RESCUER

European Project Nº: **FP7-614154**
Brazilian Project Nº: **CNPq-490084/2013-3**
Project Acronym: **RESCUER**

Project Title: **Reliable and Smart Crowdsourcing Solution for Emergency and Crisis Management**

Instrument: **Collaborative Project**
European Call Identifier: **FP7-ICT-2013-EU-Brazil**
Brazilian Call Identifier: **MCTI/CNPq 13/2012**

# Deliverable D1.5.2
# Integrated Platform Demonstrator 2

Due date of deliverable: PM20
Actual submission date: October 12, 2015



Start date of the project: **October 1, 2013 (Europe)** | **February 1, 2014 (Brazil)**
Duration: **30 months**
Organization name of lead contractor for this deliverable: VOMATEC International GmbH (Vomatec)

| Dissemination level | | |
|---|---|---|
| PU | Public | ✓ |
| PP | Restricted to other program participants (including Commission Services) | |
| RE | Restricted to a group specified by the consortium (including Commission Services) | |
| CO | Confidential, only for members of the consortium (including Commission Services) | |

RESCUER

# Executive Summary

## Integrated Platform Demonstrator 2

This document presents deliverable D1.5.2 (Integrated Platform Demonstrator 2) of project FP7-614154 | CNPq-490084/2013-3 (RESCUER), a Collaborative Project supported by the European Commission and MCTI/CNPq (Brazil), that targets at supporting emergency organizations by crowdsourcing information in crisis situations. Full information on this project is available online at http://www.rescuer-project.org.

Deliverable D1.5 provides the results of Task 1.5 (Integrated Platform Demonstrator) that are concerned with the design and implementation of an integration platform for all components developed in the RESCUER project as well as their communication patterns and interfaces.

An integration platform that follows the hub-and-spoke topology was set up using the industry standard message-oriented middleware RabbitMQ. Consequently, RESCUER components can publish and subscribe JSON messages based on certain topics. The publish/subscribe pattern leads to a complete decoupling of sending and receiving components.

D1.5.2 adds to the previous version of this document the definition of the information flow between mobile devices in the crowd and a portal solution in the command and control centre so that follow up interaction is supported. Message topics and contents to this intent have been defined.

Additionally there are data analysis components, which subscribe to messages from the crowd as well and publish analysis results. Message topics and contents for publishing analysis results have also been defined, but it is in the purpose of next iteration to investigate them more in-depth as well as to define the information flow from the command centre to the crowd for crowd steering and public communication purposes.

**List of Authors**

Silas Graffy – Vomatec (1st Version)
Matthias Breyer – Vomatec
Laia G. Pedraza – Vomatec

**List of Internal Reviewers**

Tobias Franke – DFKI (1st Version)
Taslim Arif – Fraunhofer
Vaninha Vieira – UFBA (1st Version)
Andreas Poxrucker – DFKI (2nd Version)
Rebeca Barros – UFBA (2nd Version)
Karina Villela – Fraunhofer (2nd Version)

RESCUER

# Contents

# 1. Introduction

## 1.1. Purpose

The RESCUER project aims at developing a smart and interoperable computer-based solution for supporting emergency and crisis management by a crowdsourcing approach, with a special focus on incidents in industrial areas and on large-scale events. The idea is to enable crowds (workers in industrial areas and visitors of large-scale events) to directly interact with the command and control centre, for example by sending so-called 'incident reports' through a smartphone app. Both the concept and the objectives of the RESCUER project are described in the Description of Work (DoW), which is part of the EC Grant Agreement (Annex I).

The purpose of this document (D1.5.2: Integrated Platform Demonstrator 2) is to describe the prototype developed in Task 1.5 Integrated Platform Demonstrator which aims at integrating the RESCUER platform components into one demonstrator. In the first iteration the task covered information gathering from the crowd and sending it to the command and control centre. In this second iteration the follow up interaction has been defined taking into account target-groups. The follow up interaction states the communication between the users of the Mobile Crowd Solution and the Emergency Response Toolkit to allow the exchange of information between them. This is one of the main features of RESCUER system since it offers the possibility to the command and control centre to get more accurate and specific information. After receiving initial information about an emergency situation, the command centre might need further information from people at the place of the emergency (eyewitnesses or first responders) in order to clarify some uncertainties. In addition, follow-up interaction mechanisms can be used to check the reliability of a piece of information. In large-scale events, where there are a huge group of people concentrated in a wieldy area, the idea is to get a broader overview of the situation without overloading people with several questions; different groups of people could be asked different questions and the answers together would improve situation awareness. For that, is necessary to spot people in the crowd to join a target group.

This deliverable will be modified and improved together with the respective prototype during the next iteration.

## 1.2. Change Log

This deliverable is a living document and extends the contents of D1.5.1. The main changes are in section 2.4, which includes new configurations to achieve safe system recovery. Chapter 3 has been completed with the new flows for follow-up interaction and for the Context Components. Chapter 4 discusses the different requirements which we are more concerned to.

## 1.3. Partners' Roles and Contributions

VOMATEC was responsible for coordinating the elaboration of this document. VOMATEC also contributed with the implementation of the integration platform, which enables all components to exchange messages.

## 1.4. Document Overview

The remainder of this document is structured as follows.

- Chapter 2 explains the general idea and the approach of the integration platform.
- Chapter 3 explains how the integration platform has been set up and the Integrated Platform Demonstrator has been developed.
- Chapter 4 discusses the requirements taken into account.
- Chapter 5 analyses which are the further steps to take.
- Chapter 6 describes the conclusions drawn after improving and working with the first prototype.

# 2. System Integration Approach

The RESCUER system is composed of different components built using different software platforms. This chapter describes the approaches to achieve communication between components and how the integrated system satisfies RESCUER persistence requirements.

## 2.1. Functional Decomposition and Integration Platform Approaches

A functional decomposition of the overall set of RESCUER features resulted in different components developed by different project partners (see D1.2.2 System Architecture 2). To build up the overall Integrated Platform Demonstrator, those building blocks have to exchange information.

A naive approach would be that every component has a direct communication interface to every other component. For n components this *point-to-point* (or *peer-to-peer*) integration pattern would lead up to n*(n-1)/2 interfaces – and, even worse, each component could have its own technical and logical definition. This approach is not feasible, in practice, since it makes the overall system maintenance quite difficult, especially when components are changed, added, or removed.

A better approach for an integration pattern is *hub-and-spoke*. This means that there is (usually) one central component (the hub) that has a communication interface to every other component (the spokes). No other communication interface is allowed. In this case, for n components there are only n interfaces. Every time a component is changed, added, or removed a single communication interface (the one between the component and the hub) has to be adjusted. This approach simplifies the overall system maintenance and makes it easy to change and extend it.

## 2.2. Building a Central Integration Platform

A simple approach to build a *hub-and-spoke* based integration platform would be to design and implement a hub as a *domain-aware central middleware*. This middleware would provide an information exchange platform, and deal with crosscutting concerns (e.g., central data persistency, user roles and access control management, components management, among others). However, such a solution has significant drawbacks, both technical and from a project management point of view.

From a technical perspective, providing our own implementation of a hub integration platform makes it difficult to guarantee non-functional requirements like high availability and reliability, scalability, security (for example, transparent transport layer encryption), support to different client platforms and operating systems, among others. From a project management point of view, this would end up in a dependency of every system component. However, after a publicly funded research project like RESCUER ends, the partners may lack budget to maintain the components that other partners depend on. This approach would make the economic exploitation of the project results quite complicated, since a lot of bilateral contracts would have to be signed.

To overcome the drawbacks of a hub implementation, an existing industry standard component can be used. In RESCUER we have opted for this solution and chosen RabbitMQ[1], a *message-oriented middleware*, for that task. RabbitMQ was mainly chosen because of its interoperability, scalability, and performance advantages. Other considered alternatives, including but not limited to Open AMQ[2], ejabberd[3], Apache ActiveMQ[4], Open Message Queue[5], ZeroMQ[6], MassTransit[7], Microsoft BizTalk[8], and Apache Camel[9], were discarded for different reasons. Some of them do not support all needed platforms and programming languages, others do not provide infrastructure features like transport layer security out of the box. Some middleware systems mainly target on integrating legacy applications instead of integrating new ones. Other middleware systems also use communication protocols that imply in disadvantages because they have a bad performance or target on other use cases. From the remaining systems, RabbitMQ was selected for being a widely spread industry standard offering a lot of community and also commercial support.

