system: peers will be able to use discovery services to search for appropriate interaction models, look for suitable peers, and make use of some other important functionalities, such computing trust, verifying interaction models, etc..
But how does this help our (flood) emergency scenario? Let us consider the interaction model of previous section. A more dynamic version of this scenario would be to have a firefighter coordinator asking firefighters to perform certain actions. However, instead of specifying how the firefighter will get the path between two nodes, we may keep this a general constraint to be solved by the peer. At runtime, the peer may then succeed to satisfy the constraint either by consulting its own knowledge base or by using some interaction model to communicate with others in order to achieve its goal. It could then use some discovery service of the OpenKnowledge system to search for suitable interaction models and peers. One solution could be to ask a route service, another solution could be to consult peers in that vicinity.
For emergency scenarios, this ultra dynamic P2P approach is crucial since it implies that even if parts of the system fails, for some reason or another, peers will still be able to find other methods for achieving their critical goals.

## ACKNOWLEDGMENTS

This work has been supported by the OpenKnowledge project (FP6-027253).
The scenario and flooding use case is based on extensive interviews with involved local authorities and analysis of current local emergency plans. We want to thank here the availability of ing. Mario Perghem Gelmi, Marco Bommassar, dott. Federico Bortolotti and ing. Mirko Gasperotti, geom. Cinelli and dott. Michele Zanolli.
We want to thank David Dupplaw for the development of the emergency GUI. We acknowledge Fausto Giunchiglia and Dave Robertson for comments and advice on the eResponse system architecture.

## REFERENCES

1. Autonomous Province of Trento. “Provincial and municipal emergency protection guidelines. Linee guida per le attività di previsione, prevenzione, protezione e la pianificazione di protezione civile provinciale e comunale”, 2005.
2. de Leoni M., De Rosa F., Mecella M.. “MOBIDIS: A Pervasive Architecture for Emergency Management”. Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 107-112, 2006.
3. Helin, H.; Klusch, M.; Lopes, A.; Fernandez, A.; Schumacher, M.; Schuldt, H.; Bergenti, F.; Kinnunen, A.: Context-aware Business Application Service Co-ordination in Mobile Computing Environments. In: AAMAS05 workshop on Ambient Intelligence - Agents for Ubiquitous Computing, 2005.
4. Kanno T., Morimoto Y., Furuta K.. “A distributed multi-agent simulation system for the assessment of disaster management systems”, International Journal of Risk Assessment and Management 2006 - Vol. 6, No.4/5/6 pp. 528 - 544
5. Li G., Robertson D., and Chen-Burger J.. “A novel approach for enacting distributed business workflow on a peer-to-peer platform”. In Proceedings of the IEEE Conference on E-Business Engineering, Beijing, 2005.
6. Massaguer D., Balasubramanian V., Mehrotra S., Venkatasubramanian N.. “Multi-Agent Simulation of Disaster Response”, ATDM Workshop in AAMAS 2006.
7. Mecella M., Catarci T., Angelaccio M., Buttazzi B., Krek A., Dustdar S., and Vetere G.. “Workpad: an adaptive peer-to-peer software infrastructure for supporting collaborative work of human operators in emergency/disaster scenarios”. In Proceedings of the 2006 International Symposium on Collaborative Technologies and Systems. IEEE Computer Society, 2006.
8. Municipality of Trento. “Civilian emergency plan for the municipality of Trento. Piano di Protezione Civile Comunale contro il Rischio Idrogeologico di Inondazione del Fiume Adige”, 2002.
9. Murakami, Y.; Minami, K.; Kawasoe, T.; Ishida, T. “Multi-agent simulation for crisis management”, Knowledge Media Networking, 2002

[8] OpenKnowledge Project: www.openk.org/

10. Robertson D. “A lightweight coordination calculus for agent systems”. In Declarative Agent Languages and Technologies, pages 183-197, 2004-1.
11. Robertson D. “A lightweight method for coordination of agent oriented web services”. In Proceedings of AAAI Spring Symposium on Semantic Web Services, California, USA, 2004-2.
12. Robertson D., Giunchiglia F., van Harmelen F., Marchese M., Sabou M., Schorlemmer M., Shadbolt N., Siebes R., Sierra C., Walton C., Dasmahapatra S., Dupplaw D., Lewis P., Yatskevich M., Kotoulas S. , Perreau de Pinninck A., and Loizou A., “Open Knowledge Semantic Webs Through Peer-to-Peer Interaction”, http://www.cisa.informatics.ed.ac.uk/OK/manifesto.pdf, 2006.
13. Robertson D., Walton C., Barker A., Besana P., Chen-Burger Y., Hassan F., Lambert D., Li G., McGinnis J., Osman N., Bundy A., McNeil F. van Harmelen F., Sierra C., and Giunchiglia F.. “Models of interaction as a grounding for peer to peer knowledge sharing”. Advances in Web Semantics, Vol. 1, 2007.
14. Robocup-Rescue Simulation Project. http://www.rescuesystem.org/robocuprescue/, 2005
15. Siebes R., Dupplaw D., Kotoulas S., Perreau de Pinninck A., van Harmelen F., Robertson D., “The OpenKnowledge System: An Interaction-Centered Approach to Knowledge Sharing”, R. Meersman and Z. Tari et al. (Eds.): OTM 2007, Part I, LNCS 4803, pp. 381–390, 2007, Springer-Verlag Berlin Heidelberg 2007
16. Vaccari L., Ivanyukovich A. and Marchese M., "A Web Service Approach to Geographical data distribution among public administrations" in 5th IFIP Conference on e-Commerce, e-Business, and e-Government I3E'2005, Poznan, Poland, October 26-28, 2005
17. Walton C. and Barker A.. “An agent-based e-science experiment builder”. In Proceedings of the 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid, Valencia, Spain, Aug 2004.
18. Walton C., “Typed Protocols for Peer-to-Peer Service Composition”. Proceedings of the 2nd International Workshop on Peer to Peer Knowledge Management (P2PKM 2005), San-Diego, USA, July 2005.

## APPENDIX 1

In what follows, we present the technical details of the simulation cycles and the interaction models used by the simulator peers.

**Flood.lcc**
This interaction model is used to let the flood sub-simulator inform the controller of the flood changes, which should occur once every simulation cycle. This is depicted as step 1.(a) in the simulation cycle. The interaction model is specified as follows:

```
a(flood_simulator,FS) ::=
   get_topology(URI) <-- initial_topology_source(URI) <= a(controller,C) then
   a(flood_simulator2(C),FS).

a(flood_simulator2(C),FS) ::=
   flood_info(Changes) => a(controller2,C) <-- flood_changes(Changes) then
   ok <= a(controller3(_,_,_),C) then
   a(flood_simulator2(C),FS).


a(controller,C) ::=
   initial_topology_source(URI) => a(flood_simulator,FS) <--
      get_initial_topology(URI) and flood_simulator(FS) then
   a(controller2,C) <-- time(Time).

a(controller2,C) ::=
   update_flood_changes(Changes) <--
      flood_info(Changes) <= a(flood_simulator2(_),FS) then
   a(controller3(FS,Time,Time2),C) <-- time(Time) then
   a(controller2,C).

a(controller3(Id,Time,Time2),C) ::=
   ok => a(flood_simulator2(_),Id) <-- current_timestep(Time2) and Time2>Time
   or
   a(controller3(Id,Time,Time2),C) <-- sleep(0.1).
```

The interaction model is initiated by the controller, which plays the role ‘controller’. When the simulation starts, the controller should coordinate the topology of the world with the flood sub-simulator by sending it the topology’s URI. It then takes the role ‘controller2’ to receive the flood changes sent out by the flood sub-simulator at every time-step. However, the controller should ensure that changes are received only once every time-step. Therefore, after receiving the flood changes, the controller takes the role ‘controller3’, which loops finitely often, until the controller enters a new time-step. At that point, it sends an ‘ok’ message to the flood simulator, signaling that further flood changes may now be sent. It then goes back to role ‘controller2’ to receive any further flood changes from the flood simulator.
Note that the flood sub-simulator, playing the role ‘flood_simulator’, works in parallel with the controller. After receiving the topology of the world, it takes the role ‘flood_simulator2’. In this role, the flood sub-simulator computes the flood changes and sends this information back to the controller. It then waits for an ‘ok’ message from the controller before it repeats this all over again.

