




Data modelling for emergency response

Arta Dilo and Sisi Zlatanova


GISt Report No. 54



Data modelling for emergency response

Arta Dilo and Sisi Zlatanova

GISt Report No. 54



Summary

Emergency response is one of the most demanding phases in disaster management. The fire brigade, paramedics, police and municipality are the organisations involved in the first response to the incident. They coordinate their work based on well-defined policies and procedures, but they also need the most complete and up-to-date information about the incident, which would allow a reliable decision-making.

There is a variety of systems answering the needs of different emergency responders, but they have many drawbacks: the systems are developed for a specific sector; it is difficult to exchange information between systems; the systems offer too much or little information, etc. Several systems have been developed to share information during emergencies but usually they maintain the information that is coming from field operations in an unstructured way.

This report presents a data model for organisation of dynamic data (operational and situational data) for emergency response. The model is developed within the RGI-239 project 'Geographical Data Infrastructure for Disaster Management' (GDI4DM)

ISBN: 978-90-77029-26-8

ISSN: 1569-0245

© 2010

Section GIS technology

OTB Research Institute for Housing, Urban and Mobility Studies
TU Delft

Jaffalaan 9, 2628 BX Delft, the Netherlands

Tel.: +31 (0)15 278 4548; Fax +31 (0)15-278 2745

Websites: <http://www.otb.tudelft.nl>

<Http://www.gdmc.nl>

E-mail: a.dilo@utwente.nl & s.zlatanova@tudelft.nl

All rights reserved. No part of this publication may be reproduced or incorporated into any information retrieval system without written permission from the publisher.

The Section GIS technology accepts no liability for possible damage resulting from the findings of this research or the implementation of recommendations.

This publication is the result of the RGI-239 project 'Geographical Data Infrastructure for Disaster Management' (GDI4DM).

Contents

1	Introduction	9
2	Emergency response in the Netherlands and its information needs.....	11
2.1	Organisation of emergency response	11
2.2	Information needs	14
3	Conceptual and logical data model for ER.....	17
3.1	Database management systems	17
3.2	Conceptual data model	18
3.3	Data sharing between different sectors	21
3.3.1	Data created and used by the fire brigade sector.....	21
3.3.2	Data created and used by the medical assistance sector	25
3.3.3	Data created and used by the police sector	27
3.3.4	Data created and used by the municipality	28
3.4	Oracle database schema.....	30
4	Managing spatiotemporal data	35
4.1	Existing research on spatiotemporal modelling.....	35
4.2	Working with spatiotemporal data in Oracle	37
4.2.1	Declaration and use of temporal data types	37
4.2.2	Storage and retrieval of spatiotemporal data	38
5	Spatiotemporal data analysis.....	41
6	Conclusions and recommendations.....	43
	Bibliography	45
	Appendix A: Oracle scripts creating the database schema with temporal data at attribute level.....	47
	Appendix B: Oracle scripts creating the database schema with temporal data at record level	67

1 Introduction

The first hours after a disaster happens are very chaotic and difficult but perhaps the most important for successfully fighting the consequences, saving human lives and reducing damages in private and public properties (Scholten et al, 2008). Several organisations get immediately involved in the response to a disaster incident, like the fire brigade, paramedics, police and municipality. They have to coordinate their emergency work based on well-defined policies and procedures, as well as the most complete and up-to-date information about an incident, which would allow a reliable decision-making process.

A variety of software systems has been created to help the emergency response (ER) people in their activities (Figure 1.1 and Figure 3.6 show screenshots of such systems). There are though several shortcomings (Diehl and v/d Heide 2005, Zlatanova et al, 2006, Scholten et al 2008). The systems are dedicated to specific emergency situations or emergency response sectors. Exchange of information between the different systems is difficult or not possible. They do not offer all the needed information for the different emergency responders, who have to work by combining digital information provided by such systems with analogue information, e.g. different analogue maps, forms that are filled by hand, etc. A lot of information that is coming from the field operations is stored in an unstructured way, e.g. several files in the system, which makes it problematic for a systemised analysis.

A complete inventory of the information needed during the emergency response is often lacking, as well as a good structure for storing the information. The data structuring would facilitate a fast access to a desired piece of information, as well as the automation of analysis of the information, and its use in the decision-making process. A database system provides for these: an organised way of storing the information, mechanisms that enable fast access, and functionality for the analysis of the information. This report presents a database model for the emergency response information. The model is developed within the Bsik RGI-239 project 'Geographical Data Infrastructure for Disaster Management' (GDI4DM), which aimed at the creation of a spatial data infrastructure to assist the decision-making during an emergency response. Following the user requirements (Snoeren 2006, Diehl et al 2006, Snoeren et al 2007), information needs were identified and translated to a conceptual data model. The data model was implemented in Oracle Spatial (Dilo and Zlatanova, 2008) and the tasks performed by different actors in the emergency response were translated to context-aware services, which are to be accessed via well-designed user interfaces (Scholten et al 2008).

A disaster incident in the Netherlands is managed through processes. Each process has a well-defined objective, which realisation requires certain information and often produces information during its execution. Depending on the type of process, different response units, fire brigade departments, police stations, medical services, and municipalities get involved in the incident. People from these organisations perform their tasks based on assigned roles and responsibilities. The data model presented in this report captures the situational information (Diehl and v/d Heide, 2005), e.g. incident and its effect, and the operational information, e.g. the processes

activated to handle an incident, responsible departments, persons (system users) involved in each process and their roles, measurements, etc. Much of the information produced during emergency response is temporal, i.e. it is changing with time, and we need to keep track of changes. New data types are created for temporal and spatiotemporal information: dynamic counts to store, e.g. number of injured; moving point for, e.g. the position of a vehicle; moving region for, e.g. gas plume. This model is to be used within a system that provides for monitoring and supports the decision-making during emergency response, working on the back of such systems, e.g. Eagle I (see Figure 1.1), to provide for an efficient storage, access, and analysis of the data.

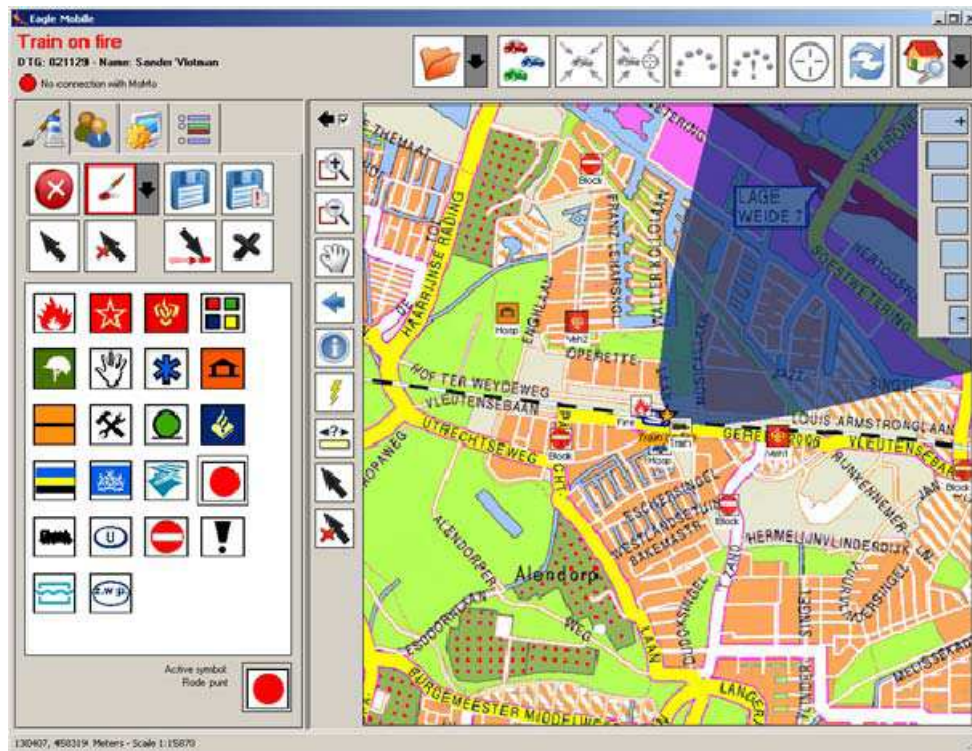


Figure 1.1: Screenshot of Eagle I emergency response system, the Eagle Mobile module (taken from Vlotman and Snoeren, 2009).

The Chapter 2 provides a summary of the emergency response in the Netherlands: the ER sectors and the processes under their responsibility, and the information needs. Chapter 3 discusses the data model. The chapter starts with a short introduction of database management systems (DBMS) and the benefits of using a DBMS. Then, Section 3.2 explains the conceptual model for the emergency response information, followed by schemas that capture the information needed by each emergency sector separately. The conceptual model was implemented in Oracle Spatial, which is the DBMS system that we chose. Section 3.4 describes the data model at this level. A substantial part of the data is (spatio-)temporal, which is not supported by Oracle. Chapter 4 treats in more detail the manipulation of this kind of data, the new data types created and the handling of the temporal data. Chapter 5 provides examples of analysis involving spatiotemporal. Chapter 6 concludes by summarizing the work and directions for future work.

2 Emergency response in the Netherlands and its information needs

Emergency response processes in the Netherlands are legally arranged within the Law for Disasters and Large Accidents (WRZO, Wet rampen en zware ongevallen, wetten.overheid.nl). The document provides definitions, describes responsibilities, manner of working, levels of emergencies, and provides classification of disasters. According to this law, the board of Mayor and Aldermen on the local level is in charge of drawing up a mandatory disaster plan. In this plan, the emergency response activities and the organisational structure should be described. The organisation of the emergency response in the Netherlands is divided into a local level, that is the site of an incident; the regional level, emergency services are regionally organised, supporting several municipalities; the provincial level. Most emergency incidents of a minor nature are responded at the local level. Within this operational structure, the local fire chief has the primary operational responsibility for the on-site coordination of local disaster response. If the magnitude of an incident increases, then a regional coordination team will be formed in liaison with the operational coordination team at site. The regional coordination team is often situated in a regional office remote from the incident. e.g. a joint office of the regional emergency services. If a regional coordination team is formed, then the mayor of the municipality in which the incident is taking place takes the administrative lead. On municipality level, a policy team is formed to support the mayor.

Many more structures can be involved in ‘managing’ when the disaster incident transcends administrative borders e.g. a municipal, provincial or national border. When the potential magnitude of an incident leads to a serious threat to a large section of the community, environment, or property, emergency officers at provincial or national level are informed. If the effects of an incident transcend provincial borders, e.g. a toxic cloud after a nuclear incident, the Ministry of Internal Affairs may take the administrative lead. They will work together with coordination teams at national, provincial, regional and local level to manage and mitigate the disaster.

2.1 Organisation of emergency response

Response and short-term recovery can be categorised into four different clusters, namely, containment and control of the disaster and its effects, medical assistance, public order and traffic management, and taking care of the population. A cluster consists of several processes that are the responsibility of an ER sectors (see Table 2.1), and may involve third parties when needed.

Table 2.1: List of emergency response processes and the responsible sectors.

Containment and control of the disaster and its effects Responsible: Fire Brigade	
1.	Fighting fire and emission of dangerous substances
2.	Rescuing and technical assistance
3.	Decontaminating people and animals

4.	Decontaminating vehicles and infrastructure
5.	Observations and measurements
6.	Alerting the population
7.	Making accessible and clearing up
Medical assistance Responsible: GHOR	
8.	Medical aid chain
9.	Preventative public health and medical/environmental measures
10.	Psycho-social aid and care
Public order and traffic management Responsible: Police and Ministry of Justice	
11.	Clearance and evacuation
12.	Fencing off disaster area
13.	Traffic control
14.	Maintaining the legal order
15.	Identification of fatal casualties
16.	Giving directions
17.	Criminal investigation
Taking care of the population Responsible: Municipality	
18.	Advice and information
19.	Relief and care
20.	Funeral arrangements
21.	Registration of victims
22.	Providing primary needs
23.	Damage registration
24.	Environment protection
25.	Follow-up care

Containment and control of the disaster and its effects: In the Netherlands, the fire brigade is usually organised at a municipal level and has equipment not only for fighting fire, but also for performing various measurements related to release of dangerous substances in the air, water or in the soil. The fire brigade is also responsible for alarming the citizens in case of emergency using the net of stationary sirens. Generally, the fire brigade is obliged to maintain a fire brigade call centre, but the tendency of the last years is to maintain a common call centre, together with the police and GHOR. Usually, the fire brigade duty officer takes the lead in all small-scale emergencies (before the operational team is formed). Several other organisations may also take a part in the containment and control of the hazard and its effect, if the operational organisations (i.e. first responders) need support. For example, in case of flood (a major threat in the Netherlands) Rijkswaterstaat (www.rws.nl), the Dutch National Reserve (www.natres.nl), KNDRD (www.rednet.nl), KNRM (www.knrm.nl) and SAR (www.werkenbijdemarine.nl) can be involved. Some of these institutions, (e.g. KNRM, SAR) follow emergency scaling, which differs from the ones described in WRZO.

Medical assistance: The second large cluster comprises processes related to medical assistance. GHOR (Geneeskundige Hulpverlening bij Ongevallen en Rampen, www.ghor.nl) is an organisation that coordinates medical assistance

during emergencies. Key actors are the Ambulance Central Point (CPA), ambulances, hospitals, and Communal Health organisation, which is responsible for general health issues such as prophylactic medical inspections, vaccinations, etc. Compared to the fire brigade and police, the medical help is quite independently organised and does not directly depend on any local administrations. In case of emergency, however, the GHOR structuring is activated and a regional medical official is appointed who takes the lead within the medical help (similar to the regional officers in fire brigade and police structures). The hospitals are seen as trauma centres, where during disasters mobile medical teams (MMT) should be available 24 hours per day. One hospital is dedicated to the victims of disasters. The SIGMA teams of the Netherlands Red Cross (NRD, www.rodekruis.nl) and special ambulance teams can be formed and included in the medical help operations. NRD is usually involved only in large disasters, which require help and evacuation of many people, such as floods.

Public order and traffic management: In case of an emergency, the police are responsible for processes that are related to evacuation of citizens from affected areas, clear threatened areas, protect shelters and commando centres, control of traffic, etc. In most cases the police are working under the authority of the mayor. In some special cases, e.g. criminal cases, the High Officer of Justice is taking the lead together with the mayor and the regional police chief.

Taking care of the population: Besides the overall responsibility for emergency response (under the authority of the mayor), the municipal structures are responsible for processes related to taking care of the population such as informing citizens, accommodating non-injured people from affected areas, registering casualties, etc. Generally the municipalities have to take care of good preparation of response sectors as well as citizens. Therefore, the municipality has to prepare (and update every 4 years) the emergency response plan. The plan describes the most important types of disaster incidents at the territory of the municipality and the way of dealing with a particular emergency. Responsibilities, tasks and all required medicaments, shelters, reserves of food and clothes, etc. are also part of the emergency response plan.

There are other agreements in the Netherlands concerning organisation and categorisation in emergency response, e.g. categorisation of disaster types, level of emergency (GRIP) (MBZ, 2003) and organisation of the country into safety regions. Some of these are also used by the data model described in Chapter 3 and implemented in the scripts that create the database structure for the emergency response provided in Appendix A: Oracle scripts creating the database schema with temporal data at attribute level.

The emergency types are categorised into 19 types of disaster, which are categories under a larger grouping: incidents in relation to traffic and transport, incidents with dangerous material, incidents in relation to public health, incidents in relation to infrastructure, incidents in relation to population, natural disasters. Disaster types under the first group are Aviation incident, Incident on water, Traffic incident on land; similarly the other groups consist of one or more disaster types. There are five GRIP levels describing the severity of an incident. More details about these

categorisations can be seen in Appendix A: Oracle scripts creating the database schema with temporal data at attribute level, page 50.