In RabbitMQ, and most other message-oriented middleware solutions, every component in a system (the spokes) can send messages with a certain topic to the hub. On the other hand, each component in the system can subscribe to messages of one or more topics. Using topics instead of addressing messages to a certain receiver, we completely decouple senders and receivers of messages. A sender does not have to care if there is a receiver for its message or not, or if there might be several receivers. The sender just sets the topic of the message and pushes it to the middleware. Furthermore, a receiver of a message (subscriber to a topic) does not have to care if there is one (or more) sender of messages with the subscribed topic or not. This level of decoupling together with the usage of an industry standard message exchange protocol (AMQP, *Advanced Message Queuing Protocol*) makes it as easy as possible to introduce, remove, and change components in the RESCUER Integrated Platform Demonstrator. In *enterprise application integration*, this approach is usually referred to as *event-driven architecture*, where sending a message is considered an event.

## 2.3. Excursus: Rabbit MQ

RabbitMQ is a message broker using the networking protocol AMQP; it enables communication between different applications accepting and forwarding messages with binary data.

The basic concepts of RabbitMQ are *producer*, *consumer*, and *queue*. A producer is an application that sends messages to the broker (producing messages). A consumer is an application that receives messages from the broker (consuming messages). When the messages arrive to the broker, they are stored in queues. A queue is a buffer and it has no storage limit. A queue can be

[1] https://www.rabbitmq.com/
[2] http://www.openamq.org/
[3] https://www.ejabberd.im/
[4] http://activemq.apache.org/
[5] https://mq.java.net/
[6] http://zeromq.org/
[7] http://masstransit-project.com/
[8] http://www.microsoft.com/en-us/server-cloud/products/biztalk/
[9] http://camel.apache.org/

filled by several producers, and consumed by several consumers. The consumers, producers, and the broker do not need to run on the same machine.

Queues have a *durability* property to be configured. A *durable* queue is stored in the disk and in case that the system reboots, e.g. because of a crash, the queues are not lost but set again. Messages have the *persistence* property. It also means that they are stored in the memory and therefore recoverable in case of a system reboot. Thus, a persistent message queued in a durable queue will be recovered in case of a fail of the system. This is a property that can be chosen by the producer of the message.

The producers send messages to an *exchange*. The exchange is in charge of receiving messages, and routing and pushing them into a queue or several queues, or discarding them. Figure 1 shows a simple example of a message routing. A producer sends a message to RabbitMQ which is handled by the exchange. The exchange routes the message to a single queue. This queue is consumed by only one consumer, which gets this message.

An exchange and a queue are linked by a *binding*; which is the rule that the exchange follows to route the messages. The rule that defines what should be done with a message, which is the meaning of a binding, depends on the *exchange type*. The exchange types are: *direct*, *fanout*, *topic*, and *headers*.

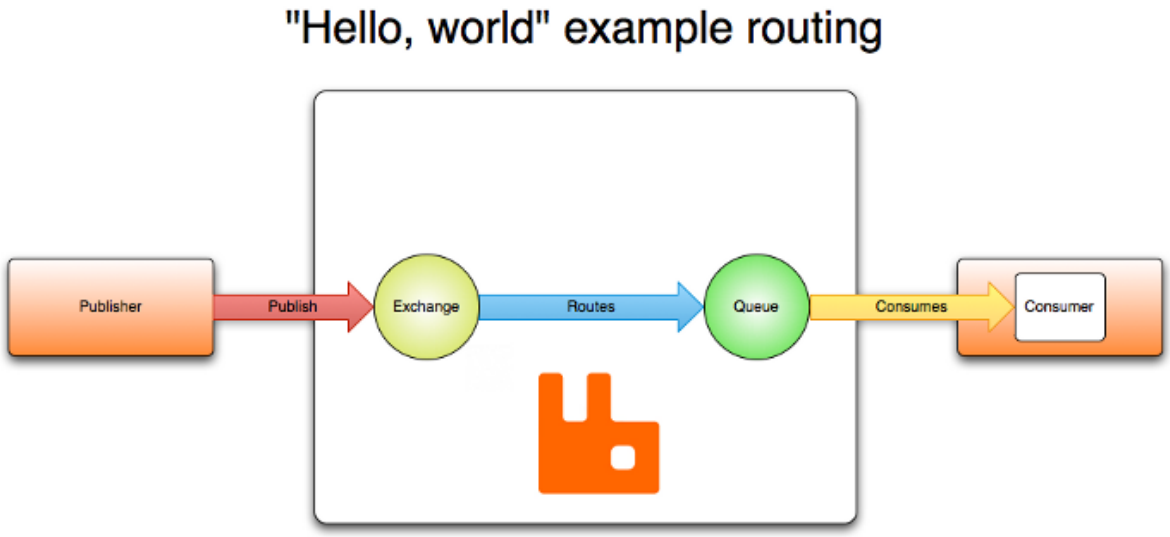


**Figure 1: Diagram of general message routing in RabbitMQ (Source: RabbitMQ AMQP Concepts[10])**

- **Direct**: messages are sent with a routing key and are pushed to queues bound to the exchange with exactly the same routing key (Figure 2).

[10] https://www.rabbitmq.com/tutorials/amqp-concepts.html

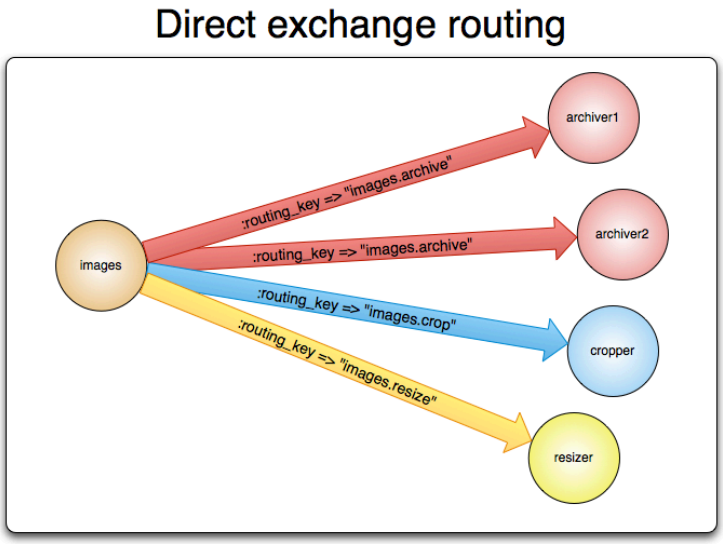


**Figure 2: Direct exchange (Source: RabbitMQ AMQP Concepts[10])**

- **Fanout**: messages are broadcasted to all queues bound to the exchange which does not take into account any routing key (Figure 3).

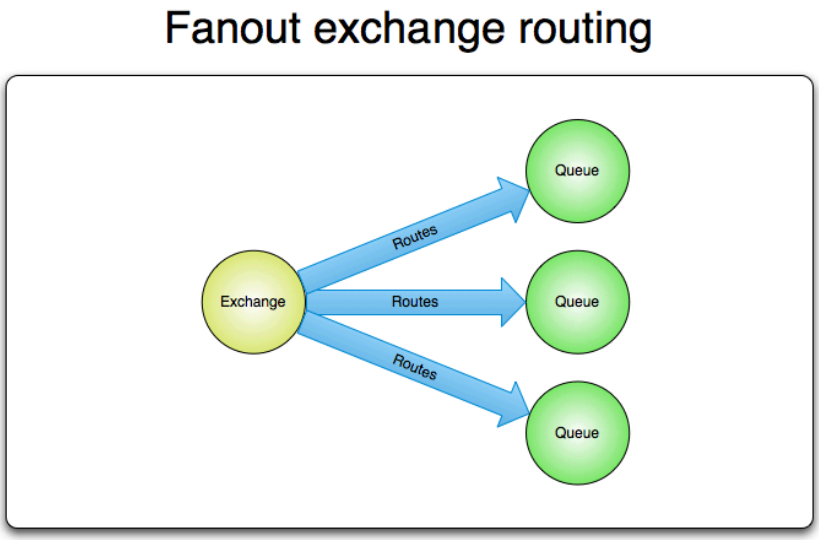


**Figure 3: Fanout exchange (Source: RabbitMQ AMQP Concepts[10])**

- **Topic**: messages are routed to queues that have a pattern as routing key matching the topic of the message. The idea is to get messages from several publish/subscribe pattern variations.
- **Headers**: messages are routed following the attributes of the message header. The exchange ignores the routing key.

In the RESCUER system, a message may be consumed by different modules for different purposes. Each module fetches the messages depending on different properties, leading to different routing keys patterns. Therefore, the most suitable exchange for our case is the topic exchange. A header exchange would be possible as well, but this would lead to a higher complexity without adding value, since all RESCUER requirements are already met by the topic exchange type.