**Coordinate_Action.lcc**
This interaction model is used to let the connected physical peers inform the controller of the physical actions they are performing. As mentioned earlier, peers should inform the controller of all their physical actions since these would result in changes in the physical world. Furthermore, it is the controller that would confirm whether an action is currently possible or not. This is depicted as step 1.(b) of the simulation cycle. The interaction model is specified as follows:

```
a(action_performer(Action,Result),Id) ::=
a(action_performer2(C,SubActions,Result),Id)
<-- controller(C) and breakdown(Action,SubActions).

a(action_performer2(C,SubActions,FResult),Id) ::=
( action(Action) => a(action_coordinator,C)
<-- SubActions=[Action|Tail] then
result(Result) <= a(action_coordinator2(_,_),C) then
( null <-- member(Result,[illegal_action(_),indecipherable(_)]) and
FResult=fail
or
a(action_performer2(C,Tail,FResult),Id)
<-- member(Result,[ok]) and performed_action(Action)  )  )
or
null <-- SubActions=[] and FResult=success.


a(action_coordinator,C) ::=
action(Action) <= a(action_performer2(_,_,_),Id) then
a(action_coordinator2(Action,Id),C) then
a(action_coordinator,C).

a(action_coordinator2(Action,Id),C) ::=
result(Result) => a(action_performer2(_,_,_),Id)
<-- update_action_results(Id,Action,Result)
or
a(action_coordinator2(Action,Id),C) <-- sleep(0.1).
```

The above interaction model is initiated by any physical peer that needs to coordinate the actions it needs to perform with the controller. The physical peer will be playing the LCC role 'action_performer'. It initiates this interaction model knowing that 'Action' needs to be performed and expecting some 'Result' to be returned. It then obtains the Id of the controller it will be communicating with, it breaks down the action into atomic sub-actions, and then takes the role 'action_performer2' to coordinate its sub-actions with the controller C.
Note that the physical peer may coordinate one atomic action at a time. Atomic actions are of three categories: *pick*, *drop*, and *move*. Complex actions will have to be broken into atomic sub-actions before they are coordinated with the controller. For example, the action of having Peer1 transport object O from Node1 to Node2 is broken down into the following sub-actions:

- Peer1 *moving* to Node1
- Peer1 *picking* object O
- Peer1 *moving* to Node2
- Peer1 *dropping* object O

Also note that *move* actions may themselves be complex actions that should be broken down into further atomic sub-actions. The only atomic move actions are the actions that allow peers to move between two nodes which are at most 500m away from each other. We assume a peer may move at most 500m per time-step. This allows the controller to keep the motion of peers realistic. For example, you do not want a peer jumping thousands of kilometers while another jumps a few meters.
After taking the role 'action_performer2', the physical peer now performs a recursion over the set of sub-actions it needs to perform. For each sub-action, it sends a message to the controller informing it of the action it would like to perform. It then receives a reply from the controller. If the reply is an 'ok' message, then it proceeds. However, the reply could also be 'illegal_action(_)' or 'indecipherable(_)' messages. If such messages are received, then this implies that either the action is not permitted, or the action is not understood by the controller. In either case, the peer cannot perform that action and it quits the coordination of all remaining sub-actions. The result returned in that case is 'fail'. A 'success' result is returned only if all sub-actions have been completed successfully.

The controller, playing the role 'action_coordinator', works in parallel with the 'action_performer'. After receiving an action, it takes the role 'action_coordinator2', which computes the result of performing such an action and informs the 'action_performer' whether the action may be carried out or not. Due to synchronization

issues, there are strict restrictions by the controller on when actions may be performed. For example, one sub-action may be performed in one time-step. If the controller tries to mark more than one action per one physical peer in one time-step, then the constraint 'update_action_results(_,_,_)' will fail, forcing the controller to wait for 0.1 seconds, before it tries again.

**Visualiser.lcc**
This interaction model is used to let the controller inform the visualiser of all the changes that have occurred in the world at every time-step. This is depicted as step 2.(a) in the simulation cycle. The interaction model is specified as follows:

```
a(get_visual_changes,V) ::=
    get_topology(URI) and start_gui <--
       initial_topology_source(URI) <= a(sending_visual_info,_) then
    a(get_visual_changes2,V).

a(get_visual_changes2,V) ::=
    update(Time,AllChanges) <--
       vis_info(Time,AllChanges) <= a(sending_visual_info2(_),_) then
    a(get_visual_changes2,V).


a(sending_visual_info,C) ::=
    initial_topology_source(URI) => a(get_visual_changes,V)
       <-- get_initial_topology(URI) and visualiser_id(V) then
    a(sending_visual_info2(V),C).

a(sending_visual_info2(V),C) ::=
    vis_info(Time,AllChanges) => a(get_visual_changes2,V)
        <-- get_visual_changes(AllChanges) and time(Time)
    or
    a(sending_visual_info2(V),C) <-- sleep(0.1).
```

The interaction model is initiated by the controller, which plays the role 'sending_visual_info'. When the simulation starts, the controller should coordinate the topology of the world with the visualiser by sending it the topology's URI. It then takes the role 'sending_visual_info2' to recursively send out the changes that occurred in the world at every time-step. If the controller tries to obtain the changes twice for the same time-step, then the constraint '`get_visual_changes(AllChanges)`' will fail, forcing the controller to wait for 0.1 seconds, before it tries again. This ensures changes are sent out only once every time-step.
Note that the visualiser, playing the role 'get_visual_changes', works in parallel with the controller. After receiving the topology of the world, the visualiser starts its GUI. It then takes the role 'get_visual_changes2', where it receives the changes from the controller at every time-step and updates its history accordingly.

**Connect.lcc**
This interaction model is used to let physical peers connect to the simulator. When connected, they can then receive sensory information about changes in their vicinity, once every time-step. This is depicted as step 2.(b) of the simulation cycle. The interaction model is specified as follows:

The interaction model is initiated by the physical peer that wishes to connect to the controller. The peer plays the role 'connecting_peer'. If the peer is interested in connecting to the controller, it would obtain its current peer type (such as firefighter, water sensor, etc.), its location, and the controller's Id. It then sends an 'exists' message to the controller informing it of its peer type and location. After that, it receives a 'connected' message from the controller confirming its connection, along with the time-step at which the peer has joined. The peer then plays the role 'connected_peer', in which it continuously receives sensory information from the controller about objects in its vicinity. Note that at any point, the peer is allowed to exit the simulation by simply sending the 'exit' message to the controller. If this happens, then this would break the recursive cycle through which sensory info is received.

Note that the controller, playing the role 'registrar', works in parallel with the physical peer. After receiving an 'exists' message, it adds the peer to the map and sends back a 'connected' message to confirm the connection. It

then plays the role ‘registrar2’, in which it continuously listens for ‘exit’ messages. If an ‘exit’ message is received, then the peer is removed from the map. Otherwise, it plays the role ‘registrar3’ for sending sensory information to the peer about objects in its vicinity. If the controller tries to obtain this sensory information twice for the same time-step, then the constraint ‘send_update_info(Id,Peers,Flood,Danger,Time)’ will fail, forcing the controller to wait for 0.1 seconds, before it tries again. This ensures that sensory information is sent out only once every time-step.

```
a(connecting_peer,Id) ::=
    exists(Id,PeerType,Location) => a(registrar,C)
        <-- connect(true, PeerType, Location) and controller(C) then
    learn(time(TS)) <-- connected(TS) <= a(registrar,C) then
    a(connected_peer(C), Id).

a(connected_peer(C),Id) ::=
    exit(Id) => a(registrar2(_),C) <-- connect(false,_,_)
    or
    ( update_info(Info) <-- sensory_info(Info) <= a(registrar3(_),C) then
      a(connected_peer(C),Id) ).


a(registrar,C) ::=
    add_peer_to_sim(Id,PeerType,Location,Time) <--
       exists(Id,PeerType,Location) <= a(connecting_peer,Id) then
    connected(Time) => a(connecting_peer,Id) then
    a(registrar2(Id),C).

a(registrar2(Id),C) ::=
    remove_peer_from_sim(Id) <-- exit(Id) <= a(connected_peer(_),Id)
    or
    ( a(registrar3(Id),C) then
      a(registrar2(Id),C) ).

a(registrar3(Id),C) ::=
    sensory_info([Id,Peers,Flood,Danger,Time]) => a(connected_peer(_),Id)
        <-- send_update_info(Id,Peers,Flood,Danger,Time)
    or
    a(registrar3(Id),C) <-- sleep(0.1).
```

## APPENDIX 2

In what follows, we present the technical details of the interaction models used by the simulator peers in the selected use case, i.e .the evacuation plan. The roles, the supported activities, related messages and the related constrains are detailed in the comments within the code.