2.2 Information needs

The information needed for emergency response is grouped into two large clusters, dynamic information (situational and operational) and static (existing) information. Data collected during a disaster incident are denoted as dynamic data, while the information existing prior the disaster is named static information. Examples of dynamic information are:

Incident: location, nature, scale

Effects: affected area and its development in time, sectoral (a diagram for first estimates of effected areas), and gas plume

Consequences: threatened area (+time/period), escalation possibility

Damages: damaged objects, damaged infrastructure

Casualties: dead, injured, missing, trapped people and animals

Accessibility: building entrances, in- and out-routes, traffic direction, blocked roads

Temporary centres: places for accommodating people (and animals), relief centres, morgues

Decontamination: decontamination centres; vehicles, houses and infrastructure to decontaminate; people and animals to decontaminate

Meteorological information: wind direction, humidity, temperature

Specific information depending on type of disaster: e.g. in case of flood – velocity and water depth, flood pattern; in case of incident with ships – ship type, numbers of people on board, owner, other ships in the surroundings; aircraft incident – type of plane, function (cargo /military /civilian), number of people on board, type of fuel and volume.

Dynamic information is collected from processes of one cluster (i.e. from actors responsible for the process) and is intended to be shared with other clusters/actors. This information can be parameters of incident such as scale, development, which are updated regularly; number of victims considering different categories such as trapped, injured people, slightly wounded, death, missing, which are also updated regularly; or a measurement. In case of detection of dangerous substances in the air, water or in the ground, special measurement teams are sent to collect samples – results of the measurements are reported to the commando and control centre and analysed by a specialist. Some dynamic information has to be gathered from other organisations. For example, the actual metrological information is obtained from the nearest meteorological station, water levels and the likelihood of a flood are obtained from Rijkswaterstaat. The information used during disaster management is very wide, and of a very different nature. The model that is described in this report is restricted to information collected by the first emergency responders.

The most commonly used static information needed by the emergency response is:

Reference data: topographic maps, aerial photographs, cadastral maps and data

Managerial and administrative data: census data, administrative borders, risk objects (gas stations, storage places of dangerous goods, etc.), vulnerable objects (schools, nursing homes, etc.)

Infrastructure: road network, water network, utility networks (gas, water, electricity), parking places, dykes, etc.

Buildings catalogues: high/low-rise material, number of floors, usage (residential, industrial), presence of dangerous materials, owners, cables and pipes, etc.

Accessibility maps: for buildings, industrial terrains, etc.

Water sources: fire hydrants, uncovered water, drilled water well, capacity, etc.

The existing information is available at various places: within the municipalities (reference data), private companies (utility networks), ministries (buildings catalogues, cadastre, road and water networks), accessibility maps (fire brigade), etc. This information is assumed available and accessible directly from the source. Information models for access and exchange of data are in process of development or readily available within the NEN (NEN, 2005). This report does not cover existing data.

3 Conceptual and logical data model for ER

This chapter provides a short overview on database systems, followed by the main topic, the data model for the emergency response. Data modelling is done in two levels:

- a conceptual level that describes the information in terms of classes/entities, their attributes and their interrelations, and is independent of a specific DBMS system;
- a logical level that is the translation of the conceptual model into a database schema, which holds the specifics of the implementation of the conceptual model to an Oracle Spatial database.

The Unified Modelling Language (UML, version 2.1) was employed for modelling data, and Enterprise Architect (EA) was the modelling tool. The translation from the conceptual level to the logical level is done partially automatic from inside EA, with additional modifications. These are also explained in the corresponding sections.

3.1 Database management systems

A database management system (DBMS) is a software package that allows a user to set up, use, and maintain a database (de By, 2005). A database is a repository of interrelated data items that are often central to the business of an enterprise or institution. Many diverse applications and multiple users, each of which may need only a fraction of the data, generally use a database. One role of the database is to provide a single representation to all these applications, avoiding redundancies and possible inconsistencies that would occur if each application managed its data separately.

A DBMS provides to applications a high-level data model and a related query and data manipulation language. Other important functionalities offered by a DBMS are indexing and join methods, authorisation, integrity constraints, concurrency, and transactions. Important elements of a data modelling language are a collection of types together with their operators (functions). They are used by the data manipulation and query language, which is offered to applications for the storage and analysis of their data. The query optimiser uses the indices for efficient performance of queries.

The classical database management systems were conceived for relatively simple business applications. For example, the data types available for attributes are simple, basically integers, floating-point numbers, or short text strings. One goal of database research in the last decades has been to widen the scope, so that as much as possible any kind of data used by any application can be managed within a DBMS, described by a high-level data model, and accessed by a powerful query language. For example, we would like to store images, geographic maps, music, videos, CAD models and so on. For all these kind of data, we are interested in appropriate extensions of data model and query language, so that any kind of question about these data can be formulated in a manner as simple as possible and be answered efficiently by the DBMS (Güting and Schneider, 2005). A spatial DBMS is an extensions of standard

(i.e. classical) DBMS's. A spatial DBMS is extended with data structures and algorithms for computation over spatial types (points, lines, polygons and volumes) together with spatial indexing techniques, and extension of the optimiser for mapping from the query language to the spatial components. Most of the spatial DBMS support spatial data types according to the simple feature specifications for SQL (www.opengeospatial.org). The spatial databases may have several data structures for management of spatial information: geometry, topology or network (Oosterom et al. 2005).

Today, there are several commercial and open source spatial DBMS systems. A critical question for an emergency response application is the selection of the DBMS, as well as the choice between commercial and open source. Important aspects to be considered are the support for different data types, appropriate for handling different kind of information coming during an emergency, as well as extendibility with new types and functions. The information collected during an emergency is of very different nature. Besides various sensor information such as optical and range images (terrestrial, aerial), videos, textual data, audio, etc will have to be managed. Most of the information is dynamic; therefore the temporal component is critical. The choice between open source or commercial DBMS is driven from pragmatic reasons. On the one hand, an open source, freeware DBMS, e.g. PostGIS, may have benefits in large area devastating disasters (similar to East Asia Tsunami or the hurricane Katrina) when existing infrastructure is destroyed and a command centre has to be set up in few hours. On the other hand, many organisations in the Netherlands have already commercial DBMS, such as Oracle Spatial. Oracle Spatial and PostGIS offer the most from the functionality we need, but within the project RGI-239 (GDI4DM) we decided to use Oracle Spatial. Many units from the emergency sector, e.g. municipalities, have already their data in Oracle Spatial.

3.2 Conceptual data model

According to the Dutch procedure for emergency management, a critical situation that needs special attention and treats the wellbeing of humans is called *incident*. An incident could be hypothetical, e.g. a forthcoming big concert or important football match, or a real incident, e.g. a big explosion or a plane crash. Usually, when a real incident happens, a kind of emergency call of a report about the incident comes to a commando centre (via the emergency number 112). The emergency response units from the closest location get involved in order to manage the incident. Based on the type of the incident, several processes are activated, each process being responsibility of one or more departments, dependent on the scale of the incident. Several people get involved in these processes having specific roles. Also, several teams can be formed in order to perform specific tasks. When the incident involves release of dangerous substances, a template, named *sectormal*, is used to sketch the zone affected by gas distribution. Several measurement teams are formed and sent in the field to perform measurements, from which the movement of gas plume is derived. An incident usually causes damage in buildings, cars, infrastructure, as well as people and animals living in the surroundings of the incident. In case there are casualties in people, detailed information is collected from the medical assistance, and they are sent to relief centres.

The conceptual model shown in Figure 3.1 captures classes of information and their associations. It is a UML class diagram; attributes and operators are hidden for the

sake of space. A class is drawn as a box, and an association is drawn as an arrow connecting two classes; an association that has attributes has a box attached, which contains the attributes. Multiplicities of an association are shown when different from 1, thus a missing label indicates a multiplicity equal to 1. Dashed lines show dependency, which in our case means a source class which existence depends on the target class.

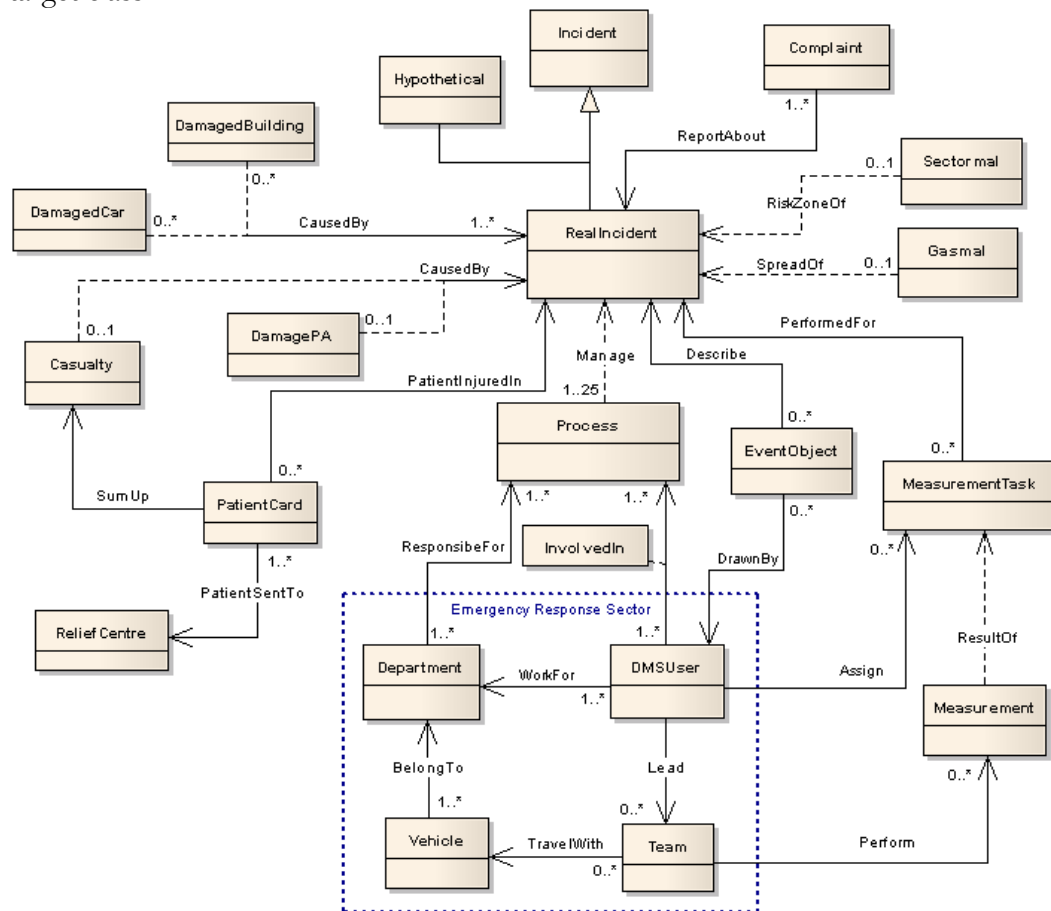


Figure 3.1: Conceptual level of dynamic data for emergency response: classes in boxes, and their associations shown by arrows, together with a box in case an association has attributes.

The classes **Hypothetical** and **RealIncident** are subclasses of **Incident**. Reported complaints (generally from citizens) are presented by the class **Complaint**. The association **ReportAbout** connects complaints to the **RealIncidents** for which they are made. Several complaints can arrive for an incident, as shown from the cardinality of **ReportAbout** association. The class **Sectormal** contains information about sectors that will possibly be affected by an incident involving dangerous substances. This is done by marking circle sectors in a fixed template (see Figure 3.3). **Gasmal** contains information about the gas plume for the incident. It is computed from the measurements performed by the measurements teams. The link between the measurements and the corresponding gas plume is not explicitly stored but can be derived by the time stamps.

Figure 3.2 is a part of the class diagram of Figure 3.1 that contains the above mentioned classes, and shows the attributes of these classes. A detailed explanation of attributes of all the classes is provided further, in Section 3.3.

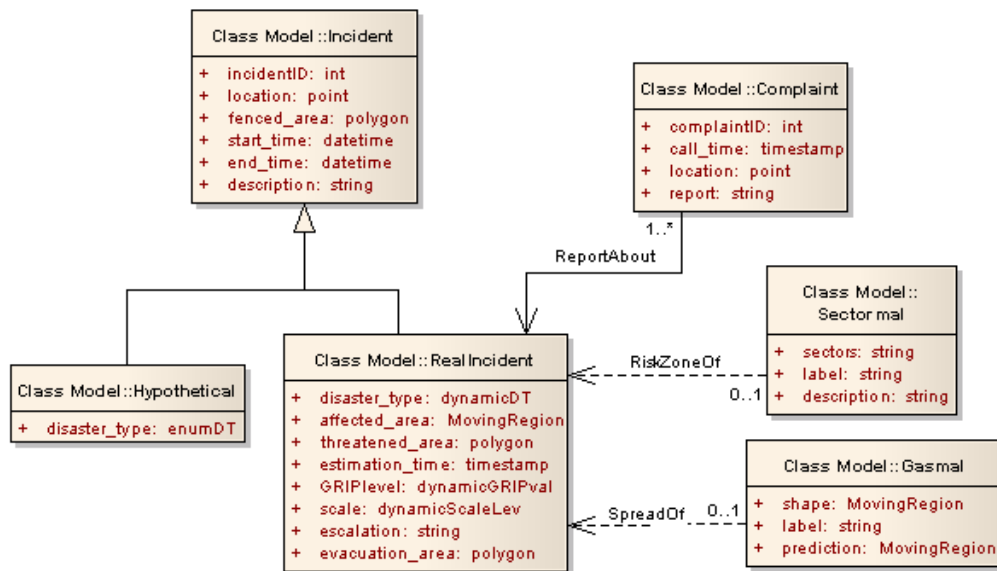


Figure 3.2: Part of the conceptual model showing classes and their attributes.

A **RealIncident** is **Managed** by one or more **Processes**; at most 25 processes will be activated for an incident (see Table 2.1). Class **Department** contains information about a department unit. A department is responsible for several processes started for the same incident or processes of different incidents. When the scale of an incident is big, several department units take the responsibility over the process. Association **ResponsibleFor** keeps track of the responsible departments for each process. A department owns one or more vehicles, e.g. a fire brigade owns trucks and boats. Class **Vehicle** keeps information about vehicles, and **BelongTo** takes care of the ownership. Class **DMSUser** contains information about the system users, i.e. emergency response actors that are users of the ER application system. A system user is involved in different processes of one or different incidents at different times. The association **InvolvedIn** contains the duration of such involvements. During the response to a disaster incident, several teams are created with people from the ER sectors, and volunteers assigned to these sectors (e.g. fire brigade). The class **Team** keeps information about teams, e.g. number of its members, and position of the team. We assume a team is created for an incident (or a task), and has a one-time existence. Not all persons from a team are necessarily users of the system, but there is one person who is a team leader and has access to the system. The association **Lead** indicates the team member that is a system user. A team uses a vehicle to travel to the place of incident, which is captured by **TravelWith** association.

Different measurements are performed for incidents that involve dangerous substances. A measurement task is designed by an advisor of dangerous substances (AGS), and sent to a team that performs the measurement according to task specifications. The class **MeasurementTask** keeps information about such task, the association **PerformedFor** keeps track of the incident for which a task was designed, and the association **Assign** keeps track of which (AGS) user assigned what task. The class **Measurement** contains results of the measurements and the association **Perform** records which team performed what measurement. A gas plume is derived from a calculation based on several measurements. The class **EventObject** contains drawings done by system users to locate different events happening in the field, e.g. a gas leak, blocked road, damaged utility (network) segment. The information represented by this class is of a very different nature. The association **DrawnBy** keeps track of objects drawn by each user. (This is an example of unstructured information,

which has to be further categorised and organised after a better understanding of the needs, probably into different classes.)