The topic exchange uses routing keys structured as a list of words separated by dots (e. g. “lazy.brown.dog”). These words are meant to reflect a property of a message, so the consumers can consume only the messages they are interested in. The number of characters in such a phrase is limited to 255. Words can be substituted by the placeholders “*” or “#”. The placeholder “*” means whatever word, but exactly one word, and the placeholder “#” means whatever zero or more words, e. g. with routing key “#” all messages will be consumed. These placeholders are used to provide flexibility when using patterns; otherwise the topic exchange would act like the direct exchange. Figure 4 shows an example of using patterns as routing keys, where each queue is bound with a routing key pattern. This means that the arrived messages would be saved in the queues if their routing key follows this pattern.

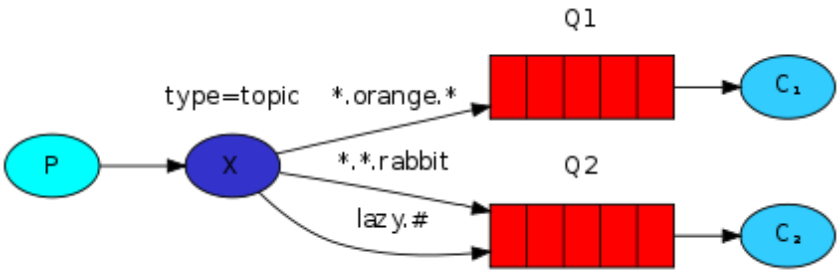


**Figure 4: Topic exchange using patterns as routing keys (Source: RabbitMQ Topic Exchange[11])**

For more details about exchange types and message routing see the AMQP explanation[12] of RabbitMQ.

## 2.4. Configuration and Persistence in the Integration Middleware

To define the topics and contents for the messages managed by senders and receivers, we need to know the information needed and provided by the RESCUER components. A questionnaire was set up and answered by each developer of a RESCUER component. The questionnaire is attached in appendix A and the answers are in appendix B.

The questionnaire was used to collect the information needs of each RESCUER component. In addition, it also asks for the information they offer. The information offered by each component defines the messages published by the component. The information needs define the messages to

[11] https://www.rabbitmq.com/tutorials/tutorial-five-python.html

[12] RabbitMQ: AMQP 0-9-1 Model Explained. https://www.rabbitmq.com/tutorials/amqp-concepts.html.

be consumed. The information needed by all components also determine the definition of the routing keys. Several components might want the same message depending on different properties.

An example is: the Mobile Crowdsourcing Solution (MCS) publishes the messages created by the crowd; the text analysis component wants to consume these messages but only the ones containing text; the video analysis component only wants the ones containing video; the image analysis component only wants the ones containing images; and the emergency response toolkit wants all of them – plus the analysis results afterwards. In this case, the attributes of the messages coming from the crowd that are interesting for each component are known. This is explained deeply in Chapter 3.

The questionnaires also asked for the platform and the programming language used for the implementation of the component. This information was considered in order to choose a message broker with client libraries available for all needed technology stacks as well as to find a suitable way to serialize and deserialize the message contents. Thus, RabbitMQ was chosen taking into account that there are available libraries for both aspects (sending/receiving of messages and serialization/deserialization of their contents) for all platforms and programming languages used in the project.

Based on the received answers, a topic pattern was defined for RabbitMQ messages and a structure was defined for their contents. To serialize the message contents, the project consortia agreed to use JSON (JavaScript Object Notation), encoded in UTF-8.

It is important that components consuming data take into account that the queue to consume messages is created the first time it is defined. From then on, even if the component breaks down or turns off, the queue is not removed, if there is not an explicit request for this. However, the data received before the queue is defined is not saved to this queue. Therefore, the system should start consuming data at the first possible moment. In case of the mobile devices, this should be as soon as they are installed and started.

Persistency of messages is a requirement of the ERTK. In case the integration middleware breaks down, the queues and messages have to be re-established. As already explained, RabbitMQ offers this possibility through the setting of queues as durable and the setting of messages as persistent. This means that, in case of a system fail, during its restart, the durable queues would be redefined and the persistent messages added again into these queues. All queues not set as durable would be lost and messages not set as persistent would not be saved in order to be added later on.

# 3. Prototype of the Integrated Platform Demonstrator

## 3.1. Overview

The Integrated Platform Demonstrator is responsible for ensuring the connection between the RESCUER platform components.

The RESCUER platform components are:

- Mobile Crowd Sourcing Solution – Part for actively sending incident reports.
- Mobile Crowd Sourcing Solution – Part for passively recording crowd density data, a.k.a. Crowd Sensing Solution. It consists of a sender on the mobile devices themselves and a sensor data receiver, which aggregates the data received and is connected to the integration platform.
- Data Analysis Solutions:
  - Text Analysis Component
  - Image Analysis Component
  - Video Analysis Component
- Emergency Response Toolkit, which consists of the user interface (web portal) and a subcomponent that cares about data transfer.
- Context Components, which aggregate, store, sort crowd data, the outputs of the analysis components and other context information, and offers this context information to the other RESCUER components. It includes the Data Sorting component and the Incident Aggregation component.

Each of these components needs to send and receive data to and from the others. Figure 5 shows a high-level overview of the message exchange between RESCUER components that are within the scope of the second project iteration.

The first project iteration focused on information gathering. The second iteration concentrates on covering group-targeted follow-up interaction. For this purpose, the data format has been made dynamic to provide flexibility, and new routing keys were defined to establish the follow up communication.

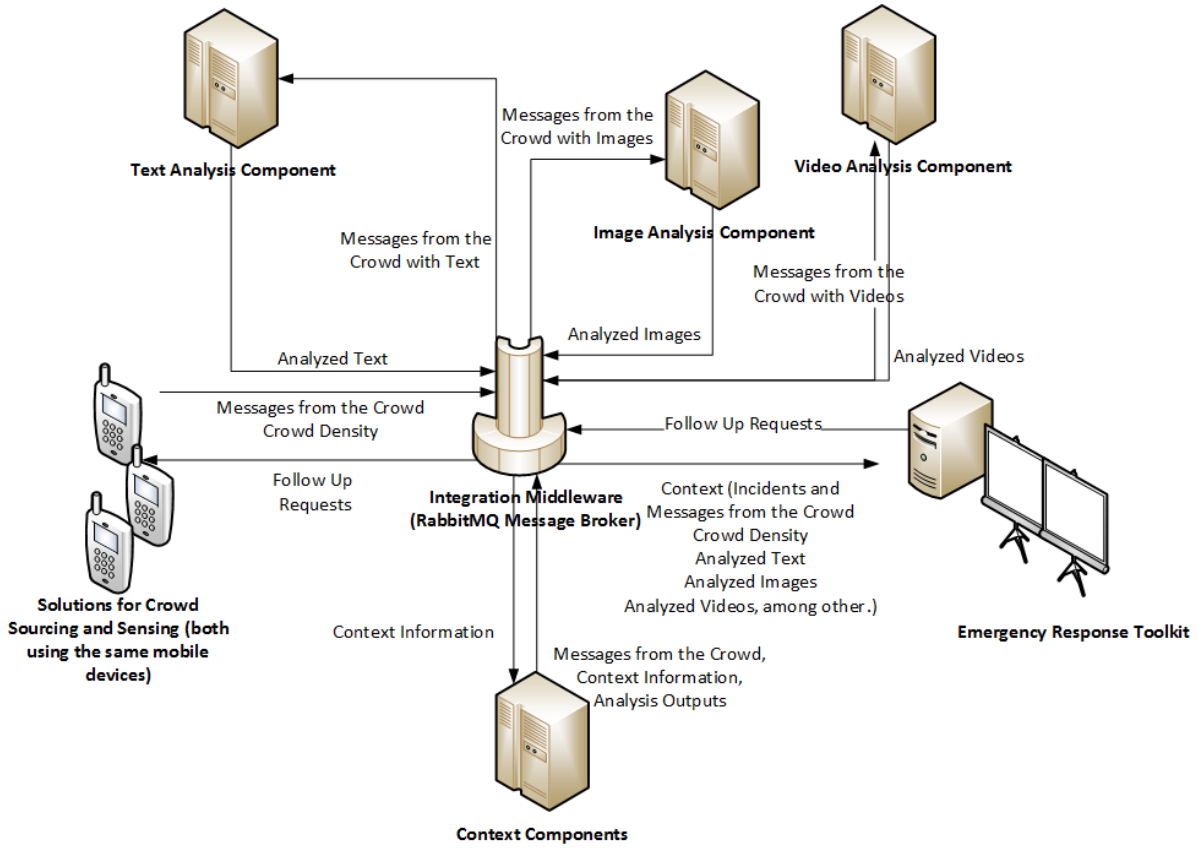


**Figure 5: RESCUER Components Overview 1**

## 3.2. Definitions and Infrastructure

- We use a single exchange of type 'topic' in RESCUER. The name of the global topic exchange, where messages have to be sent to, is "RESCUER_Global_Topic_Exchange".
- VOMATEC has set up a RabbitMQ server, connection details have been sent by e-mail to the consortia members, since this deliverable is public.
- From a RabbitMQ's point of view, message payload is just a byte array. Instead of passing bytes around in RESCUER, message payload for us is a JSON object. Details are defined in the following section of this document. The message payload is encoded using UTF-8.
- Messages could contain binary data, for example images and videos sent from the crowd. Those BLOBS are not directly included as part of the message payload in RESCUER, so they are not uploaded to RabbitMQ. Instead, they are uploaded to an Amazon server, where the files can be accessed through HTTP GET. Therefore only the URL to the uploaded data is placed in the JSON structure of the message payload.
- DFKI has set up the storage server, connection details have been sent by e-mail to the consortia members, since this deliverable is public.
- In general, message topics follow the following pattern:

- The topic starts with a constant for the subject like “crowdmessage” followed by a point (.).
- Then the topic continues with none, one, or many subject-specific parameter values, separated by a dot (.), for example “true.5.red”. The amount of parameter and the allowed parameter values are defined for each type of information flow in the sections below.
- The characters “#” and “*” have special meanings, so they have to be avoided when setting a topic. When subscribing to a topic pattern, “*” is a placeholder for exactly one word and “#” for zero to multiple words.

## 3.3. Information Flow

The different queues used in the RESCUER Integration Platform are defined depending on the information flows of the system. The information flows covered in this document are defined by the use cases listed below and described in D1.1.2 (Requirements Specification 2).

The use cases regarding the information sent from the crowd using the MCS are:

- UCMBL03 – Send Incident Notification.
- UCMBL04 – Send Report.
- UCMBL05 - Answer Follow-up Information.

This information is consumed by the Context components, which aggregate the data, store it and share the updated context with the other components.

The use cases regarding the data for the ERTK are:

- UCETK06 – Ask Follow-up Information.
- UCETK07 – Display Emergency.
- UCETK08 – Display Incident.
- UCETK09 – Display Report.
- UCETK11 – Coordinate Forces.

The use case UCETK11 has been excluded of the project scope after the first project review meeting.

## 3.4. Information Flow from the Crowd

Information flow from the crowd can be separated in crowd sourcing and crowd sensing. Crowd sourcing means that people in the crowd actively send data using their mobile devices. These data originate the incident reports. Crowd sensing works in the mobile devices of the people in the crowd. It passively gathers sensor information and sends it as a continuous stream, e.g. to estimate crowd density. Since this is a continuous stream of data, a message-based approach is not suitable. Therefore, there is a dedicated sensor data receiver, which aggregates crowd sensing data in order to provide the current crowd density (overall output of crowd sensing solution). Its results can be accessed via HTTP GET. The format is defined in appendix C, the URL is consortia-internal and has

been distributed via e-mail by DFKI. By polling it, an interested component like the ERTK could decide on its own in which interval it wants to receive updates. However, this has the drawback that coupling between components gets tighter since the interested component, the ERTK, has to know where to get the data from. Therefore, in the last project iteration a new approach with an abstraction component may be set up.

The crowd sourcing has two variants: the Incident Reports or Notifications and the Follow up. The first one is depicted in this section, but the follow up is treated in Section 3.8.

Table 1 defines how Incident Reports and Incident Notifications (UCMBL03 and UCMBL04) are actively sent from the MCS.

**Message topic:** crowdmessage.<text>.<image>.<video>.<role>

**Table 1: Crowd messages topic definition**

| Parameter | Description | Value |
|---|---|---|
| **text** | Indicates if the message contains text information | [true, false] |
| **image** | Indicates if the message contains one or more images | [true, false] |
| **video** | Indicates if the message contains one or more videos | [true, false] |
| **role** | Indicates the role of sender | [civilian, supporting, workforce] |

Using the topic definition for crowd messages, the image analysis component can subscribe to the topic pattern “crowdmessage.*.true.*.*” or even “crowdmessage.*.true.#”, for example. The ERTK subscribes to “crowdmessage.#”. The MCS instance that wants to send an incident report written by a workforce, that contains text and video but no images will do so by publishing a message with the topic “crowdmessage.true.false.true.workforce”.

The idea behind the sender role being part of the topic definition is that messages sent by a workforce can receive higher priority than other messages. To be able to do this in an elegant way, they can be filtered by defining different binding keys when subscribing to a queue.

The Incident Notification and Incident Report share the same data structure (see below). The difference is that the Incident Notification only sends the keyword with other basic metadata.

**Incident Report Content**

The JSON format of an incident report sent by the MCS is depicted below. Each element inside a report is georeferenced when possible, e.g. images, videos or the position of the user when sending the report (“reportSender”).

```
{
  "reportIdentifier": "113245",
  "reportTimestamp": "2014-07-04T12:48:06.4147832+02:00",
```

```
"keyword": "FIRE",
/*“FIRE”; “EXPLOSION”; “ENVIRONMENT”; “GAS”, “HELP” */
"reportPosition": {
  "latitude": 47.3742753222995,
  "longitude": 8.5428106828621733,
  "altitude": 0.0
 },
"reportSender": {
  "personRole": “CIVILIAN”,
  /*“CIVILIAN”; “SUPPORTING”; “WORKFORCE”*/
  "personPosition": {
    "latitude": 47.3742753222995,
    "longitude": 8.5428106828621733,
    "altitude": 0.0
  },
  "movementProfile": [
  {
    "position": {
      "latitude": 47.377275322299504,
      "longitude": 8.5498106828621729,
      "altitude": 0.0
    },
    "date": "2014-07-04T12:48:06.4147832+02:00"
  },
  {
    "position": {
      "latitude": 47.3712753222995,
      "longitude": 8.5468106828621728,
      "altitude": 0.0
    },
    "date": "2014-07-04T12:48:06.4147832+02:00"
  }]
},
"mediaPhoto": [
{
  "mediaPath": "http://exampleURL.com/images/fire3.jpg",
  /*URLofblobstorageserver,see section3.2*/
  "mediaIdentifier": "1132451",
  "size": null,
  "quality": null,
  "type": "image/png",
  "mediaTimestamp": 0,
  "location": {
    "latitude": 47.3742753222995,
    "longitude": 8.5428106828621733,
    "altitude": 0.0
  },
  "accelerometer": [
  {
    "z": 9.4954986572266,
    "y": 1.6761016845703,
    "x": -0.027999877929688,
    "t": 1411057007478
  }],
  "gyroscope":[
  {
```

```
    "r": -0.0033111572265625,
    "y": 0.0076751708984375,
    "t": 1411057007478,
    "p": -0.003021240234375
  }]
},
{
  "mediaPath": "http://exampleURL.com/images/incident.png",
  /*URLofblobstorageserver, see section3.3*/
  "mediaIdentifier": "1132452",
  "size": null,
  "quality": null,
  "type": "image/png",
  "mediaTimestamp": 0,
  "location": {
    "latitude": 47.3742753222995,
    "longitude": 8.5428106828621733,
    "altitude": 0.0
  },
  "accelerometer": [
  {
    "z": 9.4954986572266,
    "y": 1.6761016845703,
    "x": -0.027999877929688,
    "t": 1411057007478
   }],
  "gyroscope":[
  {
    "r": -0.0033111572265625,
    "y": 0.0076751708984375,
    "t": 1411057007478,
    "p": -0.003021240234375
  }]
}],
"mediaVideo": [
{
  "mediaPath": "http://exampleURL.com/data/videos/massa.mp4",
  /*URLofblobstorageserver, see section3.2*/
  "mediaIdentifier": "1132453",
  "size": null,
  "quality": null,
  "type": "video/mp4",
  "mediaTimestamp": 0,
  "location": {
    "latitude": 47.3742753222995,
    "longitude": 8.5428106828621733,
    "altitude": 0.0,
  },
  "accelerometer": [
  {
    "z": 9.4954986572266,
    "y": 1.6761016845703,
    "x": -0.027999877929688,
    "t": 1411057007478
  }],
  "gyroscope":[
```

```
    {
      "r": -0.0033111572265625,
      "y": 0.0076751708984375,
      "t": 1411057007478,
      "p": -0.003021240234375
    }]
  }],
  “formResponse”:[
  {
    "identifier":"g5d5-gsetg-65hs-sy69",
    "name": "Are you injured?",
    "value": “”,
    "type": "bool",
    “timestamp":"2015-03-24T12:18:27.0804138+02:00"
  }],
  “chatMessages”:[
  {
    "identifier":"y8s5-wq0l-6xcc-omp7",
    "name": "chat",
    "value": “There is a big fire here in the office, please help us!”,
    "type": "text",
    “timestamp":"2015-03-24T10:11:13.240Z"
  }]
 }
```


## 3.5. Information Flow from the Context Components

The Context Components are still under implementation therefore the following details are a draft. Those components consume data from MCS, the Data Analysis Components or ERTK. It aggregates and stores the data. Depending on the content of the data, it publishes the updated context to the corresponding component. For example, when a new report arrives in the system, the Incident Aggregation component assigns it to a new incident or to an existing incident. Then it stores this information into the database and it sends the new context information to the ERTK. In case this report contains data to be analysed, the Data Sorting component process it and sends it to the corresponding Analysis Component.