**StartFE.lcc**

This interaction model is used describes how the evacuation plan starts: (1) an emergency coordinator alerts its personnel to go to a specific destination; (2) each personnel member has to find the route to reach the destination and eventually notify the coordinator in case of failure. It has been tested:

```
// The firefighter coordinator retrieves a list of available firefighters, FFL, in
// order to send an alert message to each of them.

a(firefighter_coordinator,FFC) ::=
   a(firefighter_coordinator(FFL),FFC) <-- obtainFFList(FFL)
   or
   null.

// The role involves a recursion over the list FFL so that a destination meeting
// point (MP) can be assigned and an alert message can be sent to each firefighter
// FFL_H, in FFL. After having sent the messages to all the firefighters, the FFC
// assumes the role firefighter_coordinator3.

a(firefighter_coordinator(FFL),FFC)::=
 (( alert(MP) => a(firefighter,FFL_H) <-- FFL=[FFL_H|FFL_T] and assign_mp(MP,FFL_H)
 then
   a(firefighter_coordinator(FFL_T),FFC) )
   or
   ( null <-- FFL=[] ) ) then
   a(firefighter_coordinator3,FFC).

// In this role the FFC receives a confirmation (or disconfirmation) of reached
// (not reached) destination MP from a given firefighter Id ('action_performer2'
// and 'replanner' are roles assumed by the firefighter role) and the role
// recurses. If a disconfirmation is received, the FFC assigns to the firefighter
// Id a new destination - NewMP - which is then embedded in the alert message sent
// back.

a(firefighter_coordinator3,FFC)::=
   ( (confirm(Id,MP) <= a(action_performer2(_,_,_,_),Id)) then
       a(firefighter_coordinator3,FFC) )
       or
     (disconfirm(Id,MP) <= a(replanner(_,_,_),Id)  then
      (alert(NewMP) => a(firefighter,Id) <-- assign_newDest(MP,NewMP,Id)) then
       a(firefighter_coordinator3,FFC) ).

// The firefighter, FF, receives from the firefighter coordinator, FFC, an alert
// message stating that the destination MP has to be reached. Once the message is
// received, the firefighter, FF, assumes the role of route_finder.

a(firefighter,FF)::=
   ( alert(MP) <= a(firefighter_coordinator(_),FFC)
   or
   alert(MP) <= a(firefighter_coordinator3,FFC) )  then
    a(route_finder(MP,FFC),FF) .

// If the firefighter, FF, is not yet at the location Dest that must be reached, a
// vehicle will be needed together with the name of a digital service able to
// provide a route. A message for requesting a route between two locations (Pos and
// Dest) is then sent to the route service RS. Once the firefighter FF receives the
// requested path together with its sub-paths through a message sent by the route
// service RS, he/her stores the path (together with the subpaths) in its local
// memory and assumes the role action_performer in order to communicate the move
// action to the simulator which //in turn will check its feasibility
```

```
a(route_finder(Dest,FFC),Id) ::=
   ( request_route(Pos,Dest,Vehicle) => a(route_service,RS)
     <-- at(Pos) and not(Pos=Dest) and set_vehicle(Vehicle) and
          get_route_serviceID(RS) ) then
   (route(From,To,Path,SubPaths) <= a(route_service,RS) then
     a(action_performer(move(From,To,Path,Vehicle),FFC),Id)
       <--store(Path,SubPaths,Vehicle)).

// This role is entered by the firefighter when the simulator sends an
// illegal_action as a result. It is similar to the previous role except for the
// fact that
// - a list of banned subpaths - BanSubPaths - is sent to the route service
// - a disconfirmation is sent to the coordinator if no free paths leads to the
// destination.
// Note: in the future this role will be embedded in the "route_finder" role.

a(replanner(S,OldDetails,FFC),Id) ::=
   ( request_route(Location,To,V,BanSubPaths) => a(route_service,RS)
      <-- at(Location) and get_final_location(OldDetails,To) and
          get_vehicle(OldDetails,V) and get_BanSubPaths(BanSubPaths) and
          get_route_serviceID(RS) ) then
     (route(From,To,Path,NewDetails) <= a(route_service,RS) then
       ( a(action_performer(move(From,To,Path,V),FFC),Id)
        <-- Path = [H|T] and store(Path,NewDetails,V) )
            or
        ( (disconfirm(Id,To) => a(firefighter_coordinator3,FFC) <-- Path=[]) then
        a(firefighter,Id) )).

// The route service, RS, after having received a route request from a route finder
// (or replanner) Id, selects a path from node From to node To, split the path in
// sub-paths and sends them back to the route finder Id. The route service role
// then recurses to be able to accept other requests. The route service can receive
// two types of messages from a route finder:
// - A message asking for a path from A to B (a vehicle is also specified);
// - A message asking for a path from A to B (a vehicle is also specified) not
// including certain sub-paths (i.e., the blocked subpaths).

a(route_service,RS)::=
   ( request_route(From,To,Vehicle) <= a(route_finder(_,_),Id) then
   (route(From,To,Path,SubPaths) => a(route_finder(_,_),Id)
      <-- get_route(From,To,Vehicle,Path) and get_subpaths(Vehicle,Path,SubPaths))
   then a(route_service,RS))
   or
   ( request_route(From,To,Vehicle,BanSubPaths) <= a(replanner(_,_,_),Id) then
   (route(From,To,Path,SubPaths) => a(replanner(_,_,_),Id)
      <-- get_route(From,To,Vehicle,BanSubPaths,Path) and
          get_subpaths(Vehicle,Path,SubPaths))
   then a(route_service,RS)).

// The following code is needed when a peer send an action to the simulator's
// controller. Note that this is an adaptation of the interaction model
// ”Coordinate_Action.lcc”. In the future there won’t be the need for this
// adaptation and the “Coordinate_Action.lcc” should remain unmodified and valid
// for every protocol containing interactions with the simulator.

a(action_performer(Action,FFC),Id) ::=
   a(action_performer2(S,Details,Details,FFC),Id)
     <--  simulator(S) and breakdown(Action,Details).

a(action_performer2(S,OldDetails,Details,FFC),Id) ::=
   ( Action => a(action_coordinator,S) <-- Details=[Action|Tail] then
     Result <= a(action_coordinator,S) then
         ( a(replanner(S,OldDetails,FFC),Id)
              <-- member(Result,[illegal_action(_),indecipherable(_)]) and
                  update_BanSubPath(Action)
           or
           a(action_performer2(S,OldDetails,Tail,FFC),Id)
              <-- member(Result,[ok,pick,drop]) and performed_action(Action)))
       or
```

```
     or
       (action_completed => a(action_coordinator,S) <-- Details=[] then
        (confirm(Id,To) => a(firefighter_coordinator3,FFC)
              <-- get_final_location(OldDetails,To) and update_position(To))).


a(action_coordinator,S) ::=
   (action_completed <= a(action_performer2(_,_,_,_),Id))
   or
   ( update_action_results(Id,Action,Result)
             <-- Action <= a(action_performer2(_,_,_,_),Id) then
         Result => a(action_performer2(_,_,_,_),Id) then
         a(action_coordinator,S) ).
```

**Reconnaissance.lcc**
This interaction model describes the phase during which an emergency peer (firefighter) needs to close a meeting point due to the increasing level of water measured at that meeting point and its neighbourhood. It has not been tested yet. The interaction model is specified as follows:

```
// The firefighter first measures the water level (WL_local) at the meeting point
// (MP) and then ask its coordinator to give details on the water level at the
// closest areas (WL_adj_nodes). Based on this info, a parameter (Criticality) is
// computed in order to decide if it is the case to close the meeting point or not.
// If the MP is closed a message is sent to the coordinator.

a(firefighter,FF)::=
 update&request_WL_info(MP, WL_local) => a(firefighter_coordinator,FFC)
    <-- at(MP) and measure_local_WL(MP, WL_local) and get_coordinator_ID(FFC) then
    emergency_WL_info(MP, WL_adj_nodes) <= a(firefighter_coordinator,FFC) then
      null <- compute(WL_local, WL_adj_nodes, Criticality) then
       (a(perfom(close(MP)),FF) <- greater(Criticality,MaxCriticality) then
       update_on_MP_state(closed) => a(firefighter_coordinator,FFC))
       or
       a(firefighter,FF).

// The firefighter coordinator receives a message from the firefighter to update
// info on water level at meeting point MP and to send info on water levels at its
// adjacent points. The coordinator can also receives an
// update_on_MP_state(closed)" message from the firefighter.

a(firefighter_coordinator,FFC)::=
 ((update&request_WL_info(MP, WL_local) <= a(firefighter,FF) then
   emergency_WL_info(MP, WL_adj_nodes) => a(firefighter,FF)
    <-- update_WL_info(MP,WL_local) and gather_WL_adj_area(MP, WL_adj_nodes))
   or
   update_on_MP_state(closed) <= a(firefighter,FF) ) then
    a(firefighter_coordinator,FFC).
```