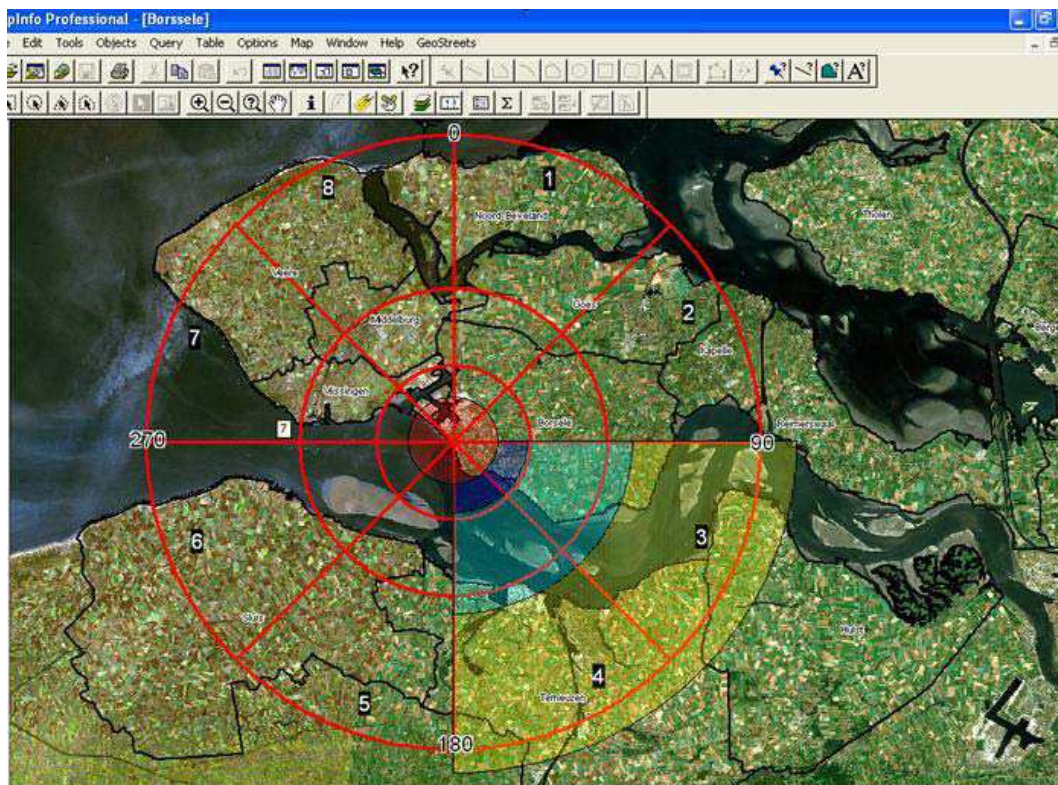


Figure 3.3: Example of a sectoral template overlaid with an orthophoto, used in the Borsele workshop for nuclear disasters (taken from Diehl et al. 2006)

The classes **DamagedBuilding**, **DamagedCar**, and **DamagePA** contain information about damaged buildings, damaged cars, and damages in people and animals, respectively. Information about individual persons that are injured during an incident is kept by the class **PatientCard** as a record of treatment, symptoms, etc. as well as the level of injury. The class **Casualty** contains summaries of injured persons categorised on the level of injury, and class **ReliefCentre** contains information about centres where the injured people are sent.

3.3 Data sharing between different sectors

The emergency response information is collected by the different ER sectors; most of this information is important for all the sectors, thus should be shared between them. The data model presented in this report deals with this common information, which is of importance for all actors. This section explains what part of the data (model) is used by each sector.

3.3.1 Data created and used by the fire brigade sector

The fire brigade departments are responsible for processes 1–7 (see Table 2.1). The attribute **process_type** of class **Process** defines which process is it from the list of 25 processes. The attribute takes values from an enumeration list **enumDMProcess**. The other attributes of class **Process** are the (start) time the process is activated, and its ending.

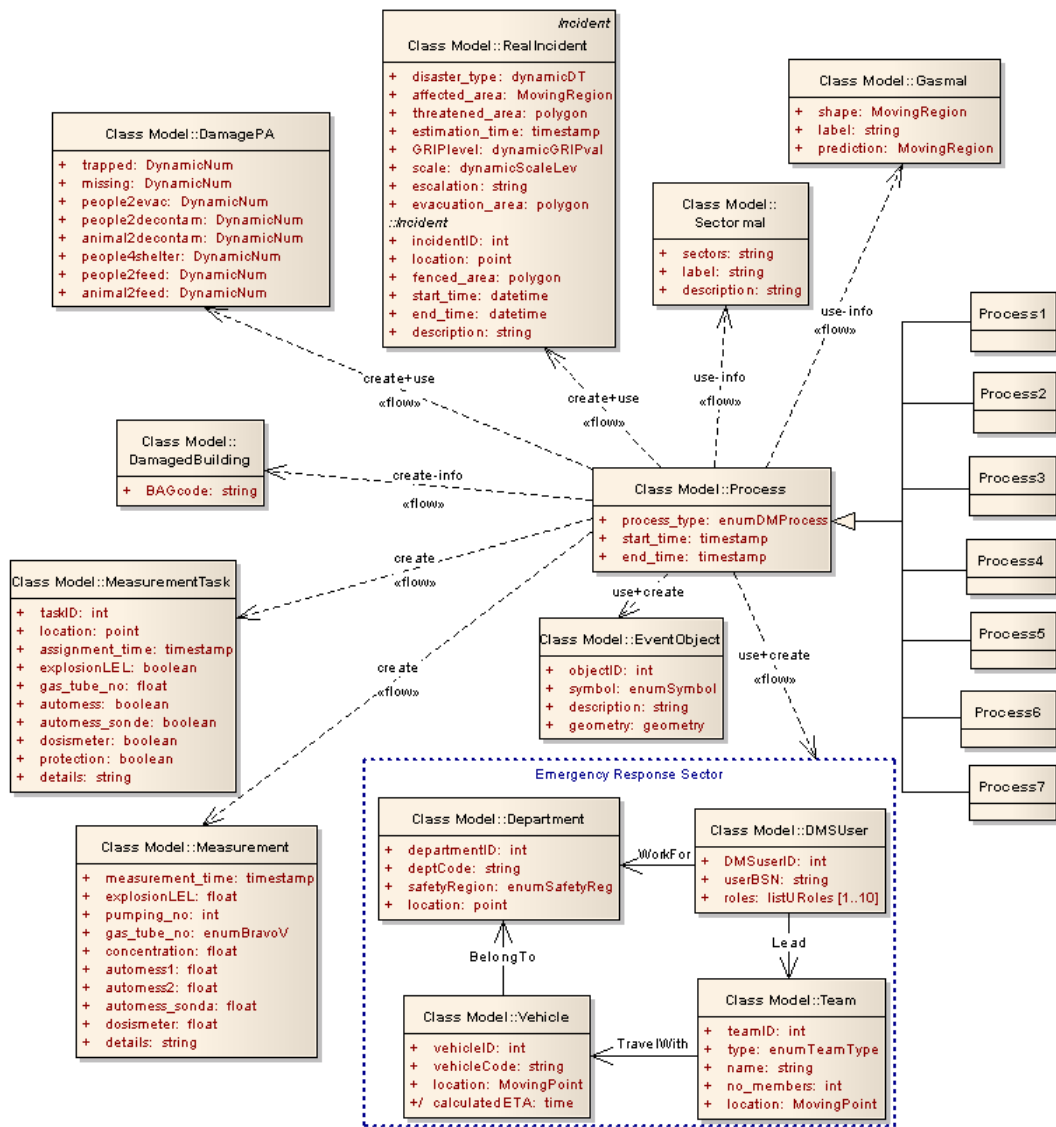


Figure 3.4: Information created and used by the fire brigade departments.

The information about **RealIncident** is expressed in the attributes: the disaster type of the incident (one of the 19 types listed in Section 2.1), which may change during the incident, the affected area by the incident, which is a dynamic area, i.e. it changes in time, an estimation of the threatened area together with the estimation time, GRIP level and scale of the incident, both (possibly) changing in time, the escalation risk as word description, and the area to be evacuated. The **RealIncident** information is mostly created by the fire brigade. The information about **DamagedBuilding** consists of a code, **BAGcode**, that uniquely identifies the building and can be used to access additional information from existing datasets.

Measurements are performed by the fire brigade. Figure 3.5 shows an example of a completed measurement form. The upper part of the form is the task formulated by an advisor who decides what measurements should be performed. The bottom part (gray-shaded) contains the results of the measurements. Classes **MeasurementTask** and **Measurement** reflect the information of the upper and bottom part of the form, respectively. The **MeasurementTask** contains an identifier for the task (and the

measurement itself), location where the measurement should be performed, the time of this task assignment, a yes/no value for a set of parameters indicating if a parameter should be measured, and a text field to give more details about the task. The **Measurement** contains the results of a measurement task: for each parameter that is requested to be measured by the task there should be a filled value in the corresponding attribute in the **Measurement**. It also contains the time of the measurement and a text field to provide more details about the accomplished measurement.

(fill / cross out)

Measurement task from leader-AGS to DTG: _____

Measurement team **39/41Whiskey** **Whiskey 745**

Sectorenmal

Coordinates place of incident **Oscar 1** **51°59'10" 5°54'17"**
 sector[s] **Oscar 2** **a1, b2, c3**

Measurement location number / coordinate **Mike** **51°59'11" 5°54'20"**

Type of measurement

Explosion state LEL **Echo** **yes**
 Gas-tube number **Bravo** **12.54**
 Automess **Romeo** **yes**
 automess + sonde **Romeo-sierra** **no**
 persoonlijke dosismeter **Delta** **no**

Personal protection **Ademlucht** **yes**

Details _____

Time of assignment **Tango** **10/10/2006 7:38 PM**

Measurement report to leader-MPO from (for simultaneous p.p.)

Measurement team **39 Whiskey** **Whiskey 745**

Measurement location number / coordinate **Mike** **51°59'11" 5°54'20"**

Measurement results

aflezing explosiemeter in %LEL **Echo** **23.56**
 gasmeetbuisje nummer **Bravo** **1.2**
 aantal pompstagen en concentratie **november** **5** **Charlie** **20.3**
 aflezing automess in µGy per uur **Romeo 1** **10.2**
 aflezing automess in mGy per uur **Romeo 2** **5.25**
 aflezing automess+sonde in s⁻¹ **Sierra** **-**
 aflezing dosismeter in mGy **Delta** **[-. -]**

Time of measurement **Tango** **10/10/2006 9:28 PM**

Details **foggy**

Figure 3.5: The form containing the measurement task and results.

The damages in people and animals are collected by the fire brigade department, GHOR and the police. The class **DamagePA** contains this information: number of trapped persons, number of missing persons, number of people to evacuate, people and animals to decontaminate, number of persons needing a shelter, people and animals needing food. All these are dynamic counts, meaning that they change in time and we want to keep track of the history of change. The class **EventObject** contains an identifier for every object that is drawn on the screen, a symbol code selected from the list of symbols that is offered by the ER application, and is recorded in the list of values of the **enumSymbol** type, a text description for the drawn object, and the shape of the object, which could be a point, a line or a polygon. Figure 3.6 shows the palette of symbols in the VNet emergency response

system. People from the fire brigade departments or from any other sector can create the event objects.

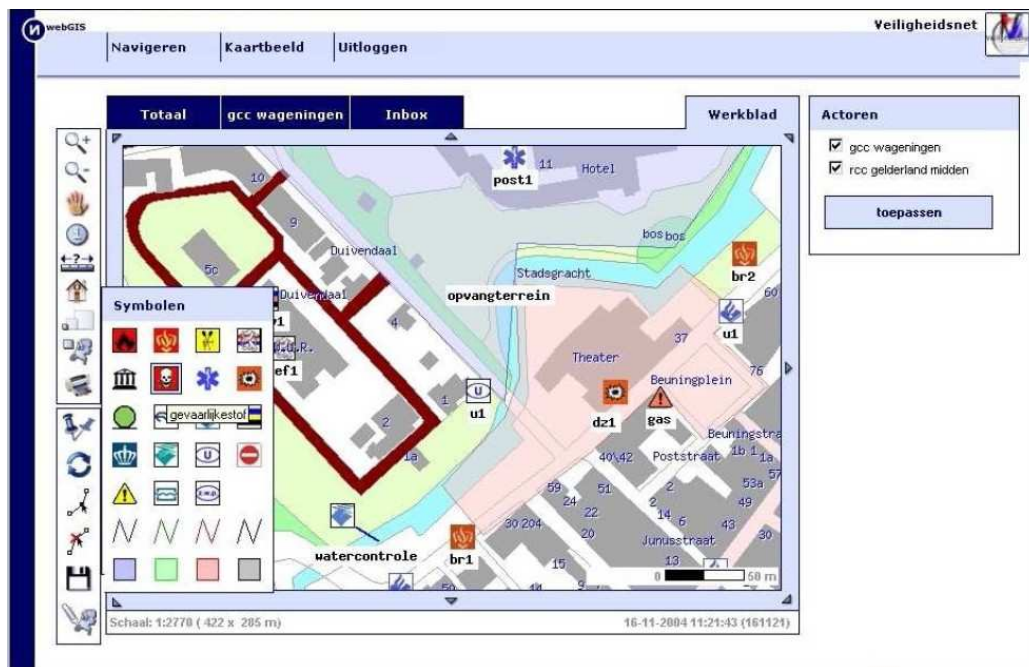


Figure 3.6: Screenshot of VNet system showing the drawing symbols palette.

The group of classes belonging to the Emergency Response sector is an example of the combination between existing and dynamic information. Departments, vehicles, and people from the ER sector belong to existing information. For each of these three classes we keep minimal information, basically a code to make the link between the operational data and the existing data. Dynamic information in this category is the involvement in the processes managing an incident. The class **Department** contains: an identifier for each emergency response unit within the ER system; the department code to connect to external (existing) databases with full information about a department; the safety region the department belongs to; the location as x-y(-z) coordinates that are stored in this system for fast access (although they may be collected from an existing database with information about a specific department). The association **ResponsibleFor** is updated for the participation of a department in an ER process. The class **DMSUser** contains: an identifier for a system user; the social security number of the person in order to be able to get additional information from existing databases; a list of roles this emergency responder takes during the response to the incident. The association **InvolvedIn** keeps track of the involvement of a system user in the processes managing the incident: the start and end time of the involvement and his/her role in this process.

The class **Vehicle** contains: an identifier for the vehicle (car, boat, motorbike or truck) within the ER system; a code to connect to existing databases; the location of the vehicle that is a dynamic position often collected from a GPS, captured by **MovingPoint** type of data; (calculation of) the estimated time of arrival in case the vehicle is requested to go to a destination. The class **Team** is completely dynamic, meaning a team is created during the management of an incident and stops existing after the incident. It contains: an identifier for each team; the type of the team, defined by its purpose, e.g. measurement team, which takes values from a pre-defined list **enumTeamType**; a name for the team, which is for the ease of use during the incident response; the number of team members; the location, which is a

dynamic position that could be defined by the vehicle the team travels with, or should be recorded separately in case the team moves freely (no vehicle). Not all the members of a team are necessarily users of the ER system, but there is always one person that has access to the system in order to receive or send information. This is captured by the **Lead** association.