**Message topics:** See following subsections.

**Table 2: Context Components message topics definition**

| Parameter | Description | Value |
|---|---|---|
| **text** | Indicates if the message contains text information | [true, false] |
| **image** | Indicates if the message contains one or more images | [true, false] |
| **video** | Indicates if the message contains one or more videos | [true, false] |
| **type** | Indicates which type of context change it contains | See following subsections. |

### 3.5.1. Outputs from Incident Aggregation

Anytime the context changes, the Incident Aggregation component publishes the new context. This happens when a new report, partial report, follow-up or an analysed result, among others, is received.

**Message topic:** context.<trigger>

**Table 3: Incident Aggregation message topic definition**

| Parameter | Description | Value |
|---|---|---|
| **Trigger** | Indicates what has changed in the context | [newincident, updatedincident] |

The data transferred when a new incident has been added to the RESCUER context has the format of an incident. In the case of an update, the data is wrapped with metadata in order to track what has been updated (encapsulated inside "parameters" property) and why (defined in "type" property). An incident is updated when one of its reports has been updated ("UP-REPORT"), a new report is received ("NEW_REPORT"), the user in the ERTK moves reports between incidents ("MOVE_REPORT"), or merges incidents ("MERGE").

The following examples of message topics and data formats show the different use cases.

When a new incident arrives, a message topic with the trigger as "newincident" is used and the message transferred follows the incident format.

**Message topic:** context.newincident

**Data Format**: {

```
{
	"timestamp": "2015-09-01T10:33:00.4147832-03:00",
	"updatedTimestampString": "2015-10-12T08:22:50.0000828+02:00",
	"reports": [...],
	"reportsMemory": [],   /*it will be use when moving reports feature*/
	"keyword": "explosion",
	"position": {
		"latitude": 38.69951,
		"longitude": -9.18349
	},
	"reliability": "low",
	"incidentIdentifier": "1_dELdOijUA3F",
	"mergedIncidents": []
}

}
```

An update of an incident uses the message topic with the trigger "updatedincident" and the format of the messages inside this queue follow the data formats shown below. The property "type"

is different depending on the case, e.g. the first one is an incident update after receiving an update of a report. In this case, there is an identifier of the updated report inside “parameters”.

**Message topic**: context.updatedincident

**Data Format:** {
```
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00",
  "type": " UP_REPORT",
  "data": { Incident },
  "parameters": [ { "reportIdentifier":" d8bvfc2fd-22fa"}]
/* updated report id */
}
```

The following example shows an incident update after receiving a new report. The “parameters” property contains the identifier of the new report.

**Data Format:** {
```
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00",
  "type": "NEW_REPORT",
  "data": { Incident },
  "parameters": [ { "reportIdentifier":" d8bvfc2fd-22fa"}]/* new report id */
}
```

The following example shows an incident update after the user of the ERTK merges two incidents. The “parameters” property contains the identifier of the incident which holds the one merged.

**Data Format:** {
```
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00",
  "type": "MERGE",
  "data": [ array<Incident> ],
  "parameters": [ { "holderIdentifier":" d8bvfc2fd-22fa"}] /*incident holder
identifier*/
}
```

The following example shows an incident update after the user of the ERTK moves a report between two incidents. The “parameters” property contains the identifier of the incident which held the report and the identifier of the incident which has been moved to.

**Data Format:** {
```
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00",
  "type": "MOVE_REPORT",
  "data": [array<Incident> ],
  "parameters": [ { "sourceIncident":"d8vd-22fa"}, {"destinationIncident":"v2d6-
b5hw"}, {"movedReport":"oweb3-5rwl"}] /*[sourceIncident, destinationIncident]*/
}
```

### 3.5.2. Outputs from Data Sorting consumed by Data Analysis Solutions

The Context Components consume the Incident Reports which may contain text, images or videos to be analysed. The system prioritizes the elements to be analysed depending on the context. This processing is performed by the Data Sorting component. This component publishes the data to be analysed under the following message topic, depending on which of the three analysis components has to be sent to.

**Message topic:** analysisqueue.<text>.<image>.<video>

**Data Format** {

```
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00"
  "data": Report
}
```

### 3.5.3. Outputs after a request of Context

The Context Module offers an API to access the central data of RESCUER to the other RESCUER components. The request of this data is done through the Integration Platform as well as the response.

**Message topic**: contextresponse.<requestIdentifier>

**Table 4: Request Context message topic definition**

| Parameter | Description |
|---|---|
| **requestIdentifier** | It corresponds to the identifier of the request. |

The format of the data, to response to a request context, is depicted below. The property “data” depends on the request.

**Data Format:** {

```
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00"
  "data": (dynamic: It can be a set of reports, an incident, a set of incidents,
set of pictures, set of videos, set of analysed elements)
  "status" : "SUCCESS"/"ERROR"/"NOTFOUND"/"TIMEOUT"
}
```

## 3.6. Information Flow from the Analysis Components

The components that analyse text, image, and video data will subscribe to "crowdmessage" topics to get all raw data from the crowd. After the analysis, they will publish their results using an equivalent topic pattern. The role of the original sender is just passed through the analysis component, again to enable subscribers to filter or order based on the sender role. Table 5 defines how data analysis data are sent from data analysis components.

**Message topic:** analysedresult.<text>.<image>.<video>.<role>

**Table 5: Analysis message topic definition**

| Parameter | Description | Value |
|---|---|---|
| **text** | Indicates if the message contains text information | [true, false] |
| **image** | Indicates if the message contains one or more images | [true, false] |
| **video** | Indicates if the message contains one or more videos | [true, false] |
| **role** | Indicates the role of sender | [visitor, employee, workforce] |

The following examples show the data format for the output of each Analysis Component.

**Analysis Result Content for Text**

```
{
  "reportIdentifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  “originalText": "Please there is fire, a lot of fire everywhere and we need
help, we injuresfsd inw wz secseond flooor",
  "reportTimestamp": "2014-10-20T11:57:02.0804138+02:00",
  "information": [
  {
    "what": "medical assistance required",
    "who": "visitor",
    "where": "second floor",
    "observations": "observations of medical assistance message"
  },
  {
    "what": "fire",
    "who": "workforces",
    "where": "everywhere",
    "observations": "observations"
  }]
}
```

**Analysis Result Content for Images**


```
{
  "reportIdentifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "location": {
    "latitude": 47.372438581730449,
    "longitude": 8.5469272998321717,
    "altitude": 0.0
  },
  "timestamp": "11:57:01",
  "reportTimestamp": "2014-10-20T11:57:01.6748164+02:00",
  "analysedElementIdentifier": "m3",
  "information": [
  {
    "what": "fire",
    "who": "workforce",
    "where": "",
    "reliability": 0.1,
    "observations": ""
  }]
}
```


**Analysed Result Content for Video**


```
{
  "reportIdentifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "location": {
    "latitude": 47.3622089745065,
    "longitude": 8.5448428429742762,
    "altitude": 0.0
  },
  "timestamp": "2014-10-20T12:34:57.8339465+02:00",
  "measures": {
    "expanseVelocity": 1.08851767,
    "pedestrianM2": 10,
    "riskOfCongestion": true
  },
  "reportTimestamp": "2014-10-20T12:34:57.8339465+02:00",
  "analysedElementIdentifier": "m1",
  “information”: [
  {
    "what": "crowd",
    "who": "civilian",
    "where": "",
    "when": "00:05",
    "observations": ""
  }],
  "resolution" : "640x480",
  "quality" : 0.65
}
```


The Analysis Components need also the context to perform a smarter analysis. To request this context, the following message topic and data format is used.

**Message topic:** "contextrequest"

**Data Format**

```
{
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00"
  "type": "apiFunctionName"
  "parameters": [] /* function inputs */
}
```

## 3.7. Information Flow from the Command and Control Centre

ERTK communicates with MCS for the follow-up interaction and crowd steering, and with the Context Components to get the required information about the emergency. The data published by ERTK regarding follow-up interaction is detailed in section 3.8. The crowd steering communication will be designed at the next iteration.