**Census.lcc**
This interaction model is used when an emergency peer registers info on people present at a certain location into a database. It has not been tested yet. The interaction model is specified as follows:

```
// The emergency peer enters this role to obtain the info (Name Type, Location, //
// etc.) related to a person and to send a relative message to the dbcurator (the
// peer who actually will store the info into the DB). A success (failure) message
// is received if things went well (wrong)

a(censuscurator, ID1)::=
update(Name,Type,Location,GPS,Status,Notes) => a(dbcurator, ID2)
     <-- obtain(List, Name,Type,Location,GPS,Status,Notes)  then
     success(Id) <= a(dbcurator, ID2)
     or
    (failure(Msg) <= a(dbcurator, ID2) then
     null <- proceed(Msg))

// The peer who stores info into a DB enters this role.

a(dbcurator, ID2)::=
   update(Name,Type,Location,GPS,Status,Notes) <=  a(censuscurator, ID1) then
     success(Id) => a(censuscurator, ID1)
      <-- updateDb(Name,Type,Location,GPS,Status,Notes,Id)
     or
     failure(Msg) => a(censuscurator, ID1) <- lastError(Msg)
```

**Bushandling.lcc**

This interaction model is used when an emergency peer (firefighter) in need to evacuate people present at a meeting point, requests a certain number of buses and, while they arrive, sends them to specific destinations. It has not been tested yet. The interaction model is specified as follows:

```
// After having counted the number of people at the meeting point (MP) and
// evaluated the criticality (danger level at that MP) the firefighter request a
// certain number of bus (No_bus) the firefighter will then receive from its
// coordinator the quantity of buses (NB) that will arrive and a list of refuge
// centers (DCL) where the buses can go after people are loaded.

a(firefighter(MP),FF)::=
 request_bus(MP,No_bus) => a(firefighter_coordinator,FFC)
   <-- count(People) and get(Criticality) and bus_needed(People,Criticality,No_bus)
       and get_coordinator_ID(FFC) then
   confirm_bus_arrival(NB,DCL) <= a(firefighter_coordinator,FFC) then
      a(wait_for_bus(NB,DCL),FF) <-- set_destination(DCL) then
       a(firefighter(MP),FF).

// The coordinator receives the request of buses, decides how many buses will send
// (No_bus) and assigns to each of them a destination. Then, he/she will send a
// confirmation message to the bus requester

a(firefighter_coordinator,FFC)::=
 request_bus(MP,Quantity) <= a(firefighter(_),FF) then
   confirm_bus_arrival(No_bus,Destinations) =>  a(firefighter(_),FF)
    <-- check_for_bus(Quantity,No_bus) and assign(No_bus,Destinations).

// The bus requester enter this role after having received a confirmation of bus
// arrival from the coordinator. He/She expects buses arriving in a quantity of
// No_expected and has a list of destinations where to send each of them.
// Once a certain number (No_Bus) of buses arrives at the location (MP), a list
// (ABL) containing details for each bus is given. The list could merely be a list
// of bus identifiers. The firefighter will assign to the arrived buses a certain
// number of destinations (Next_DCL) taken from the list DCL and will start the
// buses (give a directive to the bus drivers ). He will then, wait for the other
// remaining buses.

a(wait_for_bus(No_expected,DCL),FF)::=
   ( a(start_bus(ABL,Next_DCL),FF)
      <-- DCL = [DCL_H| DCL_T] and bus_at_location(MP, No_Bus, ABL) and
          assign_dest(No_Bus,DCL,Next_DCL,Rest_DCL) then
      a(wait_for_bus(No_expected - No_Bus,Rest_DCL),FF) )
   or
   null <- empty(DCL).

// This role is needed when the firefighter has to start the bus. To each bus is
// assigned a destination (ABL is equal to DCL in lenght) and once the driver of
// each bus is identified, a directive to him/her is sent

a(start_bus(ABL,DCL),FF)::=
    (drive_to(DCL_H) => a(bus_driver,BD)
      <-- ABL=[ABL_H|ABL_T] and DCL=[DCL_H|DCL_T] and driver_of(ABL_H,BD) then
      a(start_bus(ABL_T,DCL_T),FF))
     or
     null <- empty(ABL).

// The bus driver, once the directive is received, get the route and move towards
// the destination. (here the bus_driver could jump to a "route_finder" role as
// defined in the interaction model "startFE.lcc")

a(bus_driver,BD)::=
 drive_to(Destination) <= a(start_bus(_,_),FF) then
  a(perfom(move(Location, Destination, Path, bus)), BD)
    <-- at(Location) and get_route(Location,Destination,Path).
```

# An Application of Approximate Ontology Matching in eResponse

**Maurizio Marchese**
University of Trento
maurizio.marchese@unitn.it

**Lorenzino Vaccari**
University of Trento
vaccari@disi.unitn.it

**Pavel Shvaiko**
University of Trento
pavel@disi.unitn.it

**Juan Pane**
University of Trento
pane@disi.unitn.it

## ABSTRACT

Ontology matching is a key problem in many metadata intensive application domains, including emergency response, data integration, peer-to-peer information sharing, web service composition, and query answering on the web. In this paper we present an emergency response scenario based on the organizational model as used in Trentino, Italy. We provide a formalization of this scenario with the help of lightweight coordination calculus. Then, we discuss an automatic approximate structure preserving matching algorithm which we applied within the emergency response scenario. The evaluation results, though preliminary, are encouraging.

### Keywords

Peer-to-peer networks, web services, semantic heterogeneity, ontology matching, knowledge sharing in crisis management, interaction modeling.

## INTRODUCTION

The need to harness the potential of electronic networks in emergency situations is widely recognized as a relevant research priority. *In times of crisis - be it a natural disaster, terrorist attack or infrastructure failure - mobile personnel need to work together in time-critical and dangerous situations. Real-time access to information and knowledge will help save lives. Crises are complex situations, with large numbers and varieties of mobile personnel - medical and rescue teams, police, fire fighters and other security personnel - appearing on the spot at short notice. These different teams come from different organizations, and generally have incomplete or even contradictory knowledge of the crisis situation*[9]. The quoted text provides some examples of the key elements needed in emergency response situations. In particular, emergency management activities – that in the following we will reference as emergency response (eResponse) activities – are developed and implemented through the essential analysis of information and the coordination of the involved peers, including emergency personnel, army, volunteers, etc. Disaster data and events can be acquired, modelled, fused and displayed in state-of-the-art Spatial Data Infrastructures (SDIs) using distributed data sources, the majority of which are spatial. Moreover, emergency monitoring and management activities normally involve a range of different organizations and teams at various administrative levels with their own systems and services. The application of numerous and different actors, policies, procedures, data standards and systems, results in coordination problems with respect to data analysis, information delivery and resource management, all critical elements of emergency response management.

Current technologies and information systems can provide parts of the solution with ad-hoc and mostly centralized systems. In particular, institutional agencies (e.g., municipal, regional) have started to adopt distributed SDIs (Groot and McLaughlin, 2000, Nebert, 2004, Bernard et al., 2005). While Geographical Information Systems (GIS) are self-contained systems in which data and software applications are used mainly internally, the goal of SDI is to support the *interoperability* among different kinds of institutions, users and roles. However, SDIs are pervaded by *interoperability problems,* including: *(i)* geo-data interoperability, specifically, geographical datasets have particular properties (e.g., maps as implicit interfaces) to be tackled, which are different from other types of data (Parent et al., 2006) and *(ii)* geo-service interoperability issues, such as geo-service discovery, integration and composition.

In this paper we propose to use a peer-to-peer (P2P) infrastructure for the eResponse domain. At the core of our approach is a specific view on semantics of both web service and agent coordination as proposed in (Robertson

[9] EU "Emergency Response Grid" programme: http://cordis.europa.eu/ist/grids/emergencey_response_grid.htm, 2006.

et al., 2007). Peers share explicit knowledge of the "interactions" in which they are engaged and these models of interaction are used operationally as the anchor for describing the semantics of the interaction. Instead of requiring a universal semantics across peers we require only that semantics is consistent (separately) for each instance of an interaction. These models of interactions are developed locally by peers. However, they must be shared and interpreted by peers in order to support interaction coordination. In our approach, we achieve this by dynamically matching terms in the interaction models. This can happen both at design time (i.e., when synthesising different interactions models) and at execution time (i.e., when running them to perform specific tasks).