The **Sectormal** and **Gasmal** information are used by the fire brigade departments. The class **Sectormal** contains information about the sectors, a label and a description. Class **Gasmal** contains information about the gas plume: the shape of the gas plume, which changes in time, a description, and the prediction for the gas plume after a fixed time interval. We keep track of the history of predictions, thus the attribute is of type **MovingRegion** as it is the gas plume shape itself.

3.3.2 Data created and used by the medical assistance sector

The medical assistance (GHOR) sector is responsible for processes 8–10 (see Table 2.1 and Figure 3.8). The information about **RealIncident** and **Sectormal** are also used by the medical assistance people. The interaction with the group of classes of the Emergency Response sector is the same as for the fire brigade sector. The three processes controlled by the medical assistance sector produce the information of **PatientCard**, **Casualty**, and **ReliefCentre** classes.

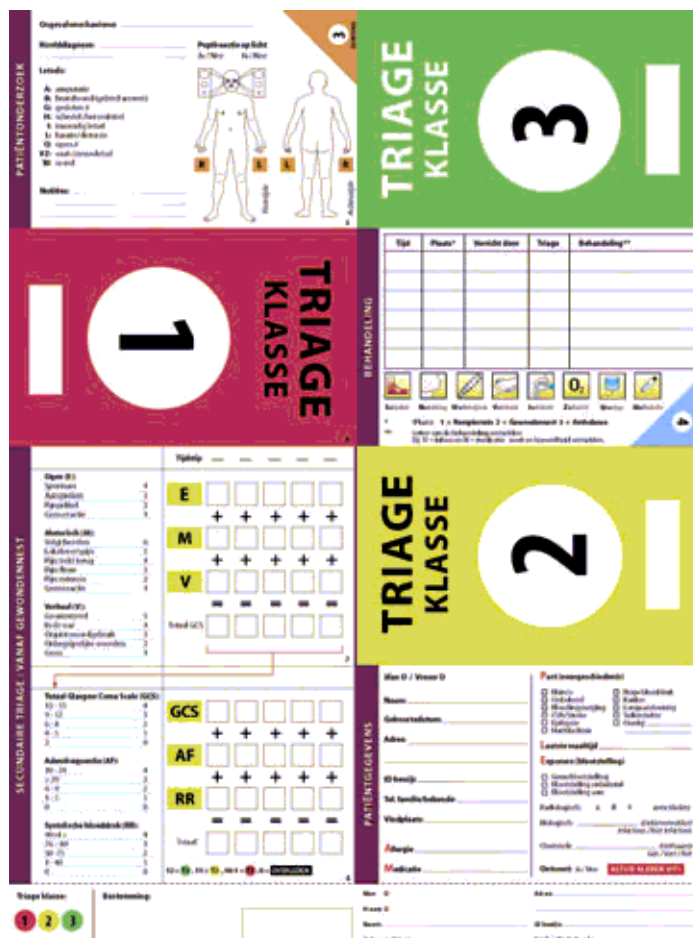


Figure 3.7: Patient Card.

The class **PatientCard** holds the information of the (analogue) patient card shown in Figure 3.7. The information collected in the patient card is complex, but well-structured. We created new data types to group together pieces of information that

are related to each other, e.g. medical history, and the new types are used for the attributes of class **PatientCard**, which contains the complete information of the analogue patient card. The class contains an identifier for the patient, personal information about the patient: gender, name, birth date, address, family phone number. The code (ID) used by the analogue patient card is also kept in **PatientCard**. Other information is the place the person was found, allergies in case (s)he has, medication that was given. The medical history consists of a group of yes/no answer about important medical problems, e.g. heart or blood pressure problems (see Figure 3.7 top-left).

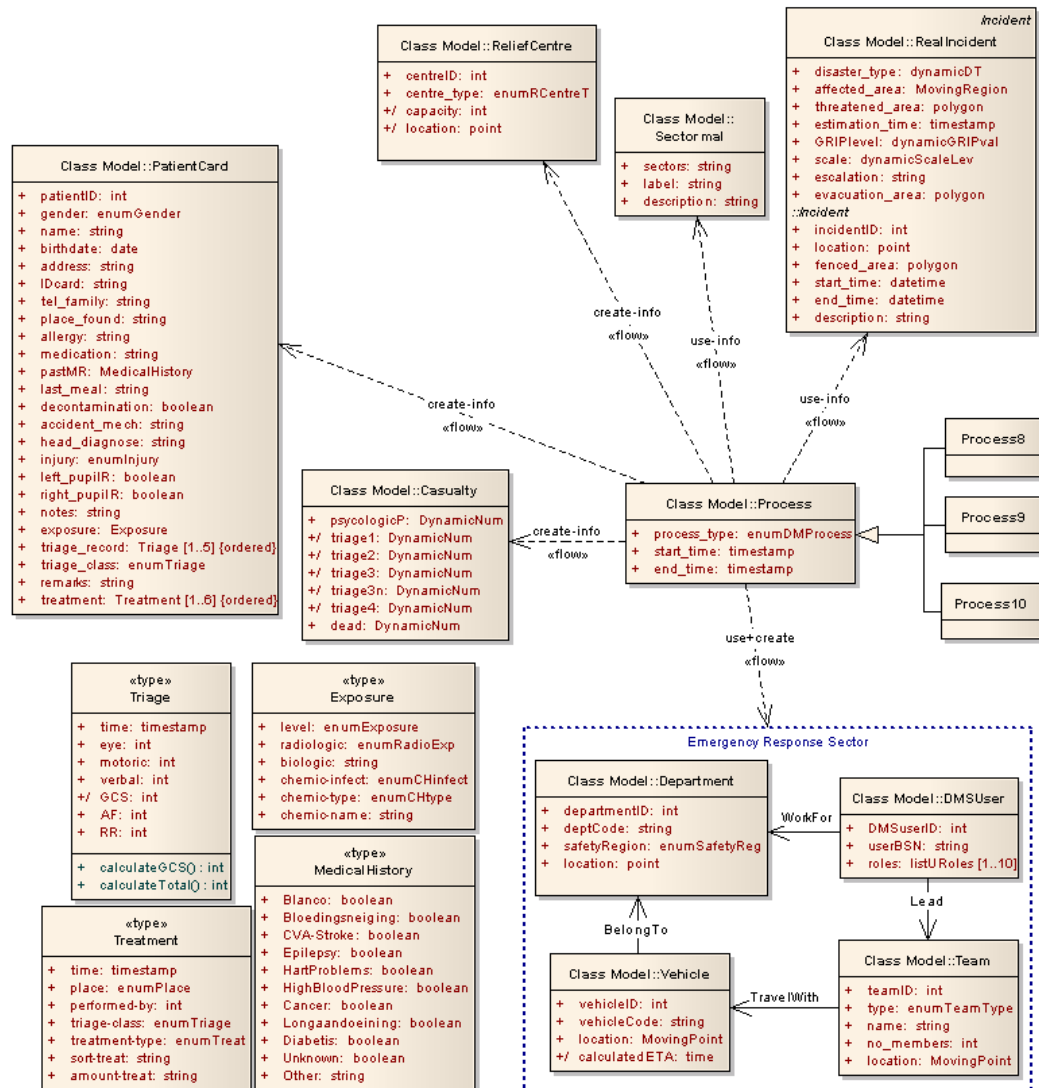


Figure 3.8: Information created and used by the GHOR departments.

A new data type, **MedicalHistory**, is created for this collection, and the data is put in the attribute **pastMR**. Other information is the last meal the person has taken, if there was decontamination, accident mechanism, head diagnose, type of injury from a list of predefined values **enumInjury**, problems with left or right pupil as a yes/no answer (**boolean** data type), and additional notes for the observed problems. Exposure to chemical substances or radiation (see Figure 3.7 bottom-right) is recorded as a group of checks taking values from pre-defined lists (**enum** types). A new type, **Exposure**, is created for this group of values. A categorisation called 'triage' defines the priority levels of handling a patient, **T1–T4** (GHOR 2008). Triage is a dynamic process. A group of medical checks are performed for a patient in a

repeated manner at different places, e.g. at the incident, in the nest of the first help, at the hospital, or after different treatments that are given to him/her. A new data type, **Triage**, is created to group together the medical checks, while the attribute **triage_record** keeps track of the repeated triage calculations. The attribute **triage_class** holds the value of the latest calculation of the triage class. Another data type, **Treatment**, is created for a treatment given to a patient containing the time and place of the treatment, the type of treatment (a predefined list stored in **enumTreat**), etc. The attribute **treatment** holds the records of several treatments (1–6, see Figure 3.7 top-right) given to a patient, and **remark** contains additional remarks.

The class **Casualty** contains summaries of triage classes, **T1 – T4**, number of patients classified in these triage levels, which are changing in time. These attributes of class **Casualty** would be updated after any relevant change in the **PatientCard** class, e.g. entering a new patient, change of **triage class** value. The other attributes of class **Casualty** store the number of people with psychological problems and deaths. The class **ReliefCentre** contains: an identifier for the centre, what kind of relief centre, e.g. hospital, field hospital, other public buildings used as shelters, and the current capacity and the location of the centre.

3.3.3 Data created and used by the police sector

The police sector is responsible for processes 11–17 (see Table 2.1). The information about the **Sectormal** and the **Gasmal** is used by the police departments (to ensure public safety and to control the risk for the emergency responders). The seven processes controlled by the police sector use and modify the group of classes of the Emergency Response sector in the same way as the previous processes (of the other sectors). A part of the incident information and information of the **EventObject** class is created by the police sector, e.g. the fenced area of an incident, or the blocked roads. The class **DamagedCar** contains the plate number of the cars that are damaged by an incident.

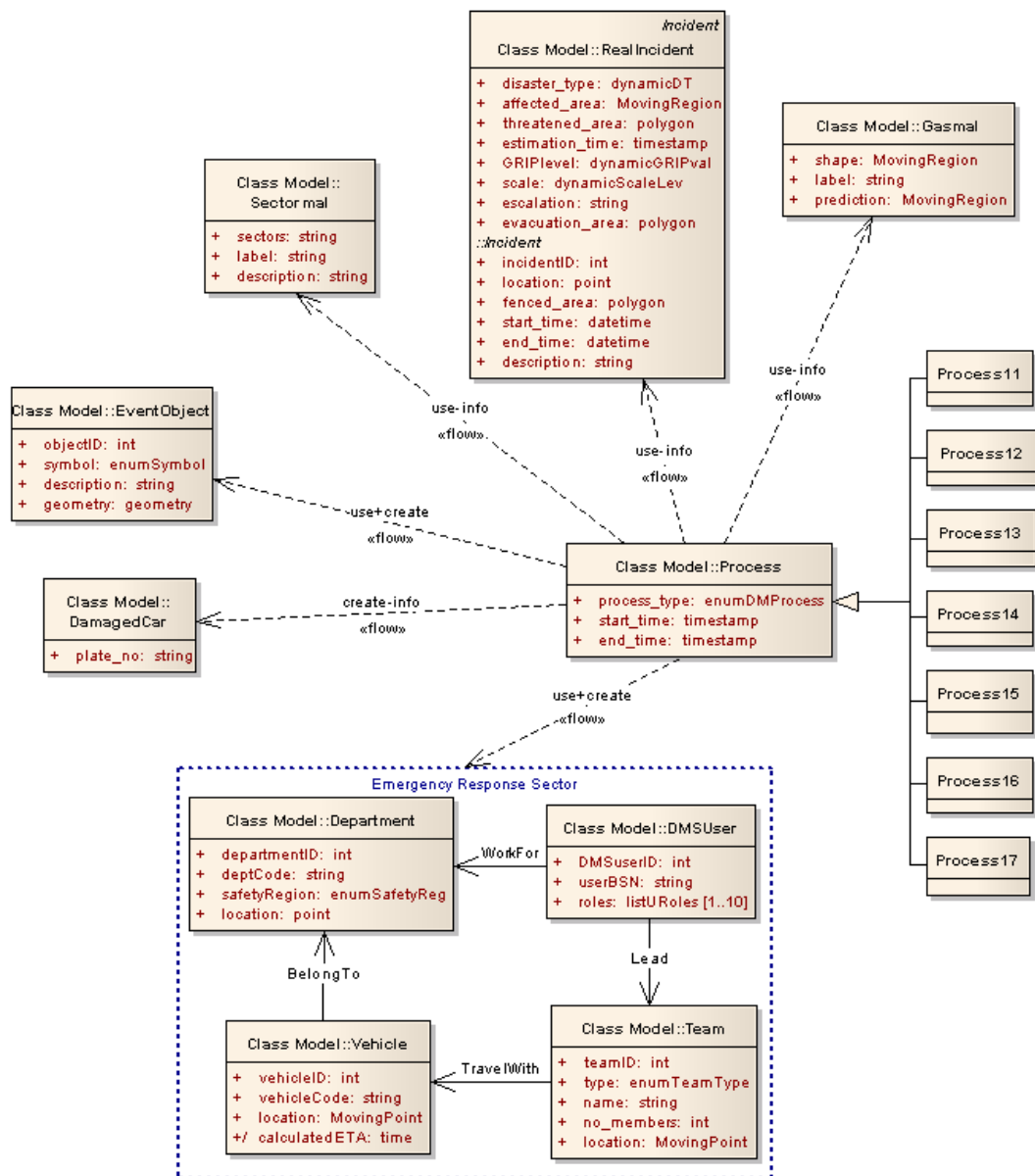


Figure 3.9: Information created and used by the police departments.

3.3.4 Data created and used by the municipality

The municipality is responsible for processes 18–25 (see Table 2.1), which are related to alarming, giving help and registrations of injuries and damages. Therefore, the classes in the model that are related to damages and casualties are mostly used. These classes are already explained in the previous sections. The processes 18–25 create/modify and use information from these classes. Figure 3.10 illustrates which classes are relevant for the municipality processes. The information of **Gasmal** is needed if dangerous gas (substance) is released. Municipality creates some information from **EventObject**. For example, they draw polygons to indicate areas to be used as distributions centres, areas to be used as field morgues for people and animals, and they use the information about blocked roads and streets as provided by the police.

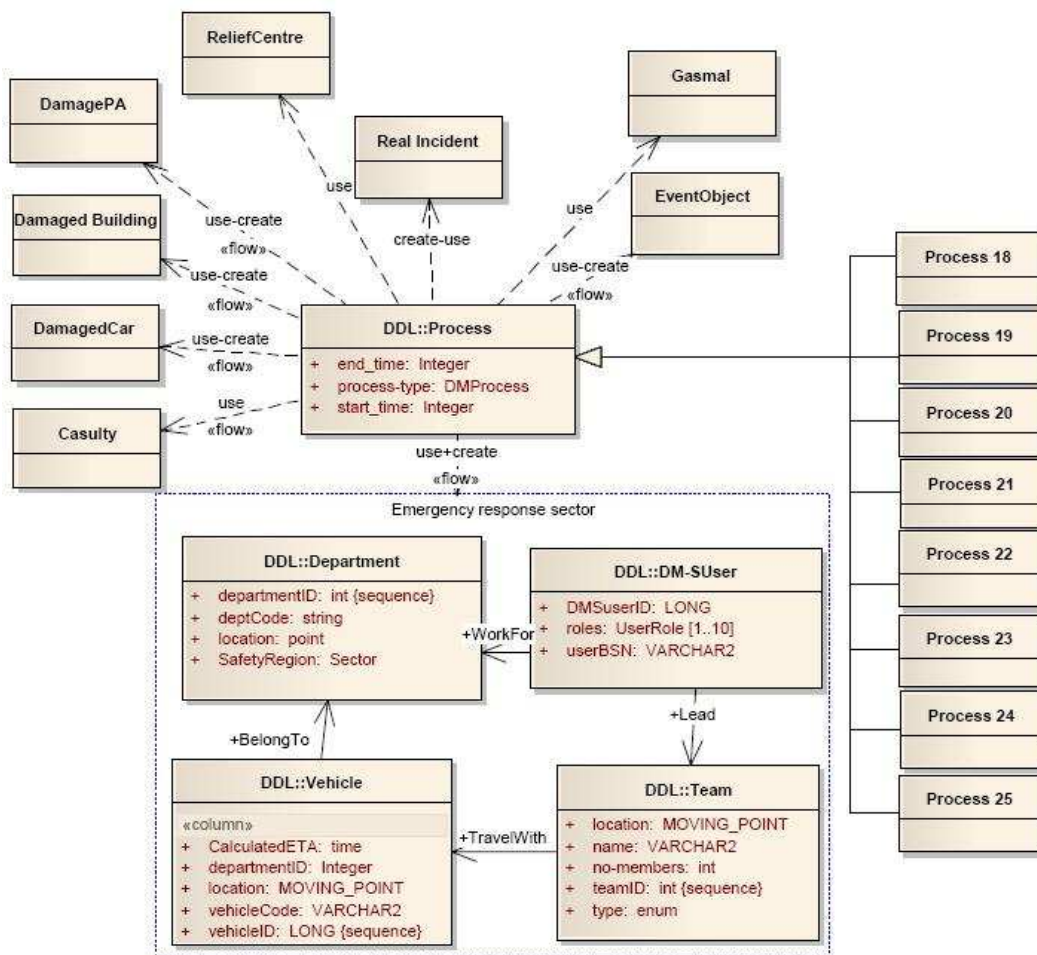


Figure 3.10: Information created and used by municipality.