The communication between ERTK and the Context Components has not been completely refined yet. There are two options: through the Integration Platform or directly by a REST API. The two approaches will be tested and the best solution regarding performance will be selected. For the first approach, a message topic must be transmitted through the RabbitMQ and it is the following:

**Message topic:** contextrequest

The data format, which does not depend on the type of communication, is the same as the one used by the analysis component to request context information.

**Data Format**

```
{
  "identifier": "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1",
  "timestamp":"2015-03-24T12:18:27.0804138+02:00"
  "type": "apiFunctionName"
  "parameters": [] /* function inputs */
}
```

## 3.8. Information Flow for Follow-Up

Support to follow-up interaction involves the communication between ERTK and MCS. This follow-up may start from an incident report or not. This means that the user in the command and control centre can communicate directly with a mobile device, which has already sent a report, or communicate to a target-group (e.g. workforces, supporting forces). Therefore, there is follow-up and group-target follow-up.

In addition, the request of follow-up may be a chat message or a form message. The form message specifies how should be the reply, e.g. giving "multichoice" options to reply. The chat

message belongs to a conversation, in which is not specified the type of the reply. It can be an image, a video or plain text. Each follow-up answer can also be a chat message or a form message.

Both chat and form messages are sent with the same message topic. The receiver is who interprets which type it is. However, the group-target follow up is filtered by different message topics.

### 3.8.1. Follow-Up

Table 6 defines how a follow-up for an Incident Report is sent from the ERTK to MCS.

**Message topic:** ertkreportfollowup.<reportIdentifier>

**Table 6: Follow up from a report message topic definition**

| Parameter | Description | Value |
|---|---|---|
| **reportIdentifier** | Indicates to which report belongs this follow up | string, e.g. "5affc0a0-22fa-4f79-b2ee-9d13c0fe88e1" |

The message content for the follow-up, chat format as well as form format, is shown with examples below. To achieve flexibility of the messages, a common structure is used for chat and forms, using dynamic data. More information about this can be found it in D4.4.2 (Emergency Response Toolkit 2).

**Form follow-up request message content**

```
{
  "requestIdentifier": "281f1fb7-5a6e-4b89-85a9-8fc22cf7b8c0",
  "reportIdentifier": "b988fb10-dc25-4061-bd76-e3172e0bbeef",
  "requestData": {
    "identifier": "24ff6169-c094-4fd7-a631-577f1a227330",
    "name": "How is the fire?",
    "type": "onechoice",
    "timestamp": "2015-04-22T17:25:33.0364338+02:00",
    "value": ["small", "medium", "large"]
  }
}
```

**Chat follow-up request message content**

```
{
  "requestIdentifier": "281f1fb7-5a6e-4b89-85a9-8fc22cf7b8c0",
  "reportIdentifier": "b988fb10-dc25-4061-bd76-e3172e0bbeef",
  "requestData": {
    "identifier": "24ff6169-c094-4fd7-a631-577f1a227330",
    "name": "chat",
```

```
    "type": "text",
    "timestamp": "2015-04-22T17:25:33.0364338+02:00",
    "value": "Can you provide more information?"
  }
}
```

When a user of the MCS receives a follow-up request from ERTK, the user may reply with another chat message or filling the form received. This data is considered an update of the report; therefore the ERTK does not need to subscribe to any special queue in order to get the replies.

### 3.8.2. Group-Target Follow-Up

In this use case, the ERTK user might be interested in asking a follow-up question to more than one MCS user. Then this message is not related to any report but to a group-target.

This group-target can be defined depending on user profile, location, and activity. The role of the user can be workforce, supporting force, or civilian. The location refers to a certain area inside the emergency area. The activity means whether the user has already had any interaction with the application (civilians who have not interacted with MCS to send a report should not be asked a follow-up question). The user role filtering may be done by the routing key, but the filter by location or by activity should be performed by the MCS itself.

**Message topic:** ertkgroupfollowup.<workforces>.<supporting>.<civilians>.<byLocation>.<byActivity>

In this case, the ERTK needs to subscribe to a queue in order to get the replies since they do not come from a report. Each request message sent from the ERTK contains a request identifier to be able to relate the reply with it. However, it should be tested, if the creation of large number of queues for requests produces performance degradation. In this case, it should be used one single queue for all the replies.

**Message topic:** “ertkgroupfollowup”.<requestIdentifier>

Table 7 defines how this target-group follow-up messages and replies are sent between ERTK and MCS.

**Table 7: Target-group follow up message topic definition**

| Parameter | Description | Value |
|---|---|---|
| **requestIdentifier** | Unique identifier of the request made by the ERTK | |
| **Workforces** | Indicates if the message is to be sent to workforces | [true, false] |
| **Supporting** | Indicates if the message is to be sent to supporting forces | [true, false] |
| **Civilians** | Indicates if the message is to be sent to civilians | [true, false] |
| **byLocation** | Indicates if the MCS must filter by the location of the user | |
| **byActivity** | Indicates if the MCS must filter by the activity of the user | |

The data format for the group follow-up is shown in the following examples.

**Group targeted follow-up request message content:**


```
{
  "requestIdentifier": "281f1fb7-5a6e-4b89-85a9-8fc22cf7b8c0",
  "requestData": {
    "identifier": "24ff6169-c094-4fd7-a631-577f1a227330",
    "name": "How is the fire?",
    "type": "one choice",
    "timestamp": "2015-04-22T17:25:33.0364338+02:00",
    "value": ["small", "medium", "large"]
  },
  "geozone": [{
    "longitude": -38.3225811,
    "latitude": -12.672151999999997
  },{
    "longitude": -38.3194311,
    "latitude": -12.672052000000008
  },{
    "longitude": -38.3208811,
    "latitude": -12.673302000000007
  },{
    "longitude": -38.3219811,
    "latitude": -12.672652000000014
  }],
  "byActivity" : true

}
```


**Group targeted follow-up reply message content:**


```
{
  "requestIdentifier": "281f1fb7-5a6e-4b89-85a9-8fc22cf7b8c0",
  "reportIdentifier": "b988fb10-dc25-4061-bd76-e3172e0bbeef",
  "requestData": {
    "identifier": "24ff6169-c094-4fd7-a631-577f1a227330",
    "name": "reply",
    "type": "text",
    "timestamp": "2015-04-22T17:25:33.0364338+02:00",
    "value": "medium"
  }
}
```

# 4. Integration Platform Quality Requirements

## 4.1. Overview

The Integration Platform deals with the exchange of the messages between all components of the RESCUER system. Therefore, some requirements should be in mind during its design. In case a critical requirement is not addressed, this might result in a severe degradation of the system quality or, in the worst case, a failure of the system. The critical requirements are scalability, data persistence, performance, and security.

## 4.2. Scalability

The RESCUER system deals with crowd data. This means that the number of source of information varies considerably, e.g. from 10.000 to 100.000 depending on the scenario. Thus, even if in a certain scenario the system can run only in one machine, it should be able to scale in order to adapt for bigger scenarios.

In terms of implementation, a simple solution with only one machine is currently held because it is sufficient for the evaluations. However, a research has been performed to analyse how the Integration Platform could scale. RabbitMQ has been chosen as the message broker, therefore a research of RabbitMQ scalability has been conducted.

The scalability of RabbitMQ depends on whether the architecture to use the tool scales or not. A way to scale is the clustering[13]. RabbitMQ works with nodes, each running the application and all sharing queues, exchange, etc. This collection of nodes is considered a cluster. There is different type of nodes. Some of them are I/O bounded because they write more often to disk. The queues lives in a node and it can be configured to which node each queue belongs to and the storage capacity of the node to ensure the message throughput.

All nodes are visible for the others. The cross-node publishing and consuming allows receiving a message in one node, transferring it to another node, and delivering it by this latter one. This has a performance penalty. Another issue to take into account refers the maximum size of a cluster, which should not exceed 32 nodes. The larger the cluster is, the bigger the performance penalty.

RabbitMQ could run into an overloading of messages. In case there is a high throughput of publishers and the consumers do not consume fast enough, the node may not be able to deal with all messages and it crashes. The system in such a case applies flow control of connections, where the system exerts “TCP back-pressure“ on the connections[14] protecting itself.

[13] https://www.rabbitmq.com/clustering.html
[14] http://bigwig.io/docs/message_throughput/

## 4.3. Data Persistence

RabbitMQ takes care of failure by replicating data to be able to recover it if necessary, which is really important when dealing with high reliability system, which is the case of the RESCUER System.