The main contributions of the paper include: *(i)* a P2P formalization of the eResponse scenario based on the emergency organizational model as used in Trentino, Italy, *(ii)* an application of the approximate semantic matching algorithm, originally proposed in (Giunchiglia et al., 2007b), within the eResponse domain.

The structure of the rest of the paper is as follows. We introduce our emergency response scenario and its formalization in Section 2 and Section 3, respectively. In Section 4 we present the use of an approximate structure preserving semantic matching algorithm in order to reduce the semantic heterogeneity problem as required by the scenario. Preliminary evaluation of the approach used is provided in Section 5. The related work is discussed in Section 6. Finally, the major findings of the paper are summarized in Section 7.

## SCENARIO

We have analyzed the organizational model of the distributed GIS Agency infrastructure of Trentino, Italy. The framework of the distributed system is represented by a number of specialized GIS agencies: civilian protection, urban planning, forestry, viability, etc. Each GIS agency is responsible for a subset of the geographic information for the local region. To support interoperability among the different GIS agencies the regional information infrastructure is shifting from a traditional GIS system to a SDI.

We focus on the details and roles of the GIS Agency (GA). The GIS Agency is responsible to provide geographic datasets and services to external service requestors. The main generic actors in the current organization model of the GA Agency in Trentino, together with a short description of their main roles, are the following:

- `GA_SR`: (GIS Agency Service Requestor). Main role: ask for service (maps, datasets, analysis, etc.).
- `GA_SP`: (GIS Agency Service Provider). Main role: interface from external actors and internal GA actors. Service aggregation (design time / run time).
- `GA_MAP`: (GIS Agency Map Provider). Main role: building geographical maps from a number of available sources of geo-data (GeoDBs).
- `GA_MEP`: (GIS Agency Metadata Provider). Main role: GIS metadata provider (formalized description, search, matching, etc.) from a number of available sources of geo-data (GeoDBs).

Figure 1 shows a simplified organizational model of the GIS Agency together with the main interactions (arrows).

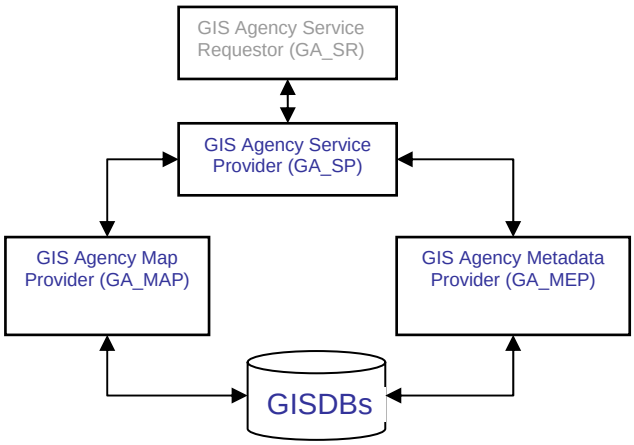


**Figure 1.** The Trentino GIS Agency organizational model.

Within the general Trentino SDI management scenario, we focus on the most commonly used specific use case,

i.e., Map Request Service. Let us discus one particular – but the most typical – request that can be made by the service requestor agent (GA_SR in Figure 1): a digital map request. A service requestor – both in an emergency or normal situation – needs to visualize a map of a region with geo-referenced information selected by a user. Therefore, the searched map is a composition of different geographic layers offered by one of the service provider agents (GA_SP in Figure 1).
A simplified interaction for the generic Map Request Service is illustrated in Figure 2. It uses a standard OGC (Open Geospatial Consortium) WMS (Web Map Service)[10] request and models a subset of the role interactions from an external requestor to the service provider for the request of a map service:

- The requestor assumes the GA_SR role and it asks to the service provider (GA_SP) the list and the characteristics of the services (`Capabilities`) provided by the service provider.
- The service provider provides its capabilities, in particular: the list of available services (`AvailableServices`), the list of geographic datasets managed by the server (`AvailableLayers`), the file format of the return services (`Format`), and the geographic limits of the available services (`XMin_ME,YMin_ME,XMax_ME,YMax_ME`).
- Then, the GA_SR asks for the map service using the information received from the previous step. The `RequestMap` message contains the URL of the requested map (`MapFile`), the version of the service (`Version`), the requested geographic layers (`Layers`), the dimension of the map (`Width, Heigth`), the graphic format of the map (`Format`), the spatial coverage of the map (`XMin_BB, YMin_BB, Xmax_BB, YMax_BB`).
- The service provider provides the map (`Map`) requested by the requestor.
- Finally, the service requestor asks for the graphic legend that describes the previous map and the service provider sends the legend (`Legend`) to the service requestor.

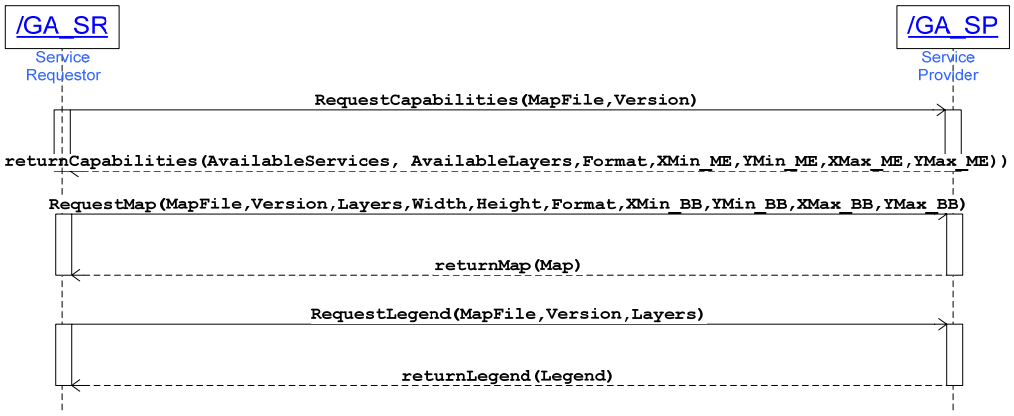


**Figure 2.** Sequence diagram for the map request service.

## INTERACTION MODELS

In this section, we first describe the Lightweight Coordination Calculus (LCC), that is, the communication language employed to implement the interactions among peers acting in our emergency network. We then focus on the selected composition problem depicted in the previous section and provide its formalization with LCC.
LCC is a protocol language used to describe interactions among distributed processes, e.g., agents, web services. LCC was designed specifically for expressing P2P style interactions within multi-agent systems, i.e., without any central control; henceforth, it is well suited for modeling coordination of software components running in an open environment. Its main characteristics are the flexibility, the modularity and the neutrality to the distributed communication infrastructure (Robertson, 2004).
Interactions in LCC are expressed as the message passing behaviors associated with roles. The most basic behaviors are to send or receive messages, where sending a message may be conditional on satisfying a constraint (pre-condition) and receiving a message may imply constraints (post-condition) on the agent accepting it. A basic LCC interaction is shown in Figure 3.

[10] Open Geospatial Consortium web site http://www.opengeospatial.org

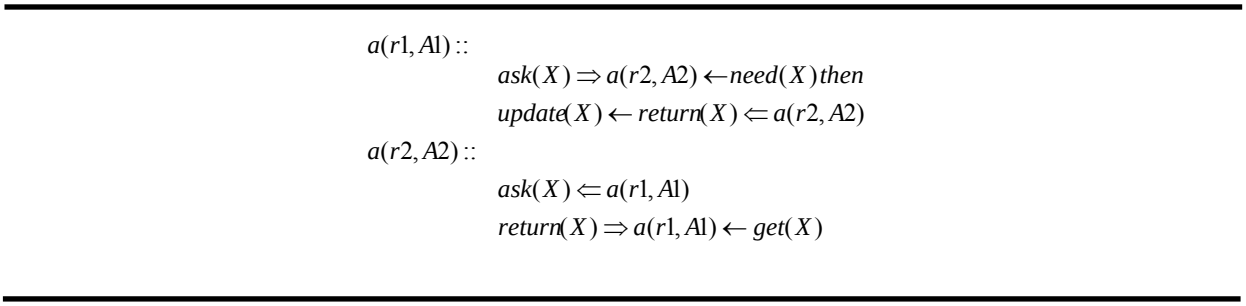


**Figure 3.** LCC fragment; double arrows indicate message passing; single arrows indicate constraint satisfaction.

In Figure 3, the agent `A1` playing the role `r1` verifies if it needs the info `X` (pre-condition `need(X)`); if yes, `A1` asks for `X` to the agent `A2` playing the role `r2` by sending the message `ask(X)`. `A2` receives the message, `ask(X)` from `A1` and then obtain the info `X` (pre-condition `get(X)`) before sending back a reply to `A1` through the message `return(X)`. After having received the message `return(X)`, `A1` updates its knowledge (post-condition `update(X)`).
The constraints embedded into the protocol express its semantics and could be written as first-order logic predicates (e.g., in Prolog) as well as methods in an object-oriented language (e.g., in Java). The characteristic of modularity allows separating the protocol from the agent engineering. While performing the protocol, peers can therefore exchange messages, satisfy constraints before (after) messages are sent (received) and jump from one role to another so that a flexible interaction mechanism is enabled still following a structured policy, which is absolutely necessary for team-execution of coordinated tasks.