It should be noted that not all the processes the municipality is responsible for are modelled in detail. For example, process 19 is taking care of people (and pets) who need help in evacuation (such as old, ill or disabled people), deciding where temporal shelters have to be placed, how much food is needed, etc. This process considers all the needs for a maximum of three days. Process 20 deals with funeral arrangements and is activated in case of a large number of dead people and animals, which have to be transported to places for cremation of funerals. Process 21 is specifically devoted to the registration of victims. The information needed for this process is available in **DamagePA**, **ReliefCentre** and **Casulty**. Process 22 deals with situations when people need care for longer periods. The information that has to be maintained is the location of the shelters, the capacity of each shelter and the number of people (with specification of their needs). Process 23 deals with registration of damages and destructions (usually in large disasters). In general, when this process is to be started, a special center (CRAS, Centraal Registratiebureau Aangerichte Schade is established). The work of the centre is quite complex, requiring an overall view on the damages on buildings, infrastructure, cars, data (e.g. cadastre data and other registers), claims from citizens and companies etc. Parts of this information can be found in **DamagecCar** and **DamagedBuilding**, but we expect that the model needs to be extended to serve Process 23 better. Processes 24 and 25 are not modeled as well, since they are very specific and depend on the type of disasters. Furthermore many new actors may get involved (especially in Process 25, follow-up care), which do not belong to the sectors considered in this model. Furthermore it is not quite clear which information from these processes is meant to be shared with other emergency response sectors.

relationship with **DMSUser** and **Process**, while a new table **JoinDMSUserToProcess** is created from the association between **DMSUser** and **Process**. The automatic translation adds a primary key attribute named after the table name and followed by 'ID' in case there is no such attribute; e.g. the attribute **eventObjectID** is added to table **EventObject**, while we have meant **objectID** to be the primary key of the table. In addition to these, several data types in our conceptual model require special treatment, e.g. boolean, enumeration types, the spatial types, as well as the new data types that were created.

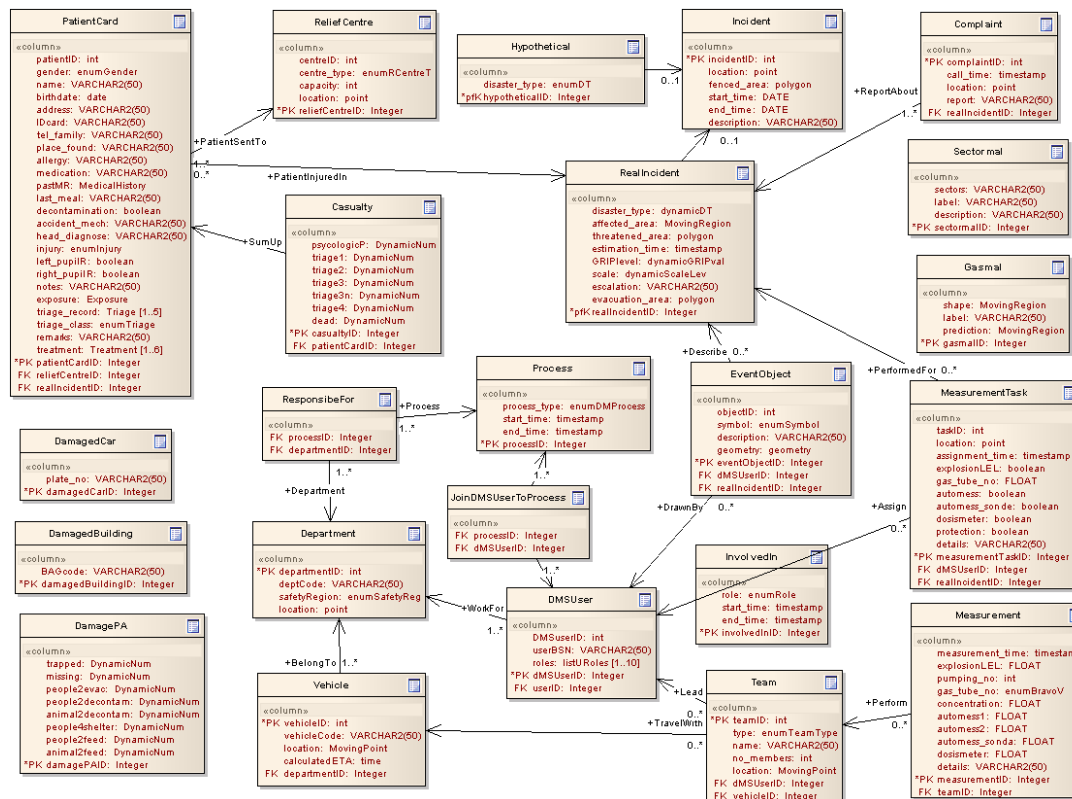


Figure 3.12: EA automatically generated tables from the class model: PK indicates a primary key, FK a foreign key, pFK a foreign key that is also a primary key; an arrow indicates tables relationship.

Figure 3.13 shows tables and relationships after making the necessary modifications: types **int** and **integer** are changed to **NUMBER()**, the types **date** and **timestamp** are changed to **DATE** and **TIMESTAMP**, respectively. We changed the spatial types to **MDSYS.SDO_GEOMETRY**, which is the spatial type of Oracle Spatial for all points, lines, and polygons. The boolean types are changed to **CHAR(1)**, with a check constraint added to the field to restrict values to 'Y' (true) and 'N' (false), e.g. for the attribute **decontamination** of the table **PatientCard** the constraint is **CHECK (decontamination in ('Y', 'N'))**. The enumeration types are changed to **NUMBER()**, and these numeric values will be used as codes that refer to the values of the enumeration list. The relationship tables **InvolvedIn** and **JoinDMSUserToProcess** are merged under the name **UserInProcess**. The relationship table **ResponsibleFor** is renamed to **DeptResponsibe4Proc**. The relationships are added for the dependent tables, e.g. **Process**. The primary key attributes added from the automatic generation were deleted and the identifier (class) attributes were declared as primary keys, e.g. **objectID** is the primary key of table **EventObject** instead of **eventObjectID**, which is removed from the table. The primary keys were added in the depended tables. Then, the primary-foreign key relationships were added or corrected. We merged the

generation of DDL statements can also be done automatically from EA; a screenshot is shown in Figure 3.14.

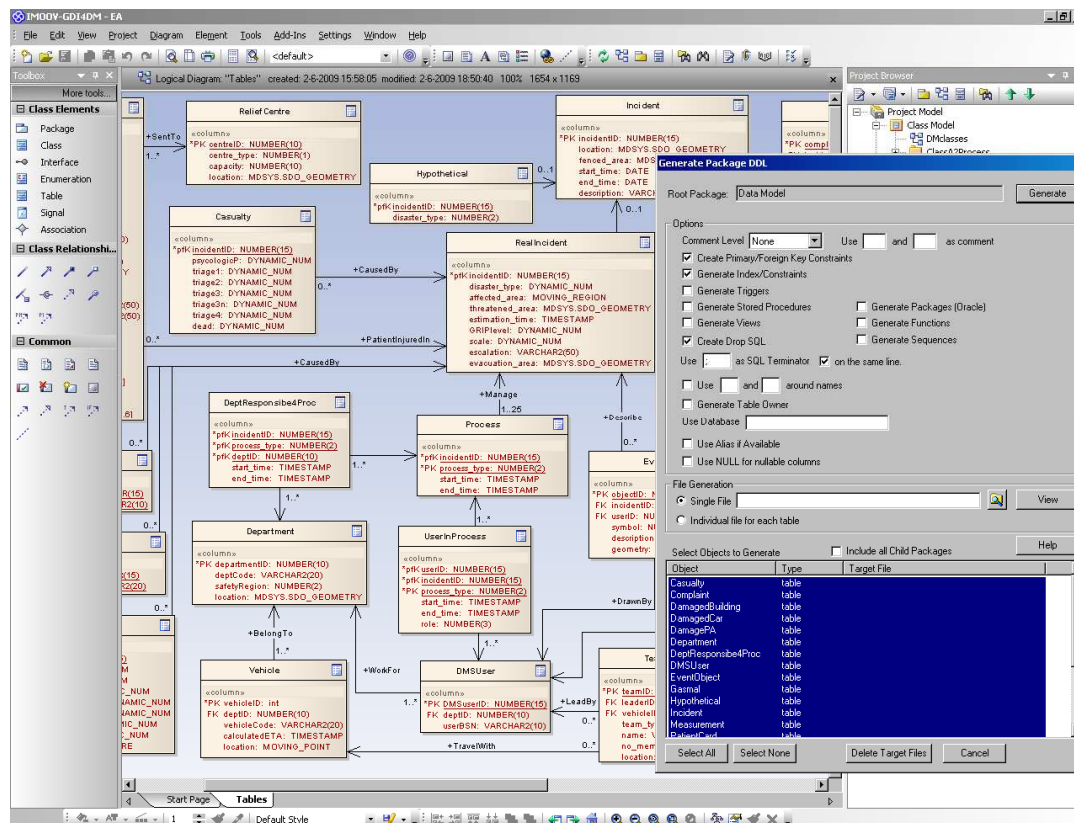


Figure 3.14: Automatic generation of SQL scripts from the data tables.

The DDL script created by EA needs modifications. The automatic generation cannot handle new data types, neither a solution for enumeration lists. We added most of the constraints directly in the DDL scripts (and not in the UML table model, for not being practical, though possible). We created new data types for the patient card information, e.g. the Exposure data type is created by

```
CREATE TYPE EXPOSURE AS OBJECT (
    level NUMBER(1),
    radiologic NUMBER(1),
    biologic VARCHAR2(25),
    chemic_infect NUMBER(1),
    chemic_type NUMBER(1),
    chemic_name VARCHAR2(25)
);
```

The creation of temporal and spatiotemporal data types is treated in Chapter 4. All the enum types are implemented as (look-up) tables in Oracle. For example, the attribute process_type takes values from a look-up table Process_Type containing information for all processes. The table Process_Type is created from:

```
CREATE TABLE Process_Type (
    code NUMBER(2) CONSTRAINT PK_Process_Type PRIMARY KEY,
    value_EN VARCHAR2(70),
    value_NL VARCHAR2(70)
);
```

The table is filled with information for all the 25 processes. Values of process_type attribute are constrained to codes available in Process_Type table, by its declaration inside the Process table:

```

CREATE TABLE Process (
    incidentID NUMBER(15) CONSTRAINT FK_Process_Incident
    REFERENCES RealIncident,
    Process_type NUMBER(2) CONSTRAINT REF_Process
    REFERENCES Process_Type,
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    CONSTRAINT PK_Process PRIMARY KEY
    (incidentID, process type)
);

```

The automatic generation of DDL scripts creates the primary and foreign key constraints with an ALTER TABLE statement after the creation of all the tables. After modifying the DDL scripts we put the constraints within the CREATE TABLE statement.

Other constraints are added to control attributes of boolean type (Oracle tables do not support boolean type), as well as (pre-)conditions of the information. For example, explosion state LEL is a measurement from the form of Figure 3.5: attribute `task_exposionLEL` contains values true/false deciding if this measurement should be performed, and attribute `explosionLEL` contains the measurement value in case the task decided that this measurement should be performed. This is checked through these constraints written in the declaration of Measurement table:

```

task_exposionLEL CHAR(1) CONSTRAINT CHK_explosion_task
CHECK (task_exposionLEL in ('Y', 'N')),
...
CONSTRAINT CHK_explosion CHECK
(task_exposionLEL <> 'Y' OR explosionLEL IS NOT NULL),

```

The first declares `task_exposionLEL` attribute as having only values 'Y' (true) and 'N' (false); the second assures that if `task_exposionLEL` is set to true, then `explosionLEL` is not empty. Appendix A: Oracle scripts creating the database schema with temporal data at attribute level contains the complete scripts that create the database schema.

4 Managing spatiotemporal data

The databases managed by a standard DBMS normally describe the current state of the world. A standard DBMS offers data types like **date** and **time** that can be used from the attributes. If an application needs to keep track of the history of changes, it has to manage time itself by adding it explicitly as attribute(s), and performing the right kind of computation in the queries. When a join is done between two tables extended by time attributes, explicit conditions should be added to the query to assure concurrency in the lifetime of joined tuples. This results soon in quite complicated queries, and long execution times. A temporal DBMS system takes care that such conditions are checked automatically, so that there is no need to include them explicitly in a query. The objective of a temporal DBMS system is the integration of temporal concepts deeply into its data model and query language, to achieve efficient execution of queries. A spatiotemporal database system aims at a combination of temporal and spatial concepts, which bring to structures and techniques for handling spatiotemporal data.

4.1 Existing research on spatiotemporal modelling

The basic concepts of a temporal DBMS are the time domain and the time dimensions. Time is generally perceived as a one-dimensional space extending from the past to the future. The time space can be viewed as *bounded* or *infinite*. A bounded model assumes some origin and also an end of time. Time can be seen as *discrete* or *continuous*. While time is perceived as continuous, for practical reasons temporal databases work with discrete time. Two important time dimensions are *valid time* and *transaction time*. The valid time refers to the real world time instant when a change occurs, or the period during which a fact is valid. The transaction time refers to the time when the change is reflected in the database, or the period during which the database is in a particular state. In this context, standard databases are called *snapshot databases*; those dealing with valid time only are called valid-time or *historical databases*; those handling only transaction time are called transaction-time or *rollback databases*; and those treating both kinds of time are called *bitemporal databases*. The term *temporal database* refers to a model or system offering any kind of time support.¹

Some relational and object-oriented databases are extended with the temporal concepts. The general approach has been to consider elements of the DBMS data model (e.g. tuples) as *facts* and to associate elements of the time domain with them to describe when facts are valid (*timestamps*). Timestamps are added at the tuple (object) level, or at the attribute level. Several temporal models have been proposed (Zaniolo et al 1997); some of the models store change at the instant it occurs, others store the period during which a fact or a database state exists. For example, a temporal model working with valid time at the tuple level may add a new tuple for each change, time-stamping it with the instant it became valid, or time-stamping every tuple with the period they are valid.