The queues can be defined as “durable”. The durability property ensures that this queue survives reboot. In addition, if the messages stored in this queue have the delivery mode set as “persistent”, they are also recovered after a reboot. Messages are written to disk before they are dequeued. These mechanisms allow the Integration Platform to satisfy the requirement of data persistence.

However, message persistence implies a 50 to 70% performance penalty

## 4.4. Performance

Few issues affecting performance have already been pointed in the previous sections. One of them is the TCP back-pressure to not run out of memory; it is a flow control that slows down the message rate of the publishers. Another one is the message persistence, which implies a 50 to 70% performance penalty, but it must be paid because of the nature of the application.

There are more issues that influence performance. However, going deeper into RabbitMQ technology is not part of our scope. There are tests performed[15] with different configurations and message sizes.

In our system the critical communication is the one coming from the crowd because it is an unknown number of consumers and publishers. Since scalability is possible, in case this data traffic and amount of connections overtake the Integration Platform, dedicated machines for this communication can be set. This would avoid performance issues because of the number of the users from the crowd.

A message coming from MCS is maximal 6KB. The crowd publishes more than consumes, therefore the nodes bound to their queues have to be aware of storage capacity.

Furthermore, there is a parameter called “prefetch”. This is the number of message that RabbitMQ delivers per time. It does not send more messages until the sent messages are acknowledged. Thus it is a parameter to be tuned to optimize performance. If it is too low, it degrades performance because the client may be waiting for another message while there are available messages, but too big may cause problems to the client. This parameter has not yet been configured for the RESCUER system. Currently is preconfigured to one, which ensures the correct functionality even though is not the optimized parameter. When the new architecture is defined, this parameter will be tested with different message loads.

[15] https://www.rabbitmq.com/blog/2012/04/25/rabbitmq-performance-measurements-part-2/

## 4.5. Security

A problem that may also be faced is security, the tolerance to hacks. For example, another system could bind to a queue in the system and get the messages from the crowd. RabbitMQ system offers the “Exclusive” configuration of the queues. Since the consuming queues of the crowd data are known, these queues can be exclusive. This means that only one known consumers may consume messages.

Task 3.4 (Data Usage Control) deals with the elicitation of security requirements for the RESCUER system and with the specification of more methods to satisfy those requirements.

# 5. Roadmap for Future Work

A concept planned to be investigated in the last iteration is task queues which dispatch messages to several subscribers of the same message topics using a *round robin* approach[16]. This simplifies the design of very scalable distributed systems.

Furthermore, performance tests have to be performed for the Integration Platform as a component (to be reported in D.1.5.3 (Integrated Platform Demonstrator 3)) and for the system as a whole, which is one of the goals of the evaluations to be reported in D5.4.1 (Evaluation Report of the Integrated Solution 1) and later on in D5.4.2 (Evaluation Report of the Integrated Solution 2).

The last iteration of this deliverable will focus on defining the communication flow and characteristics of the group-targeted crowd steering and communication of the emergency situation to the public. In addition, it should also validate whether the definitions made in the second iteration are valid or they have to be readjusted.

[16] https://www.rabbitmq.com/tutorials/tutorial-two-python.html

# 6. Conclusion

The integration middleware has been implemented as an event-driven architecture using RabbitMQ because of its hub-and-spoke topology and the AMQP standard protocol. The type of exchange chosen for the RESCUER system is 'topic' since this allows us to easily define message flows using the routing key structure that was shown in this deliverable.

The current state of the middleware broker takes into account the information flow from the Mobile Crowdsourcing Solution to ERTK to send Incident Reports and Notifications, the follow-up interaction involving ERTK and the Mobile Crowdsourcing Solution, the output from the Data Analysis Components that go to the Context Components.

The different types of follow-up interaction are taken into account to optimize it as much as possible in terms of performance.

# Glossary

## Terms

**Command and Control Centre** Group of people and tools assigned to evaluate risks and make decisions in an emergency and/or crisis in an industrial area or at a large-scale event, usually at the same physical place.

**Communication Infrastructure** Component of the RESCUER platform whose goal is to support the information flow between the crowd and the command centre.

**Data Analysis Solutions** Component of the RESCUER platform whose goals are 1) fusing similar data coming from different eyewitnesses, 2) analysing photos, videos, and text messages in order to extract information such as the type of incident, the position and dimensions of the affected area, people density, surrounding sources of further danger, evacuation routes, and possible approach routes for the formal responders.

**Emergency** Critical situations caused by incidents, natural or man-made, that require measures to be taken immediately to reduce their adverse consequences to life and property.

**Emergency Response Toolkit** Component of the RESCUER platform whose goals are to: 1) get contextual information about the emergency, 2) ask eyewitnesses and formal responders for relevant missing information, 3) give instructions to eyewitnesses, first responders and potentially affected people or companies, and 4) communicate the emergency to the media, public authorities, and the general public in a context-aware way. The emergency response toolkit is meant to be used primarily by the command and control centre staff.

**Mobile Crowdsourcing Solution** Component of the RESCUER platform whose goal is to support eyewitnesses and formal responders in providing the command and control centre with information about an emergency situation, taking into account the different smartphones that might be used and how people interact with smartphones under stress.

## Abbreviations

**EC** European Commission

**RESCUER** Reliable and Smart Crowdsourcing Solution for Emergency and Crisis Management

**MCS** Mobile Crowdsourcing Solution

**ERTK** Emergency Response Toolkit

RESCUER

# Appendix A

## Questionnaire: Information needs and offers of RESCUER partners

**Name of partner:**
**Component(s) to develop:**
**Programming Language(s):**
**Platforms (Hardware, Operating System):**

### Information Needs

1. Which information do you need?
2. Can you give a detailed explanation about this information?
3. What will you use this information for?
4. Do you know who provides this information?

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| *N1* | *People Coordinates* | *The available coordinates from the mobile app running with a latency of 5s.* | *To analyse it and obtain the crowd density data.* | *Mobile Application* |
| *N2* | | | | |

### Information Offers

5. Which information do you offer?
6. Do you know who needs this information?

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| *O1* | *Crowd Density Data* | *JSON data describing the density of people vs coordinates.* | *Emergency Response Toolkit to show it in the map.* |
| *O2* | | | |

RESCUER

# Appendix B

## Questionnaire: Information needs and offers of RESCUER partners

**Name of partner: Fraunhofer, DFKI, MTM**
**Component(s) to develop: RESCUER Mobile Solution**
**Programming Language(s): PhoneGap: JavaScript, HTML5, and CSS3**
**Platforms (Hardware, Operating System): Android, iOS**

### Information Needs

1. Which information do you need?
2. Can you give a detailed explanation about this information?
3. What will you use this information for?
4. Do you know who provides this information?

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| | | | | |
| | | | | |

RESCUER

## Information Offers

5. Which information do you offer?
6. Do you know who needs this information?

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O1.1 | Crowd Density Data | JSON data describing the density of people vs coordinates. | Emergency Response Toolkit to show it in the map. |
| O1.2 | Incident local | Where is the incident? | Emergency Response Toolkit to show it in the map. |
| O1.3 | Incident perimeter | Definition of incident perimeters i.e. critical zone, isolation zone,… | Emergency Response Toolkit to show it in the map. |
| O1.4 | Incident type | What it happened? | Emergency Response Toolkit to show it in a summary board. |
| O1.5 | Incident time | When it happened? | Emergency Response Toolkit to show it in a summary board. |
| O1.6 | Affected people | Who is affected? | Emergency Response Toolkit to show it in a summary board. |
| O1.7 | User profile | who is using the app and consequently providing the information | Emergency Response Toolkit to calculate reliability. |
| O1.8 | Potential hazardous materials | Dangerous materials stored in the incident area. | Emergency Response Toolkit to show it in the map. |
| O1.9 | Integrity of physical infrastructures | Description of damages in buildings, roads… and their identification | Emergency Response Toolkit to show it in a summary board. |
| O1.10 | Traffic Information | The current usage of the road infrastructure. | Emergency Response Toolkit to show it in the map. |

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O1.11 | People movement dynamic | Movement patterns of people in the incident area and neighbourhood | Emergency Response Toolkit to show it in the map. |
| O1.12 | Conducted protective actions | Protective action that have been taken | Emergency Response Toolkit to show it in the map and in a summary board (?). |
| O1.13 | Public safety and security status | Occurrence of violent actions, e.g. robberies, plunder, aggressions | Emergency Response Toolkit to show it in the map and in a summary board (?). |
| O1.14 | Incoming and understanding of messages | Confirmation that instruction were read and understood, especially in the case of instructions for the operational forces in venue or specialists working on the response against HAZMATs | Emergency Response Toolkit to show it in a summary board. |
| O1.15 | Human resources | Information about the usage of human resources for the emergency situation. | Emergency Response Toolkit to show it in a summary board. |
| O1.16 | Equipment | Information about the usage of equipment for the emergency situation. | Emergency Response Toolkit to show it in a summary board. |
| O1.17 | Images/Videos + Meta Data | Images and videos of the incident. This should support to extract anyway the information need described above. | Image/Video Analysis Component. |

We are refining this information need in the data model shown in Figure “Data model for the RESCUER app” of deliverable D2.2.1.