Figure 4 and Figure 5 show respectively the LCC code related to `GA_SR` and to `GA_SP` roles used in Figure 2.

- *GIS agency service requestor* (see Figure 4). The GIS agency service requestor `R` initiates the coordination task by entering the `GA_SR` role. According to this role the agent `R` assumes the role `ga_sr` and asks for the capabilities of the services provider. It uses the service address (`MapFile`) and the software version of the client request (`Version`). After that, the service requestor waits until the service provider returns the list of its capabilities. In our case it waits for the list of the available services (`AvailableServices)`, the list of the available layers (`AvailableLayers`), the format of the returned file (`Format`), and the geographic coverage (map extent) of the available services (`XMin_ME, YMin_ME, Xmax_ME, YMax_ME`). In the second step the agent `R` asks to the service provider for a map. It selects some of the available geographic layers (`selectLayers(AvailableLayers,Layers)`), defines the map dimension (`needMap(Width,Height)`) and selects an area from the available geographic extension (`selectBoundingBox(XMin_ME, YMin_ME, XMax_ME, YMax_ME, XMin_BB, YMin_BB, XMax_BB, YMax_BB)`). The third operation is the request of the map legend for the selected layers.
- 
- *GIS agency service provider* (see Figure 5). The GIS agency service provider `P` assumes the role `ga_sp` and waits for one of the following requests: `requestCapabilities`, `requestMap` and `requestLegend`. In the former case, after receiving the request from the service requestor, it builds its capabilities (`getCapabilities(MapFile, Version, AvailableServices, AvailableLayers, Format, Xmin_ME, YMin_ME, Xmax_Me, YMax_ME)`) and passes them to the requestor. In the second request the service provider builds a digital map (`getMap` constraint) and passes it to the service requestor. In the third case the service provider agent provides a legend for a collection of requested layers.

In the following section we will use the `getMap` constraint (highlighted in a box in Figure 5 to facilitate the presentation) as the motivating example to the structure preserving semantic matching approach.

$a(ga\_sr, R)$ ::

$requestCapabilities(MapFile, Version) \Rightarrow a(ga\_sp, P)$
$\leftarrow needCapabilities(MapFile, Version)$ *then*

$returnCapabilities(AvailableServices, AvailableLayers, Format,$
$XMin\_ME, YMin\_ME, XMax\_ME, YMax\_ME)$
$\Leftarrow a(ga\_sp, P)$ *then*

$requestMap(MapFile, Version, Layers, Width, Height, Format,$
$XMin\_BB, YMin\_BB, XMax\_BB, YMax\_BB) \Rightarrow a(ga\_sp, P)$
$\leftarrow selectLayers(AvailableLayers, Layers) \wedge needMap(Width, Height)$
$selectBoundingBox(XMin\_ME, YMin\_ME, XMax\_ME, YMax\_ME,$
$XMin\_BB, YMin\_BB, XMax\_BB, YMax\_BB)$

$returnMap(Map) \Leftarrow a(ga\_sp, P)$ *then*

$requestLegend(MapFile, Version, Layers) \Rightarrow a(ga\_sp, P) \leftarrow needLegend$ *then*

$returnLegend(Legend) \Leftarrow a(ga\_sp, P)$

**Figure 4.** LCC fragment for the GIS agency service requestor role.

$a(ga\_sp, P)$ ::

$requestCapabilities(MapFile, Version) \Leftarrow a(ga\_sr, R)$ *then*
$returnCapabilities(AvailableServices, AvailableLayers, Format,$
$XMin\_ME, YMin\_ME, XMax\_ME, YMax\_ME$
$\Rightarrow a(ga\_sr, R)$
$\leftarrow getCapabilities(MapFile, Version, AvailableServices, AvailableLayers,$
$Format, XMin\_ME, YMin\_ME, XMax\_ME, YMax\_ME)$

*or*

$requestMap(MapFile, Version, Layers, Width, Height, Format,$
$XMin\_BB, YMin\_BB, X\max\_BB, YMax\_BB)$
$\Leftarrow a(ga\_sr, R)$ *then*
$returnMap(Map) \Rightarrow a(ga\_sr, R)$
$\leftarrow getMap(MapFile, Version, Layers, Width, Height, Format,$
$XMin\_BB, YMin\_BB, XMax\_BB, YMax\_BB, Map)$

*or*

$requestLegend(MapFile, Version, Layers) \Leftarrow a(ga\_sr, R)$ *then*
$returnLegend(Legend) \Rightarrow a(ga\_sr, R)$
$\leftarrow getLegend(MapFile, Version, Layers, Legend)$

**Figure 5.** LCC fragment for the GIS agency service provider role.

## APPROXIMATE STRUCTURE PRESERVING SEMANTIC MATCHING

An ontology typically provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary. Depending on the precision of this specification, the notion of ontology includes various data and conceptual models (Euzenat and Shvaiko, 2007). The term ontology is used here in a wide sense, and, hence, encompasses sets of terms, classifications, database schemas, thesauri, etc.

Ontology matching is a plausible solution to the semantic heterogeneity problem faced by information management systems. We view ontologies as graph-like structures (Giunchiglia and Shvaiko, 2003). We think of matching as an operation that takes two graph-like structures, such as web service descriptions, and produces a set of correspondences between the nodes of the graphs that correspond semantically to each other (Giunchiglia *et al.*, 2007a). Then, these correspondences can be used for various tasks, including data translation, etc. Thus, matching ontologies enables the knowledge and data expressed in the matched ontologies to interoperate.

In our scenario, since there is no a priori semantic agreement (other than the interaction model), the ontology matching component is needed to automatically make semantic commitments between the interacting parts. It poses additional constraints on conventional ontology matching. Specifically, we need to compute the correspondences holding among the full graph structures and preserve certain structural properties of the graphs under consideration. Thus, the goal here is to have a *structure preserving semantic matching* operation. This operation takes two graph-like structures and produces: (*i*) one-to-one correspondences between semantically related nodes of the structures preserving a set of structural properties of the graphs being matched, namely that functions are matched to functions and variables to variables, (*ii*) only in the case if the graphs globally correspond semantically to each other, e.g., $graph_1$ is *0.65* similar to $graph_2$, according to some measure.

Let us suppose that we want to match the constraint (e.g. a web service description) - `getMap(MapFile, Version, Layers, Width, Height, Format, XMin_BB, YMin_BB, XMax_BB, YMax_BB)` (see also Figure 5) - with a similar but different one - `getMap(Dimension(Width, Height), MapFile, Edition, Layers, DataFormat, Request, Xmin, Ymin, Xmax, Ymax)`. As shown in Figure 6, the first web service description requires the first argument of `getMap` function (`MapFile`) to be matched to the third (`MapFile`) of `getMap` function in the second description. `Version` in the first web service description must be passed to the second web service as the `edition` argument. Moreover, `request` on the right has no corresponding term on the left.