¹ Most of the text in this section is taken from (Güting and Schneider, 2005).

Classical research on spatiotemporal databases has focused on discrete changes: sporadic events e.g. volcano eruptions, earthquakes; stepwise constant changes, that are spatial objects whose shape and position changes discretely in time, e.g. capital of a country or headquarter of a company change position discretely, land parcels in cadastral applications or state boundaries change shape discretely. Figure 4.1 (taken from Güting and Schneider, 2005) illustrates discrete and continuous change of spatial objects. The point and the region in Figure 4.1(a) change at discrete moments of time, whereas the point and region of Figure 4.1(b) change continuously.

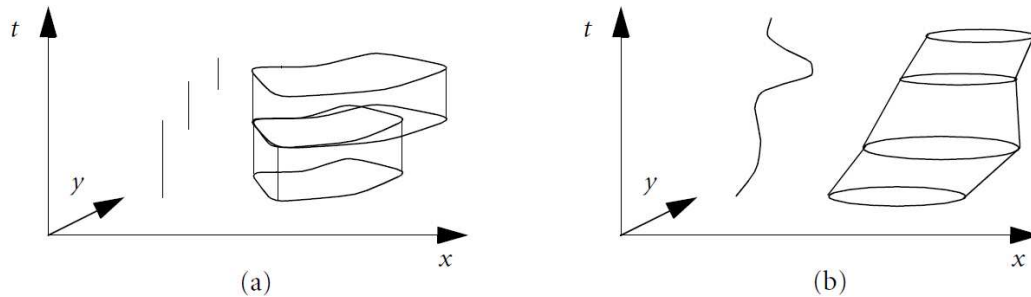


Figure 4.1: Spatial objects changing over time: (a) discretely changing point and region, (b) continuously changing point and region.

Later research concentrates on continuous changes (Güting et al. 2000, Güting et al. 2003, Güting and Schneider 2005) and uses the term moving objects. A moving point is the basic abstraction of a physical object moving around in the plane or a higher-dimensional space, for which only the position, but not the extent, is relevant. The moving region abstraction describes an entity in the plane that changes its position as well as its extent and shape, i.e. a moving region may not only move but also grow and shrink.

In general, a moving object can be represented as a partial function from the time domain to the set of spatial objects. For example, a moving point is an element of the set $mPoint$ defined as $mPoint \equiv \{o : R \rightarrow Point\}$. All other ‘moving’ types are defined similarly, and are named by prefixing the argument type with an ‘m’: $mPoint$, $mLine$, $mRegion$, $mInt$, etc. collection of operators is defined over the spatial types. These operators should be able to express the most common questions one may ask about changing objects in time. One group of operators performs questions in the time domain, another group of operators extends with time the (static) spatial operators (see Güting 1994 for a list of standard spatial operators), a third group gives the rate of change of moving objects. The operator **DefTime** returns the time intervals during which a moving object exists. This is the domain of the function representing the $mSpatial$ object. The operator **AtInstant** returns the state of a $mSpatial$ object at a given instant. The **AtInstant** operator provides snapshots of a moving object as a pair of (static) object state and instant of time. The operators **Initial** and **Final** return, respectively, the state of a moving object at the first and last instant of its existence, respectively. The operator **Val** extracts the object from a pair of object state and time. The operator **Present** returns true if a moving object exists at a given time instant, and returns false otherwise. A collection of static (spatial) operators can be extended in the time domain through a process called lifting (Güting et al. 2000, Güting et al. 2003). The idea is to allow any argument of a spatial operator to be made spatiotemporal, i.e. to become a changing object, and to return a temporal type. For example, we can perform the intersection between two moving regions, or between a moving region and a region, both returning a moving region. A

region object is constant and exists all the time. It can thus be presented as a moving region object that is a constant total function. All lifted operators are defined for moving objects, or a combination of moving objects with static objects. Lifted operators are defined for set operators (e.g. union, intersection), topological predicates (e.g. overlap, disjoint), metric operators (e.g. area, perimeter, distance). A third group of operators contains **Derivative**, **Speed**, **Turn**, **Velocity**.

Other work on spatiotemporal data modelling and querying are presented on (Chomiski and Revesz 1999, Tryfona and Christian 1999, Oosterom et al. 2002, Pernt et al. 2006, Meratnia, 2005, Mokbel and Aref 2008)

A moving object is presented by a (partial) function from the time domain to a spatial object type. This assumes that the changing process is fully known and modelled, which however is generally not the case. Spatial objects are extracted from acquired data, which is done at discrete moments of time. A more realistic solution is to store an object at various discrete moments of validity, together with functions that model the change (transformation) from a stored instant to the consecutive one. Modelling change between short periods of time is less prone to errors, and can be accepted as a better approximation of the real change. A moving object can thus be represented by a sequence of objects at discrete moments of valid time, together with a function for each of these moments, giving the change from that moment to the next one.

A moving object can be stored as a sequence of the corresponding static simple objects. The complete (approximate) information about a changing object is then compiled from a linear interpolation method. An **mPoint** object is a (time) sequence of **Point** objects, each associated with a time stamp, i.e. $\langle ((x_1, y_1), t_1), \dots, ((x_n, y_n), t_n) \rangle$. Linear interpolation should be performed between two consecutive elements of the sequence in order to get the state of the **mPoint** object at any moment of time. An **mPoint** object is thus a piecewise linear feature in a four-dimensional space.

4.2 Working with spatiotemporal data in Oracle

The data model described in Chapter 3 requires three temporal and spatiotemporal data types: **DYNAMIC_NUM** for dynamics counts, e.g. number of missing people; **MOVING_POINT** for dynamic points, e.g. position of a team in the field; **MOVING_REGION** for dynamic regions, e.g. the gas plume. A dynamic count is stored as a sequence of pairs (cnt_i, t_i) , where cnt_i is the count value at instance t_i . For any time instant t , the value for count can be calculated from linear interpolation between two consecutive counts cnt_i and cnt_{i+1} , such that $t \in [t_i, t_{i+1}]$. Similarly, **MOVING_POINT** is stored as a sequence of pairs point location and time, and a **MOVING_REGION** as a sequence of pairs polygon shape and time. Different interpolation techniques can be used for calculating point position and polygon shape for any moment of time, see e.g. (Meratnia and de By 2003, Meratnia 2005) for moving point interpolation.

4.2.1 Declaration and use of temporal data types

We use nested tables in Oracle to store sequences for dynamic types. For example, to create **MOVING_POINT** type, first an object type is created containing a time

instance and a point location, which is then used to build the sequence as a table of such objects:

```
CREATE TYPE MPointInst AS OBJECT (
    meas_time TIMESTAMP,
    point_geo MDSYS.SDO GEOMETRY
);
/
CREATE TYPE MOVING_POINT AS TABLE OF MPointInst;
/
```

The other types, MOVING_REGION and DYNAMIC_NUM are created in a similar way. The new data types are used in tables containing attributes of a dynamic nature. For example, Team table has an attribute position, which is a moving point. The DDL statement for creating the Team table is:

```
CREATE TABLE Team (
    teamID NUMBER(20) CONSTRAINT PK_Team PRIMARY KEY,
    name VARCHAR2(20),
    team_type NUMBER(2) REFERENCES TeamType,
    no_members NUMBER(2),
    leaderID NUMBER(15) REFERENCES DM_SUser,
    vehicleID NUMBER(15) REFERENCES Vehicle,
    position MOVING_POINT
)
NESTED TABLE position STORE AS TeamPosition;
```

Other tables, like RealIncident, Gasmal, Casualty, and DamagePA, which contain dynamic attributes, have similar declaration (see Appendix A, from page 57, for the declaration of the other tables).

The use of temporal and spatiotemporal types in the database tables is an example of handling time at the attribute level. Another option is to work with time at the tuple level. In this case the management of data is performed by standard SQL (Structured Query Language) queries. The performance of queries over a database schema that uses spatiotemporal data types, as compared to a database schema that handles time at tuple level is a relevant matter. For this reason we created a database schema, i.e. DDL scripts to create the tables and other structures in Oracle Spatial. Appendix B provides the scripts for the creation of a database that handles time at the tuple level.

Working with (spatio)temporal data that is stored by employing temporal and spatiotemporal types, requires special queries, i.e. not the standard SQL. In the following section we elaborate on these queries.

4.2.2 Storage and retrieval of spatiotemporal data

A spatiotemporal attribute can be filled by adding values (point location or polygon shape) for each time instant one by one. First, an empty table should be created for the spatiotemporal attribute, and then time-geometry pairs can be added one by one. For example, assume a new team is created, and its data is entered into Team table, together with an empty instance for the spatiotemporal attribute position:

```
INSERT INTO Team (teamID, name, team_type, no_members,
    position)
VALUES(12, 'Whisky 349', 2, 4, MOVING_POINT());
```

Once the empty nested table is created, data can be added to it:

```
INSERT INTO TABLE(
  SELECT t.position
  FROM Team t
  WHERE TeamID = 12)
VALUES(sysdate, sdo_geometry(2001, 90112,
  sdo_point_type(86875.2, 447457.9, null), null, null));
```

The above statement adds one tuple, time-position pair, where time is the current system time, and position is point type geometry of Oracle Spatial. It is possible to enter a block of values at once in a spatiotemporal attribute. We have the GPS logs for the position of (six) teams in the field, containing (among other data) an identifier for the team, time and position. The data is entered in a temporary table GPSTrack:

```
desc GPSTrack;
Name                Null?              Type
-----            -
ID                  NUMBER(11)
TIME_               CHAR(19)
GEOMETRY            MDSYS.SDO_GEOMETRY
```

To insert the data from this flat table into the Team table in our model, e.g. for team with ID = 16, the following statement is executed:

```
INSERT INTO Team(teamID, position)
VALUES(16,CAST(
  MULTISET(SELECT MPointInst(time_, geometry)
  FROM GPSTrack
  WHERE id = 16)
  AS MOVING_POINT )
);
```

Each individual record of a spatiotemporal attribute can be accessed by performing an un-nesting of the table. For example, the following statement returns the trajectory of team 16 that was entered above.

```
SELECT tp.meas_time, tp.point_geo
FROM THE(SELECT position
  FROM Team
  WHERE teamID = 16) tp;
```

For the purpose of visualisation, spatiotemporal data is converted to spatial data. We create views that perform an un-nesting of tables, in order to turn the spatiotemporal types to spatial types. For example, to view data from Team table, TeamTracking view is created:

```
CREATE VIEW TeamTracking AS
SELECT t.teamID, t.team_type, t.name, t.no_members,
t.vehicleID, p.*
FROM Team t, TABLE(t.position) p;
```

which structure is

```
desc TeamTracking;
Name                Null?              Type
-----            -
TEAMID              NOT NULL          NUMBER(20)
TEAM_TYPE           NUMBER(2)
NAME                VARCHAR2(20)
NO_MEMBERS          NUMBER(2)
VEHICLEID           NUMBER(15)
MEAS_TIME           TIMESTAMP(0)
POINT_GEO           SDO_GEOMETRY()
```

To visualise data from Oracle Spatial, metadata should be filled for spatial attributes. For the view created above we add metadata information for the spatial attribute, specifying the extent and the reference system (for the whole Netherlands):

```
INSERT INTO USER_SDO_GEOM_METADATA VALUES
('TEAMTRACKING',
 'POINT_GEO',
 SDO_DIM_ARRAY(
   SDO_DIM_ELEMENT ('X', 0, 220000, 0.05),
   SDO_DIM_ELEMENT ('Y', 250000, 630000, 0.05)),
 90112);
```

Data of TeamTracking view can be visualised from FME software (or another software visualising Oracle Spatial data), and can also be accessed from a web service. Figure 4.2 shows the trajectories of six teams, which data was entered by the statements given above.

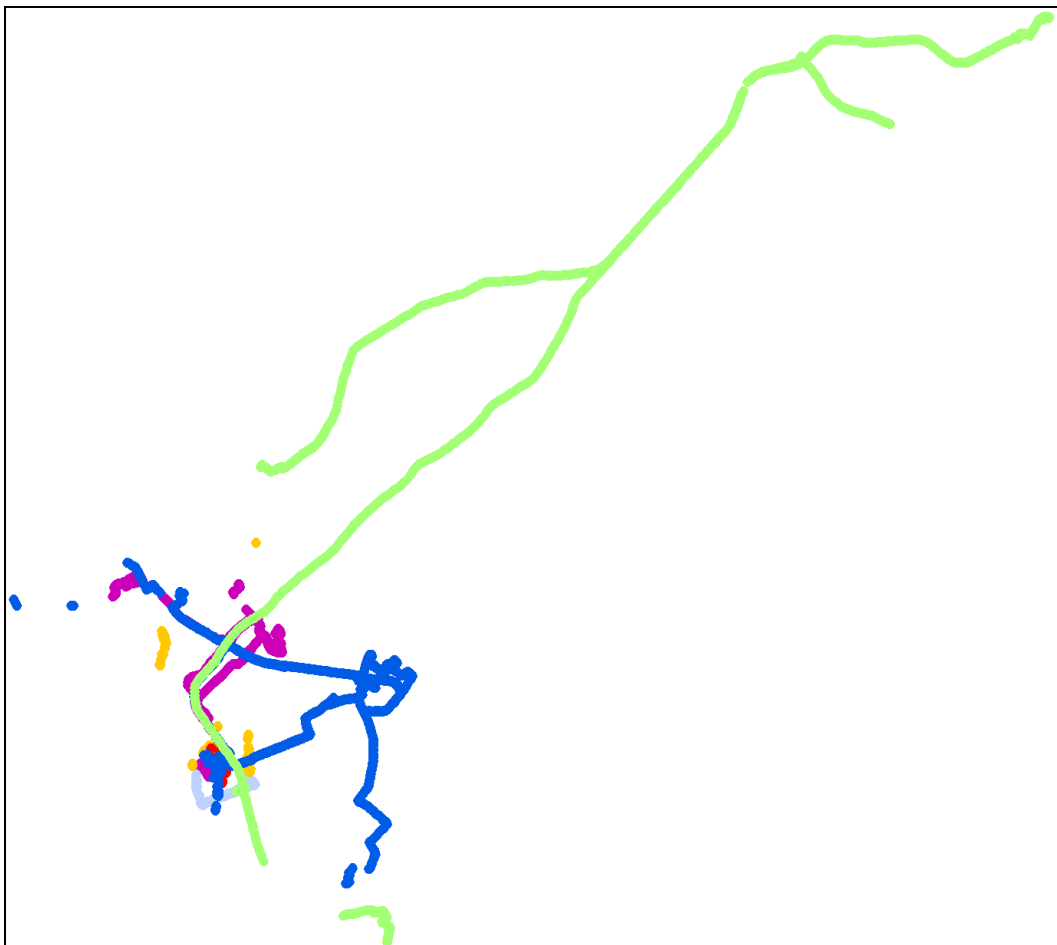


Figure 4.2: Trajectories of six teams in the field; colors identify the different teams.

5 Spatiotemporal data analysis

The spatiotemporal data model is to be used in analysis, in order to help decision-making during emergency. Analysis usually requires combination of existing data with the dynamic data, which in turn requires combination of spatial functionality (on static data) with spatiotemporal functionality.

The AGS adviser (for dangerous substances) might need to know, e.g. the positions of measurement teams in the last two hours. This requires analysis of temporal (dynamic) data for teams of type 'measurement team' (TEAM_TYPE = 2), selecting their positions starting from two hours from the actual time. Figure 5.1 shows, in the left, the trajectories of six teams in the field (the same shown in Figure 4.2), and in the right, positions of the measurement teams (12 and 14) in the last two hours.