RESCUER

# Questionnaire: Information needs and offers of RESCUER partners

**Name of partner: DFKI (Andreas Poxrucker)**
**Component(s) to develop: Text Analysis Module**
**Programming Language(s): Java**
**Platforms (Hardware, Operating System):**

## Information Needs

1. Which information do you need?
2. Can you give a detailed explanation about this information?
3. What will you use this information for?
4. Do you know who provides this information?

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| N2.1 | (Raw) Text | The textual description of the incident sent by people with the mobile application OR textual description sent with images and videos | To analyse it and obtain semantic annotations for the Emergency Response Toolkit | Mobile Crowdsourcing Solution (Dispatcher component) |

## Information Offers

5. Which information do you offer?
6. Do you know who needs this information?

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O2.1 | Semantic text annotations about incident | (Probably JSON) data structure containing the original text and the semantic annotations covering at least type of incident (what), additional location descriptions (where), and information about affected people (who) | Emergency Response Toolkit |

RESCUER

# Questionnaire: Information needs and offers of RESCUER partners

**Name of partner: USP (University of Sao Paulo)**
**Component(s) to develop: T3.2, modules that form the data analysis system**
**Programming Language(s): C++/Java**
**Platforms (Hardware, Operating System): X86 PCs and X86 high-end servers; Windows**

## Information Needs

1. Which information do you need?
2. Can you give a detailed explanation about this information?
3. What will you use this information for?
4. Do you know who provides this information?

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| N3.1 | Image and video with structured data (e.g. geographic location, time, image and video resolution) that presents fire, smoke and/or explosion. | We will use only the images/videos that present minimum digital quality so that we can process it analytically (Filtering Step) and return the data with annotation of crisis situations and extracted features generated in the analysis module. | We will classify, cluster, match and perform similarity querying with the data to identify crises, and to spot regions of interest that indicate fire, smoke, and/or explosion. | These data must come from T3.1 leaded by UPM. |

## Information Offers

5. Which information do you offer?
6. Do you know who needs this information?

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O3.1 | We offer sets of images/videos geo referred and time contextualized that have the potential of assisting the Command Centre in monitoring the crises. | The images/videos shall be tagged (what, who, where) indicating crisis situations along with details about the fire, smoke, and/or explosion that started the crisis. | Emergency Response Toolkit |

RESCUER

# Questionnaire: Information needs and offers of RESCUER partners

**Name of partner: UPM**
**Component(s) to develop: Video Fusion & Filtering Components, Video Analysis Component**
**Programming Language(s): C++**
**Platforms (Hardware, Operating System):**

- **Two computers for parallelize processes at least.**
  - **Minimum requirements: Intel Core i7, 2.8 GHz, 4GB RAM (using now).**
- **OS: Linux.**

## Information Needs

1. Which information do you need?
2. Can you give a detailed explanation about this information?
3. What will you use this information for?
4. Do you know who provides this information?

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| N4.1 | Location | All data received should be geo-located. | To cluster data based on its location and then filter them. | Mobile Application |
| N4.2 | Video sequences | Video sequences recorded by end-users, whose duration is more than 15 seconds. | To analyse them and extract as much information about the crowd as possible in the Video Analysis Component. | Mobile Application |
| N4.3 | Frame stamps | Timestamps of each frame that compose the video sequence.<br>Alternatively, we could work with the timestamp of the beginning of the recording and the frame rate to obtain a temporal reference for each frame. | To analyse the quantity of motion in the scene for the Video Filtering Component. | Mobile Application |
| N4.4 | Orientation measurements | Orientation measurements during the video recording with the objective to associate the orientation of the mobile to each frame of the video. | To analyse the quantity of motion in the scene for the Video Filtering Component. | Mobile Application |

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| N4.5 | Orientation timestamps | Timestamps of each orientation sample. | To synchronize the temporal references of the frames with the orientation samples in the Video Filtering Component. | Mobile Application |
| N4.6 | Acceleration measurements | Acceleration measurements during the video recording with the objective to associate the acceleration of the mobile to each frame of the video. | To analyse the quantity of motion in the scene for the Video Filtering Component. | Mobile Application |
| N4.7 | Acceleration timestamps | Timestamps of each acceleration sample. | To synchronize the temporal references of the frames with the acceleration samples in the Video Filtering Component. | Mobile Application |

## Information Offers

5. Which information do you offer?
6. Do you know who needs this information?

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O4.1 | Location | All data provided by the Video Analysis Component will be geo-located. | Emergency Response Toolkit to show it in the map. |
| O4.2 | Kind of event | Kind of event that is contained in the video-sequence:<br>• Smoke<br>• Fire<br>• Crowd<br>• Combination. | Emergency Response Toolkit to show it in the map. |
| O4.3 | People by m2 | Estimation of the people/m2 presents in the video sequence. | Emergency Response Toolkit to show it in the map. |

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O4.4 | Risk of congestion | Risk of congestion alarm based on the density temporal analysis that the Video Analysis Component realises. | Emergency Response Toolkit to show it in the map. |
| O4.5 | Expanse velocity | Crowd/Fire/Smoke expanse velocity based on the density temporal analysis that the Video Analysis Component realises. | Emergency Response Toolkit to show it in the map. |

RESCUER

# Questionnaire: Information needs and offers of RESCUER partners

**Name of partner: DFKI**
**Component(s) to develop: DFKI Library / Crowd Density Server**
**Programming Language(s): DFKI Library: Objective-C/Java, Server: Python**
**Platforms (Hardware, Operating System): Server Cluster running Linux (Debian Wheezy AMD 64)**

## Information Needs

1. Which information do you need?
2. Can you give a detailed explanation about this information?
3. What will you use this information for?
4. Do you know who provides this information?

| No. | Information Need | Detailed Description | What do you need it for? | Who offers it? |
|---|---|---|---|---|
| N5.1 | People Coordinates | The available coordinates from the mobile app uploaded once a minute. | To analyse it and obtain the crowd density data. | Sensing Library (built into the mobile application) |
| N5.2 | User ID | The unique ID of each app user. | To be able to assign uploaded data to an anonymous entity and to filter crowd data. | Sensing Library (built into the mobile application) |
| N5.3 | Messages (*) | Messages to be received by the mobile app. | To present the mobile app users with messages from the command and control centre. | Later on: the message queue (which will be fed by the Emergency Response Toolkit)<br>Right now: DFKI message server (fed by a DFKI Web UI) |
| N5.4 | Outgoing Text Data (**) | Everything text based that the app is sending to the command and control centre. | Since the DFKI Library is responsible for enabling the Ad-hoc mode, it will need to route all outgoing text data to make sure that it will be sent out via Ad-hoc during scenarios where the network is down. | Mobile application (after the mobile application created all messages for all analysis components, the message should be handed to the DFKI library, which sends it out). |

*: receiving messages is being handled through the DFKI Library since we are also responsible for switching between normal mode and AdHoc mode which directly affects the way messages are received.

**: due to size limitations, the AdHoc mode won't be able to send out binary data. Hence, the DFKI library will only route text data.

## Information Offers

5. Which information do you offer?
6. Do you know who needs this information?

| No. | Information Offer | Detailed Description | Who may need it? |
|---|---|---|---|
| O5.1 | Crowd Density Data | JSON data describing the density of people vs coordinates. | Emergency Response Toolkit to show it in the map. |
| O5.2 | Textual Data | Messages and other text-based information (like reports) from the mobile app. | Emergency Response Toolkit |

RESCUER

# Appendix C

### Data Format for Crowd Density


```
{
  "max": 7,
  "stats": {
    "active_users": 1788,
    "ts": 1373047440,
    "max_timestamp": 1373241360.0
  },
  "data": [
    {
      "lat": 47.37340013035328,
      "count": 1,
      "lng": 8.542822217576711
    },
    {
      "lat": 47.36695102421234,
      "count": 1,
      "lng": 8.543362421557969
    },
    {
      "lat": 47.36456543677241,
      "count": 1,
      "lng": 8.53699210100671
    },
    {
      "lat": 47.36904095397086,
      "count": 1,
      "lng": 8.543561752893536
    }
]
```