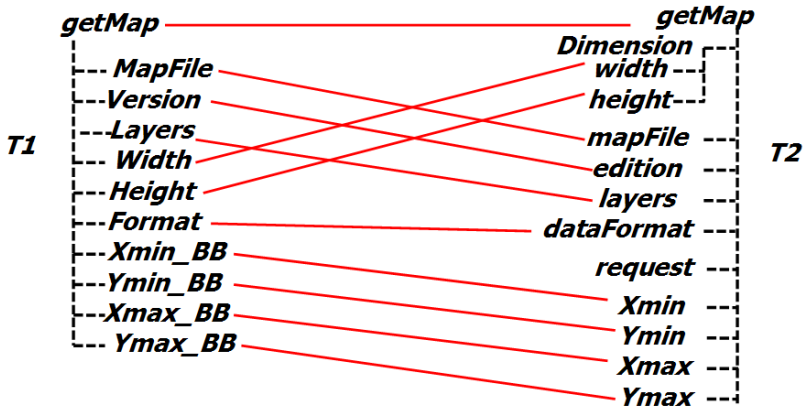


**Figure 6.** Two web service descriptions (trees) and correspondences (lines) between them.

*The approach*

The approach outlined next follows the work in (Giunchiglia *et al.*, 2007b). We briefly report it here for completeness and discuss it with the help of examples from the eResponse domain.

We focus on tree-like-structures. The matching process is organized in two steps: (*i*) node matching and (*ii*) tree matching. Node matching solves the semantic heterogeneity problem by considering only labels at nodes and domain specific contextual information of the trees. We use here the S-Match system (Giunchiglia *et al.*, 2007a). Technically, two nodes *n1* and *n2* in trees *T1* and T2 match if: *c@n1 R c@n2* holds based on S-Match.

*c@n1* and *c@n2* are the concepts at nodes *n1* and *n2*. $R \in \{=, \sqsubseteq, \sqsupseteq\}$. In particular, in semantic matching

(Giunchiglia and Shvaiko, 2003) as implemented in the S-Match system (Giunchiglia *et al.*, 2007a) the key idea is that the relations (e.g., =, ⊑) between nodes are determined by (*i*) expressing the entities of the ontologies as logical formulas and (*ii*) reducing the matching problem to a logical validity problem. Specifically, the entities are translated into logical formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet (Miller, 1995). This allows for a translation of the matching problem into a logical validity problem, which can then be efficiently resolved using sound and complete state of the art satisfiability (SAT) solvers (Giunchiglia *et al.*, 2005). Notice that the result of this stage is the set of correspondences holding between the nodes of the trees.
Tree matching, in turn, exploits the results of the node matching and the structure of the trees to find if these globally match each other. For obvious reasons we are interested in approximate tree matching. Technically, two trees *T1* and *T2* approximately match if there is at least one node $n1_i$ in *T1* and node $n2_j$ in *T2* such that: (*i*) $n1_i$ approximately matches $n2_j$, (*ii*) all ancestors of $n1_i$ are approximately matched to the ancestors of $n2_j$. Notice that the horizontal order of siblings is not preserved being not a desirable property for the data translation purposes.

*The implementation*

The implementation of approximate structure matching is based on (*i*) the theory of abstraction and (*ii*) the tree edit distance. Specifically, the work in (Giunchiglia and Walsh, 1992) categorizes the various kinds of abstraction operations, including:

- *Predicate (Pd)*: Two or more predicates are merged, typically to the least general generalization in the predicate type hierarchy, e.g., *Height(X) + Dimension(X)* $\rightarrow$ *Dimension (X)*. We call *Dimension(X)* a predicate abstraction of *Height(X)*, namely *Dimension(X)* $\sqsupseteq_{Pd}$ *Height(X)*. Conversely, we call *Height(X)* a predicate refinement of *Dimension(X)*, namely *Height(X)* $\sqsubseteq_{Pd}$ *Dimension(X)*.
- *Domain (D)*: Two or more terms are merged, typically by moving constants to the least general generalization in the domain type hierarchy, e.g., *Xmin_BB + Xmin* $\rightarrow$ *Xmin*. We call *Xmin* a domain abstraction of *Xmin_BB*, namely *Xmin* $\sqsupseteq_D$ *Xmin_BB*. Conversely, we call *Xmin_BB* a domain refinement of *Xmin*, namely *Xmin_BB* $\sqsubseteq_D$ *Xmin*.
- *Propositional (P)*: One or more arguments are dropped, e.g., *Layers (L1)* $\rightarrow$ *Layers*. We call *Layers* a propositional abstraction of *Layers(L1)*, namely *Layers* $\sqsupseteq_P$ *Layers(L1)*. Conversely, *Layers(L1)* is a propositional refinement of *Layers*, namely *Layers(L1)* $\sqsubseteq_P$ *Layers*.

Let use consider the following example: (*Height(H))* and (*Dimension*). In this case there is no abstraction/refinement operation that makes those first order terms equivalent. However, consequent applications of propositional and predicate abstraction operations make the two terms equivalent: *Height (X)* $\sqsubseteq_P$ *Height* $\sqsubseteq_{Pd}$ *Dimension*.

Then, the key idea is to use abstractions/refinements as tree edit distance operations in order to estimate the similarity of two tree structures. Tree edit distance is the minimum number of tree edit operations, namely node *insertion*, *deletion*, *replacement*, required to transform one tree to another (Valiente, 2002). We want to: (*i*) minimize the editing cost, i.e., computation of the minimal cost composition of abstractions/refinements, (*ii*) allow only those tree edit operations that have their abstraction theoretic counterparts. Thus, we assign the same unit cost to all operations that have their abstraction theoretic counterparts, while operations not allowed by definition of abstractions/refinements are assigned an infinite cost, see Table 1.

Finally, the global similarity between two trees is computed as follows:

$$TreeSim = 1 - \frac{\sum_{i \in S} n_i \cdot Cost_i}{\max(size\ of\ T1, size\ of\ T2)}$$

where, *S* is the set of allowed tree edit operations, $n_i$ is the number of *i*-th operations necessary to convert one tree into the other, $Cost_i$ is the cost of the *i*-th operation. For the example in Figure 6, *TreeSim* would be 0.58. Then, based on a predefined threshold (e.g., 0.5) it is decided whether the trees under consideration are similar enough, namely if *TreeSim* is lower than the threshold, those trees are considered as not similar, which is not the case in our example. If the similarity score exceeds a given threshold, the correspondences connecting the nodes of the term trees are further used for data translation.

| Abstractions | Operation | Preconditions | $Cost_{=}$ | $Cost_{\sqsubseteq}$ | $Cost_{\sqsupseteq}$ |
|---|---|---|---|---|---|
| $t_1 \sqsupseteq_{Pd} t_2$ | ***replace(a, b)*** | $a \sqsupseteq b$; ***a*** and ***b*** correspond to predicates | 1 | ∞ | 1 |
| $t_1 \sqsupseteq_{D} t_2$ | ***replace(a, b)*** | $a \sqsupseteq b$; ***a*** and ***b*** correspond to functions | 1 | ∞ | 1 |
| $t_1 \sqsupseteq_{P} t_2$ | ***insert(a)*** | ***a*** corresponds to predicate, function | 1 | ∞ | 1 |
| $t_1 \sqsubseteq_{Pd} t_2$ | ***replace(a, b)*** | $a \sqsubseteq b$; ***a*** and ***b*** correspond to predicates | 1 | 1 | ∞ |
| $t_1 \sqsubseteq_{D} t_2$ | ***replace(a, b)*** | $a \sqsubseteq b$; ***a*** and ***b*** correspond to functions | 1 | 1 | ∞ |
| $t_1 \sqsubseteq_{P} t_2$ | ***delete(a)*** | ***a*** corresponds to predicate, function | 1 | 1 | ∞ |

**Table 1.** Tree edit distance operations and their costs.

## PRELIMINARY EVALUATION

The approximate structure preserving matching algorithm has been implemented in Java (Giunchiglia *et al.*, 2007b). As the work in (Avesani *et al.*, 2005, Giunchiglia *et al.*, 2007c) indicates, it takes around one year to build a large scale evaluation dataset. We have already started the process of building such a dataset for the eResponse domain and extensive evaluation constitutes one of the key directions of our future work. Here, we only report our preliminary evaluation results based on the motivating example of Figure 6 and several dozens of similar test cases we have acquired so far. The reference results for these problems were established manually. Then, the results computed by our solution have been compared with the reference results.

As match quality measures we have used *F-measure*. It varies in the [0-1] range. The version computed here is the harmonic mean of *precision* (the measure of correctness) and *recall* (the measure of completeness), namely that each of these was given equal importance (Euzenat and Shvaiko, 2007). While computing F-measure we considered only if the trees match globally. Based on our previous experience in (Giunchiglia *et al.*, 2007a) we used here a cut-off threshold of 0.5. As a performance measure we have used *time*. It estimates how fast our solution when matching trees fully automatically. All these tests have been performed on a standard laptop: Core Duo CPU - 2Hz, with 2 GB of RAM, with the Windows Vista operating system, and with no applications running but a single matching system.