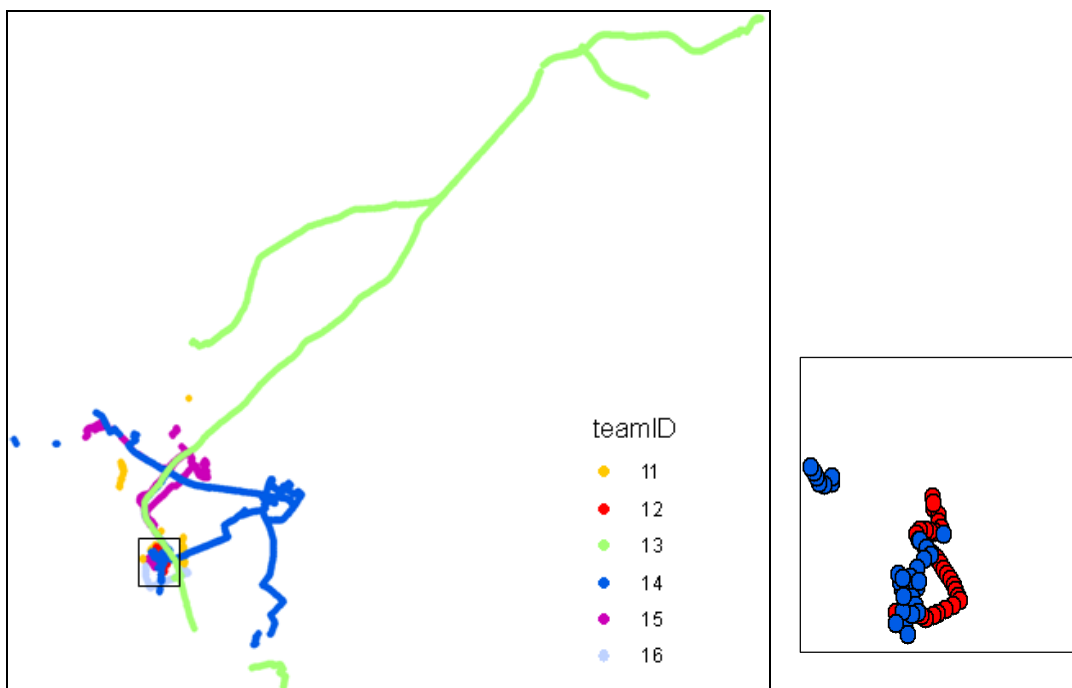


Figure 5.1: Trajectories of six teams in the field (left); trajectories of the measurements teams in the last two hours of an incident (right).

As soon as the threatened area of an incident is entered in the system, it should display the residential buildings and calculate the number of people (registered in these buildings) that are within this area. Such information is needed to organise the evacuation procedure. This requires a combination of dynamic data, threatened area, and existing data, buildings and data on registered residents in these buildings.

Figure 5.2 shows a residential area, the location of an incident, and its threatened area by the semi-transparent bluish circle. Buildings (in red colour) that overlap with this circle are selected, and databases of the municipality are queried to estimate the number of people leaving in these buildings (using BAG code to identify each building).

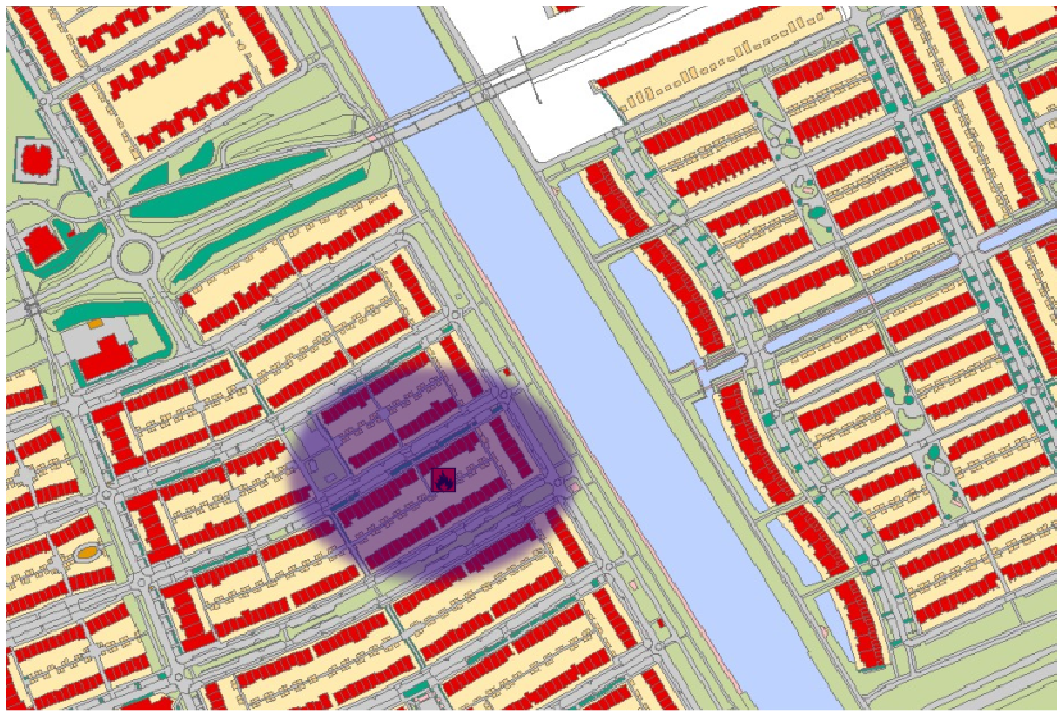


Figure 5.2: A fire incident in a residential area, threatened area is shown by the semi-transparent blue circle.

Spatiotemporal operators (functions) are to be build on the spatiotemporal types introduced in Section 4.2 to provide the functionality for the data analysis. These functions will be defined based on the typical queries needed for the decision-making during the emergency response. Other examples of such queries are:

- Find the location of all the fire brigade teams.
- How many people of the police sector are involved?
- When did the fire brigade, ambulance arrived at the place of incident?
- Find police vehicles that are in a radius of 5km from the incident.
- Give information that has been available 2 hours after the incident has taken place
- Give the number of injured, missing, trapped people ... two hours, four hours, six hours ... after the incident. How did they develop?
- Which police car is the closest to the incident? (dynamic routing)
- Calculate the route for an ambulance, which does not overlap with the gas plume.
- Calculate the speed of expansion of the gas plume.
- Evaluate the evacuation area for the next 8 hours from the **Gasmal** shape and prediction.

6 Conclusions and recommendations

This report presents a spatiotemporal model to maintain operational and situational information in emergency response. It is developed after a careful investigation of the information flow from processes performed by the first responders: fire brigade, paramedics, police and municipality. The model is derived from the organization of emergency response in the Netherlands. It practically represents pieces of information that are currently collected in analogue way (paper templates), via telephone or digitally (but stored in an unstructured way). It captures the type of disaster, the involvement of response sectors (allowing registering of their locations), consequences of the disaster for people, animals and infrastructure, and captures other significant objects. Some of the processes need further consideration, which eventually will require an extension of the model.

The model is currently tested only for the management of spatiotemporal data and more specifically for moving point objects. Since not all the operational and situational data (intended for storage) in the model are currently recorded, it was not possible to test the entire model. An appropriate interface to collect this data is in process of development. Further experiments are needed to validate all the developments and especially the digital replacements of the templates used by the fire brigade and the paramedics. Appropriate digital templates and interfaces (on mobile devices) are needed as well.

Future work includes development of spatiotemporal functionality to allow analysis of spatiotemporal data, e.g. distance between two moving points, direction of movement of a moving region, dynamic shortest route on a (road) network. The model is to be part of a complete ER system where the dynamic data will be combined with existing data accessed via the internet. Investigation of the functionality needed over this combination followed by the development of such functionality is another direction for future work.

Next developments of the model would be to accommodate higher GRIP levels which mean involvement of a wider range of organisations, more actors (i.e. people with specified roles) and wider range of information.

Bibliography

- Chomicki, J. and Revesz, P. Z. 1999, Constraint-based interoperability of spatiotemporal databases. *GeoInformatica*, 3(3):211–243, 1999.
- de By, R. 2005, *Principles of Geographical Information Systems*. ITC Educational Textbook series; 1. International Institute for Geo-Information Science and Earth Observation (ITC), Hengelosestraat 99, P.O. Box 6, 7500 AA Enschede, The Netherlands, 3d edition
- Diehl, S. and van der Heide, J. 2005, Geo Information Breaks through Sector Think, in: Oosterom, Zlatanov&Fendel (eds) *Geo-information for Disaster Management, Earth and Environmental Science*. Springer Berlin Heidelberg, pages 85–108.,
- Diehl, S., Neuvel, J., Zlatanov, S. and Scholten, H. 2006, Investigation of user requirements in the emergency response sector: the Dutch case, Second Symposium on Gi4DM, 25-26 September, Goa, India, CD ROM, 6p.
- Dilo, A. and Zlatanov, S. 2008, Spatiotemporal data modeling for disaster management in the Netherlands, in Walle, Song, Zlatanov, and Li, *Information Systems for Crisis Response and Management*, Harbin Engineering University, 4–6 August 2008, pp. 517–528.
- Güting, R. H. 1994, An introduction to spatial database systems. *VLDB Journal*, 3(4):357–399, 1994.
- Güting, R. H., Bohlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N., Schneider, M. and Vazirgiannis, M. 2000, A foundation for representing and querying moving objects. *ACM Transactions on Databases Systems*, 25(1):1–42, March 2000. 37
- Güting, R. H., Bohlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N., Nardelli, E., Schneider, M. and Viqueira, J. R. R. 2003, Spatio-temporal Models and Languages: An Approach Based on Data Types, in: *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, Lecture Notes in Computer Science. Springer, pp 117–176.
- Güting, R. H. and Schneider, M. *Moving Objects Databases*. Data Management Systems. Morgan Kaufmann, August 2005.
- GHOR Academie. Basisleerstof GHOR, 2008, Technical report, Nederlands Instituut Fysieke Veiligheid Nibra, Arnhem, the Netherlands, July 2008.
- Dong, H. K., Keun, H. R. and Chee, H. P. 2002, Design and implementation of spatiotemporal database query processing system. *Journal of Systems and Software*, 60:37–49, 2002.
- MBZ, 2003, Handboek Voorbereiding Rampendstrijding (in Dutch) available at http://www.nifv.nl/upload/157207_668_1246372018436-HBOEK1_zonder_GRIP.swf (last accessed December 2009)
- Meratnia, N. and de By, R. A. Trajectory representation in location-based services: Problems and solution. In Klas et al. Santucci, editor, *Proceedings of Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03)*, pages 18–24, 2003.
- Meratnia, N. 2005, *Towards Database Support for Moving Object Databases*. PhD thesis, Twente University, Enschede, the Netherlands, 2005.
- Mokbel, M. F. and Aref, W. G. 2008, Sole: scalable on-line execution of continuous queries on spatio-temporal data streams. *VLDB Journal*, 17:971–995, 2008.

- Nederlands Normalisatie Instituut, 2005 *Basis model Geo-informatie*, NEN 3610 (in Dutch) available at <http://www.geonovum.nl/content/basismodel-nen3610>.
- Parent, C., Spaccapietra, S. and Zimanyi, E. 2006, *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer, 2006.
- Scholten, H., Fruijter, S., Dilo, A. and van Borkulo, E. 2008, Spatial Data Infrastructure for emergency response in Netherlands, in: Nayak & Zlatanova (eds) *Remote Sensing and GIS technology for monitoring and prediction of disaster, Environmental Science and Engineering*. Springer-Verlag, pp 177–195
- Shih-Lung, S. and Wang, D. 2000, Handling disaggreate spatiotemporal travel data in gis. *GeoInformatica*, 4(2):161–178, 2000.
- Snoeren, G. 2006, *Rampbestrijdingsprocessen: Actoren, werkwijze, data*. Internal report for GDI4DM, Bsik project RGI-239, Public Aid ‘Gelderland Midden’, October 2006. in Dutch.
- Snoeren, G., Zlatanova, S., Cromptvoets, J. and Scholten, H. 2007, Spatial Data Infrastructure for emergency management: the view of the users. In *Proceedings of the 3rd International symposium on Gi4DM*, Toronto, Canada, 22–25 May 2007.
- Tryfona, N. and Christian, J. S. 1999, Conceptual data modeling for spatiotemporal applications. *GeoInformatica*, 3(3):245–268, 1999.
- Oosterom, P. van, Maessen, B. and Quak, W. 2002, Generic query tool for spatio-temporal data, *International Journal of Geographical Information Science*, 16(8):713–748, 2002. 38
- Oosterom, P. van, Tijssen, T. and Penninga, F. 2005, Topology storage and use in the context of consistent data management, GIS Report No. 33, Delft, 2005, 54 p. (available at www.gdmc.nl/publications)
- Vlotman, S. and Snoeren, G. 2009, Eagle for Public Safety Disaster. Response. In *ESRI Education User Conference*, San Diego, July 11-14, 2009.
- Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R. T., Subrahmanian, V. S. and Zicari, R. 1997, *Advanced Database Systems*. Data Management Systems. Morgan Kaufmann, May 1997.
- Zlatanova, S., van Oosterom, P. and Verbree, E. 2006, Geo-information supports management of urban disasters, *Open House International*, Vol. 31, No.1, March 2006, pp.62-79

Appendix A: Oracle scripts creating the database schema with temporal data at attribute level

The DDL (Data Definition Language) statements for creating the structures for the data model in Oracle Spatial are organised in a number of scripts. A main script, `CreatedMNesT.SQL`, calls all the others: a script that creates the new data types needed in tables, `TypesNesT.SQL`; a script for creating and filling the look-up tables, `LookupTables.SQL`, one for each enumeration type; the script that creates tables shown in Figure 3.13, named `TablesNesT.SQL`; one for creating spatial indices and indices for columns that are foreign keys, `Indices.SQL`; and a script that creates views and metadata for displaying spatiotemporal data, named `ViewsNesT.SQL`. These scripts create tables that manage temporal and spatiotemporal data at attribute level, which is realised through the creation of temporal and spatiotemporal data types as nested tables.