For the example of Figure 6, 11 node-to-node correspondences, namely 7 equivalence and 4 abstraction/refinement relations, were identified by the matching algorithm. These were further aggregated into a similarity score of 0.58, which in turn is higher than the cut-off threshold of 0.5, and, therefore, the two trees globally match as expected. The average F-measure of the matching algorithm on the test cases we ran was 0.7, while the average execution time was 46ms. These results look encouraging, especially for what concerns the

execution time, though further extensive large scale evaluation is needed.

## RELATED WORK

We discuss the related work along two dimensions: *(i)* ontology matching and *(ii)* ontologies in crisis management. Related to *(i)*, many different matching solutions have been proposed so far: see (Noy, 2004, Shvaiko and Euzenat, 2005) for recent surveys, while examples of individual approaches addressing the matching problem can be found on www.OntologyMatching.org. These solutions take advantage of the various properties of ontologies (e.g., labels, structures) and use techniques from different fields (e.g., statistics and data analysis, machine learning, linguistics). These solutions share some techniques and attack similar problems, but differ in the way they combine and exploit their results. A detailed analysis of the different techniques in ontology matching has been given in (Euzenat and Shvaiko, 2007).

The problem of discovery of web services on the basis of their capabilities has recently received a considerable attention. Most of the approaches to the problem of web service matching employ a single ontology approach, i.e., the web services are assumed to be described by the concepts taken from a shared ontology, see, e.g., (Paolucci *et al.*, 2002). The most similar to the solution that we used in our eResponse scenario, i.e. the one in (Giunchiglia *et al.*, 2007b), is the approach taken in (Aggarwal *et al.*, 2004) and in (Oundhakar *et al.*, 2005), where the services are assumed to be annotated with the concepts taken from various ontologies. The matching algorithm combines the results of atomic matchers that roughly correspond to the element level matchers exploited as part of the S-Match algorithm (Giunchiglia *et al.*, 2007a).

Related to *(ii)*, relevant works can be found within the DIP[11], ORCHESTRA[12], WORKPAD[13], SAFE[14], and CASCOM[15] projects. For example, in the DIP project, a GIS based on the web services modelling ontology has been developed (Tanasescu *et al.*, 2006). In WORKPAD, the key idea is that P2P information integration systems do not rely on a single ontology, but use mappings, dynamically established between peers, to collect and merge data from various sources when answering user queries in the crisis scenarios (Mecella *et al.*, 2006). The SAFE project aims at assisting in the management of the response to natural disasters. Specifically, the work in (Iannella *et al.*, 2007) reviews some of the emerging requirements for crisis information management systems and provides an outlook of the current and future technologies that are needed to address these requirements.

All these projects develop at design time domain ontologies for the crisis management. In turn, the matching solution we used can support coordination of web services at run time using both predefined ontologies and the "emerging" ontologies (being the LCC constraints) in order to search and chain web services at run time, which is crucial for many emergency situations.

## CONCLUSIONS

We have presented a typical emergency response scenario, which includes geo-data sharing through geo-services provided by the GIS agencies and its formalization using LCC. The scenario is based on the current organizational model as used in Trentino, Italy. We then discussed an automatic approximate structure preserving matching algorithm used as a solution to the semantic heterogeneity problem between different implementations of the required geo-services. We applied the matching algorithm to the eResponse web service matching scenario. The evaluation results, though preliminary, look encouraging.

Currently, the matching solution is elementary in the sense that it provides means to match web services available in the LCC interaction models. However, to run an interaction model, a peer should know which interaction model it wants to execute and with which peers it will be interacting. The ultimate goal is to provide a unifying framework based on interaction models that are mobile among peers, being a mechanism for web service composition and enabling ad hoc peer coalition formation, as required by hastily formed networks (Denning, 2006). In turn, future work on the approximate structure preserving semantic matching proceeds at least in the following directions: (*i*) conducting an extensive evaluation, and (*ii*) extending the matching approach for dealing with fully-fledged GIS ontologies.

## ACKNOWLEDGMENTS

This work has been supported by the OpenKnowledge[16] project (FP6-027253). We are grateful to Fausto Giunchiglia, Mikalai Yatskevich, Fiona McNeill and Paolo Besana for many fruitful discussions on the structure

[11] http://dip.semanticweb.org/
[12] http://www.eu-orchestra.org/
[13] http://www.workpad-project.eu
[14] http://nicta.com.au/research/projects/smart_applications_for_emergencies
[15] http://www.ist-cascom.org/
[16] www.openk.org/

preserving semantic matching.

## REFERENCES


Aggarwal R., Verma, K., Miller J.A., and Milnor W. Constraint driven web service composition in METEOR-S. In *Proceedings of IEEE SCC*, pages 23-30, 2004.

Avesani, P., Giunchiglia F., and Yatskevich, M. A large scale taxonomy mapping evaluation. In *Proceedings of ISWC*, pages 67-81, 2005.

Bernard, L., Craglia, M., Gould, M., and Kuhn W. Towards an SDI research agenda. In Proceedings of the *EC-GI & GIS Workshop,* 2005.

Denning, P. Hastily Formed Networks. *Communications of the ACM*, 49(4):15-20, 2006.

Euzenat J. and P. Shvaiko, P. *Ontology matching*. Springer, 2007.

Giunchiglia F. and Shvaiko P. Semantic matching. *The Knowledge Engineering Review*, 18(3):265–280, 2003.

Giunchiglia F. and Walsh T. A theory of abstraction. Artificial Intelligence, 57(2-3):323-389, 1992.

Giunchiglia, F., Yatskevich, M., and Giunchiglia, E. Efficient semantic matching. In *Proceedings of ESWC*, pages 272–289, 2005.

Giunchiglia, F., M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, IX:1-38, 2007.

Giunchiglia, F., Yatskevich, M. and McNeill, F. Structure preserving semantic matching. In *Proceedings of the ISWC+ASWC Workshop on Ontology Matching*, pages 13-24, 2007.

Giunchiglia, F., Yatskevich, M., and Avesani, P. A large scale dataset for the evaluation of matching systems. In *Posters of ESWC*, 2007.

Groot R. and J. McLaughlin J. Geospatial Data Infrastructure: Concepts, Cases and Good Practice. *Oxford University Press*, 2000.

Iannella, R., Henricksen, K., and Rinta-Koski, O. Towards a Framework for Crisis Information Management Systems. In *Proceedings of TIEMS*, 2007.

Mecella, M., Angelaccio, M., Krek, A., Catarci, T., Buttarazzi, B., and Dustdar, S. WORKPAD: an Adaptive Peer-to-Peer Software Infrastructure for Supporting Collaborative Work of Human Operators in Emergency/Disaster Scenarios. In *Proceedings of CTS,* pages 173-180, 2006.

Miller A. G. WordNet: A lexical database for English. *Communications of the ACM*, 38(11), pages 39–41, 1995.

Nebert D.. Developing Spatial Data Infrastructures. The SDI CookBook, version 2.0. *GSDI-Global Spatial Data Infrastructure*, 2004.

Noy N.. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

Oundhakar, S., Verma, K., Sivashanugam, K., Sheth, A., and Miller J.. Discovery of web services in a multi-ontology and federated registry environment. *International Journal of Web Services Research*, 2(3):1-32, 2005.

Paolucci, M. Kawamura, T., Payne, T., and Sycara K. Semantic matching of web services capabilities. In *Proceedings of ISWC*, pages 333-347, 2002.

Parent, C., Spaccapietra, S. and Zimanyi, E. *Conceptual modeling for traditional and spatio-temporal applications: MADS approach.* Springer, 2006.

Robertson D. A lightweight coordination calculus for agent systems. In *Declarative Agent Languages and Technologies,* pages 183-197, 2004.

Robertson D., Walton C., Barker A., Besana P., Chen-Burger Y., Hassan F., Lambert D., Li G., McGinnis J., Osman N., Bundy A., McNeil F. van Harmelen F., Sierra C., and Giunchiglia F.. "Models of interaction as a grounding for peer to peer knowledge sharing". *Advances in Web Semantics*, Vol. 1, 2007.

Shvaiko P. and Euzenat J. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV:146–171, 2005.

Tanasescu, V., Gugliotta, A., Domingue, J., Davies, R., Gutiérrez-Villarías, L., Rowlatt, M., Richardson, M., and Stinčić, S. A Semantic Web Services GIS based Emergency Management Application. In *Proceedings of ISWC*, pages 959-966, 2006.

Valiente G.. *Algorithms on Trees and Graphs.* Springer, 2002.