The main script, which calls the other scripts performing specific tasks, `CreatedMNesT.SQL`:

```
/* Creates the data model for operational data.
Uses nested tables for spatiotemporal data. */

spool gdi4dm_struct.log
set trimspool on
set trim on
set echo on

/* Delete existing database: metadata, views, indices, tables &
records, data types. */
@ClearNesT.SQL

/* Create spatiotemporal types: dynamic count, moving point and
moving region, and also types for GHOR info. */
@TypesNesT.SQL

/* Create and fill the look-up tables */
@LookupTables.SQL

/* Create tables */
@TablesNesT.SQL

/* Create indices: foreign keys and spatial indices */
@Indices.SQL

/* Create views to display spatiotemoral data, and fills their
metadata */
@ViewsNesT.SQL

commit;

set trimspool off
```

The script that creates new types for temporal, spatiotemporal and data types needed for GHOR information, `TypesNesT.SQL`:

```

/* Types for a moving point instance
and the full track of moving point */

CREATE OR REPLACE TYPE MPointInst AS OBJECT (
    meas_time    TIMESTAMP,
    point_geo    MDSYS.SDO_GEOMETRY
);
/
CREATE TYPE MOVING_POINT AS TABLE OF MPointInst;
/

/* Types for a moving region instance
and the full history of moving region */

CREATE OR REPLACE TYPE MRegionInst AS OBJECT (
    meas_time    TIMESTAMP,
    region_geo   MDSYS.SDO_GEOMETRY
);
/
CREATE TYPE MOVING_REGION AS TABLE OF MRegionInst;
/

/* Types for dynamic counts, i.e. integers changing with time */

CREATE OR REPLACE FORCE TYPE CntAtInstance AS OBJECT (
    time    TIMESTAMP,
    value   NUMBER(15)
);
/
CREATE TYPE DYNAMIC_NUM AS TABLE OF CntAtInstance;
/

/* Data types for GHOR information */
@Types.SQL

```

The script Types.SQL:

```

-- Type for medical history
CREATE OR REPLACE TYPE MEDICAL_HISTORY AS OBJECT (
    blanco        BOOLEAN,
    bleeding      BOOLEAN,
    CVAstroke     BOOLEAN,
    epilepsy      BOOLEAN,
    hart_problem  BOOLEAN,
    cancer        BOOLEAN,
    lung_affect   BOOLEAN,
    diabetis     BOOLEAN,
    unknown      BOOLEAN,
    other        BOOLEAN
);
/

-- Type for Triage
CREATE OR REPLACE TYPE TriageRec AS OBJECT (
    time    TIMESTAMP,
    eye     PLS_INTEGER,
    motoric PLS_INTEGER,
    verbal  PLS_INTEGER,
    GCS     PLS_INTEGER,
    AF      PLS_INTEGER,
    RR      PLS_INTEGER
);

```

```

/
CREATE TYPE CREATE TYPE TRIAGE AS VARRAY(5) OF TriageRec;
/

-- Type for Treatment
CREATE OR REPLACE TYPE TreatmentRec AS OBJECT (
    time          TIMESTAMP,
    place         NUMBER(1),
    performed_by  PLS_INTEGER,
    triage_class  NUMBER(1),
    treatm_type   NUMBER(1),
    sort_treatm   VARCHAR2(50),
    amount_treat  VARCHAR2(25)
);
/
CREATE TYPE TREATMENT AS VARRAY(6) OF TreatmentRec;
/

-- Type for Exposure
CREATE OR REPLACE TYPE EXPOSURE AS OBJECT (
    level         NUMBER(1),
    radiologic    NUMBER(1),
    biologic      VARCHAR2(25),
    chemic_infect NUMBER(1),
    chemic_type   NUMBER(1),
    chemic_name   VARCHAR2(25)
);
/

```

The script that creates the look-up tables and fills their content, LookupTables.SQL:

```

/* Create Process_Type table, insert records */

CREATE TABLE Process_Type (
    code          NUMBER(2)
    CONSTRAINT PK_Process_Type PRIMARY KEY,
    value_EN      VARCHAR2(70),
    value_NL      VARCHAR2(70)
);

insert into Process_Type values (1,
'Fighting fire and emission of dangerous substances',
'Bestrijden van brand en emissie gevaarlijke stoffen');
insert into Process_Type values (2,
'Rescuing and technical assistance',
'Redden en technische hulpverlening');
insert into Process_Type values (3,
'Decontaminating people and animals',
'Ontsmetten mens en dier');
insert into Process_Type values (4,
'Decontaminating vehicles and infrastructure',
'Ontsmetten voertuigen en infrastructuur');
insert into Process_Type values (5,
'Observations and measurements',
'Waarnemen en meten');
insert into Process_Type values (6,
'Alerting the population',
'Waarschuwen van de bevolking');
insert into Process_Type values (7,
'Making accessible and clearing up',
'Toegankelijk maken en opruimen');

```

```

insert into Process_Type values (8,
'Medical aid chain',
'Geneeskundige Hulpverlening-somatisch');
insert into Process_Type values (9,
'Preventative public health and
medical/environmental measures',
'Preventieve Openbare Gezondheidszorg
(incl.verzamelen besmette waren)');
insert into Process_Type values (10,
'Psycho-social aid and care',
'Geneeskundige Hulpverlening-psychosociaal');
insert into Process_Type values (11,
'Clearance and evacuation',
'Ontruimen en evacueren');
insert into Process_Type values (12,
'Fencing off disaster area',
'Afzetten en afschermen');
insert into Process_Type values (13,
'Traffic control','Verkeer regelen');
insert into Process_Type values (14,
'Maintaining the legal order',
'Handhaven openbare orde');
insert into Process_Type values (15,
'Identification of fatal casualties',
'Identificeren slachtoffers');
insert into Process_Type values (16,
'Giving directions','Begidsen');
insert into Process_Type values (17,
'Criminal investigation','Strafrechtelijk onderzoek');
insert into Process_Type values (18,
'Advice and information','Voorlichten en informeren');
insert into Process_Type values (19,
'Relief and care','Opvangen en verzorgen');
insert into Process_Type values (20,
'Funeral arrangements','Uitvaartverzorging');
insert into Process_Type values (21,
'Registration of victims','Registratie van slachtoffers');
insert into Process_Type values (22,
'Providing primary needs',
'Voorzien in primaire levensbehoeften');
insert into Process_Type values (23,
'Damage registration',
'Registratie van schade en afhandeling');
insert into Process_Type values (24,
'Environment protection','Milieuzorg');
insert into Process_Type values (25,
'Follow-up care','Nazorg');
insert into Process_Type values (26,
'Alerting','Alarmering');
insert into Process_Type values (27,
'Care/logistics of disaster recovery staff',
'Verzorging/logistiek rampbestrijdingspotentieel');
insert into Process_Type values (28,
'Connection/communication','Verbindingen/communicatie');
insert into Process_Type values (29,
'Registration and reporting/archiving',
'Registratie en verslaglegging/Archivering');
insert into Process_Type values (30,
'Evaluation','Evaluatie');

```

```

/* Create Disaster_Type table, insert records */

```

```

CREATE TABLE Disaster_Type (

```

```

code    NUMBER(2)
CONSTRAINT PK_Disaster_Type PRIMARY KEY,
value_EN  VARCHAR2(50),
value_NL  VARCHAR2(80),
group_NL  VARCHAR2(50)
);

insert into Disaster_Type values (1,
'Luchtvaartongeval','Aviation accident',
'Rampen met betrekking tot verkeer en vervoer');
insert into Disaster_Type values (2,
'Ongeval op het water','Accident on water',
'Rampen met betrekking tot verkeer en vervoer');
insert into Disaster_Type values (3,
'Verkeersongeval op het land',
'Traffic accident on land',
'Rampen met betrekking tot verkeer en vervoer');
insert into Disaster_Type values (4,
'Ongeval met brandbare of explosieve stof',
'Accident with inflammable/ explosive material (in open air)',
'Rampen met gevaarlijke stoffen');
insert into Disaster_Type values (5,
'Ongeval met giftige stof',
'Accident with toxic gasses (in open air)',
'Rampen met gevaarlijke stoffen');
insert into Disaster_Type values (6,
'Kernongeval','Nuclear accident',
'Rampen met gevaarlijke stoffen');
insert into Disaster_Type values (7,
'Bedreiging volksgezondheid','Threat to public health',
'Rampen met betrekking tot de volksgezondheid');
insert into Disaster_Type values (8,
'Ziektegolf','Dispersal of disease',
'Rampen met betrekking tot de volksgezondheid');
insert into Disaster_Type values (9,
'Ongevallen in tunnels','Accidents in tunnels',
'Rampen met betrekking tot de infrastructuur');
insert into Disaster_Type values (10,
'Branden in grote gebouwen','Fire in big buildings',
'Rampen met betrekking tot de infrastructuur');
insert into Disaster_Type values (11,
'Instortingen van gebouwen','Collapse of big buildings',
'Rampen met betrekking tot de infrastructuur');
insert into Disaster_Type values (12,
'Uitval nutsvoorzieningen','Disruption of utility',
'Rampen met betrekking tot de infrastructuur');
insert into Disaster_Type values (13,
'Paniek in menigten','Panic in large groups',
'Rampen met betrekking tot de bevolking');
insert into Disaster_Type values (14,
'Grootschalige ordeverstoringen',
'Large-scale disturbance of public peace',
'Rampen met betrekking tot de bevolking');
insert into Disaster_Type values (15,
'Overstromingen','Flood','Natuurrampen');
insert into Disaster_Type values (16,
'Natuurbranden','Nature Fire','Natuurrampen');
insert into Disaster_Type values (17,
'Extreme weersomstandigheden',
'Extreme weather conditions','Natuurrampen');
insert into Disaster_Type values (18, '',
'Incident out of city boundaries, in which citizens
of that city are involved','Ramp op afstand');
insert into Disaster_Type values (19, '', '', '');

```

```

/* Create Team_Type table & insert records */

CREATE TABLE Team_Type (
  code    NUMBER(2)
  CONSTRAINT PK_Team_Type PRIMARY KEY,
  value_EN  VARCHAR2(40),
  value_NL  VARCHAR2(40)
);

insert into Team_Type values (1,
'First measurement team','Eerste meetploeg');
insert into Team_Type values (2,
'Second till fifth measurement team',
'Tweede t/m vijfde meetploeg');

/* Create User_Role table & insert records */

CREATE TABLE User_Role (
  code    NUMBER(3)
  CONSTRAINT PK_User_Role PRIMARY KEY,
  value_EN  VARCHAR2(40),
  value_NL  VARCHAR2(40)
);

/* Create Bravo_Measure table & insert records */

CREATE TABLE Bravo_Measure (
  code    NUMBER(2)
  CONSTRAINT PK_Bravo_Measure PRIMARY KEY,
  bravo_code  VARCHAR2(10),
  colouring  VARCHAR2(20)
);

insert into Bravo_Measure values (1,
'Bravo I.1','blauw -> geel');
insert into Bravo_Measure values (2,
'Bravo I.2','geel -> rood');
insert into Bravo_Measure values (3,
'Bravo I.3','wit -> bruingroen');
insert into Bravo_Measure values (4,
'Bravo I.4','geel -> blauw');
insert into Bravo_Measure values (5,
'Bravo I.5','lichtgrijs -> blauw');
insert into Bravo_Measure values (6,
'Bravo II.1','blauw -> wit');
insert into Bravo_Measure values (7,
'Bravo II.2','wit -> oranje');
insert into Bravo_Measure values (8,
'Bravo II.3','wit -> lichtbruin');
insert into Bravo_Measure values (9,
'Bravo II.4','wit -> blauwviolet');
insert into Bravo_Measure values (10,
'Bravo II.5','wit -> rood');

/* Create Drawing_Symbol table & insert records */

CREATE TABLE Drawing_Symbol (
  code    NUMBER(2)
  CONSTRAINT PK_Draw_Symbol PRIMARY KEY,

```

```

name_EN    VARCHAR2(45),
name_NL    VARCHAR2(45),
description VARCHAR2(200),
symbol     BLOB
);

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (1,'Anker','Gas Leak','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (2,'Brand','Fire','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (3,'Defensie','Military','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (4,'Derden','Third Parties','Anyone else besides the
parties explicitly defined by other symbols, e.g volunteers');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (5,'Gemeente','Municipality','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (6,'Brandweer','Fire-fighters','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (7,'GHOR','Medical Services','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (8,'Hulpverlening','Assistance','');
insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (9,'Logistiek','Logistics','Used to indicate any place
that might be relevant to logistics,such as water or any other
kind of supplies or even the position of the person in charge of
the logistics.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (10,'NS','Train Station (NS)',
'Train station or anything else that is related to trains.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (11,'Meetopdracht**','Measurement task',
'This is a special symbol because after it is positioned on the
map, a form is presented where a set of instructions is defined
for a selected vehicle.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (12,'Politie','Police','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (13,'Provincie','Province Building','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (14,'Rijk','Government Building','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (15,'Rijkwaterstaat','Ministry for traffic and water
management','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (16,'Uitgangstelling','','In case a building is on fire,
the fire brigade has already made plans of positions where
vehicles should be placed. This symbol is used to indicate exactly
these points.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (17,'Versperring','Barricade','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (18,'Waarschuwing','Warning','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (19,'Waterschap','Authority responsible for water
management','');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (20,'Zwaartepunt','Point of high importance','This symbol
is used to indicate a place of very high priority or where the
biggest damage has occurred.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (21,'Lijn blauw','Blue Line','Used to draw a line
on the map. Action is finalized by double-click.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (22,'Lijn groen','Line Green','Used to draw a line
on the map. Action is finalized by double-click.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (23,'Lijn rood','Red Line','Used to draw a line
on the map. Action is finalized by double-click.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (24,'Lijn zwart','Black Line','Used to draw a line
on the map. Action is finalized by double-click.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (25,'Vlak blauw','Polygon in Blue','Used to draw
a polygon on the map. Action is finalized by double-clik.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (26,'Vlak groen','Polygon in Green','Used to draw
a polygon on the map. Action is finalized by double-clik.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (27,'Vlak rood','Polygon in Red','Used to draw
a polygon on the map. Action is finalized by double-clik.');
```

```

insert into Drawing_Symbol(code, name_EN, name_NL, description)
values (28,'Vlak zwart','Polygon in Black','Used to draw
a polygon on the map. Action is finalized by double-clik.');
```

```
/* Create Safety_Region table & insert records */
```

```

CREATE TABLE Safety_Region (
  region_code  NUMBER(2) CONSTRAINT PK_Safe_Region PRIMARY KEY,
  region_name  VARCHAR2(45)
);
```

```

insert into Safety_Region values (1,'');
-- ...
insert into Safety_Region values (26,'');
```

```
/* Create GRIP_level table & insert records */
```

```

CREATE TABLE GRIP_Level (
  grip_level  NUMBER(1) CONSTRAINT PK_GRIP_Lev PRIMARY KEY,
  grip_descr  VARCHAR2(200)
);
```

```

insert into GRIP_Level values (0,'');
-- ...
insert into GRIP_Level values (5,'');
```

```
/* Create Scale_Level table & insert records */
```

```

CREATE TABLE Scale_Level (
  scale_level  NUMBER(1) CONSTRAINT PK_Scale_Lev PRIMARY KEY,
  scale_descr  VARCHAR2(200)
);
```

```

insert into Scale_Level values (1,'');
-- ...
```

```
/* Create Triage_Val table & insert records */
```

```

CREATE TABLE Triage_Val (
  triage_code  NUMBER(1) CONSTRAINT PK_Triage PRIMARY KEY,
  triage_name  VARCHAR2(45)
);
```



```

insert into Triage_Val values (1,'');
-- ...
insert into Triage_Val values (4,'');

/* Create Gender table & insert records */

CREATE TABLE Gender (
  gender_code NUMBER(1) CONSTRAINT PK_Gender PRIMARY KEY,
  gender_name VARCHAR2(6)
);

insert into Gender values (0,'male');
insert into Gender values (1,'female');

/* Create Injury_Type table & insert records */

CREATE TABLE Injury_Type (
  injury_code NUMBER(1) CONSTRAINT PK_Injury PRIMARY KEY,
  injury_name VARCHAR2(25)
);

insert into Injury_Type values (1,'');
-- ...

/* Create Relief_Centre table & insert records */

CREATE TABLE Relief_Centre (
  rcentre_code NUMBER(1) CONSTRAINT PK_RCentre PRIMARY KEY,
  rcentre_desc VARCHAR2(25)
);

insert into Relief_Centre values (1,'');
-- ...

/* Create Exposure_Type table & insert records */

CREATE TABLE Exposure_Type (
  exposure_code NUMBER(1) CONSTRAINT PK_Exposure PRIMARY KEY,
  exposure_desc VARCHAR2(25)
);

insert into Exposure_Type values (1,'');
-- ...

/* Create RadioExp_Type table & insert records */

CREATE TABLE RadioExp_Type (
  radioexp_code NUMBER(1)
  CONSTRAINT PK_RadioExp PRIMARY KEY,
  radioexp_desc VARCHAR2(25)
);

insert into RadioExp_Type values (1,'');
-- ...

/* Create Chemic_Infection table & insert records */

```

```

CREATE TABLE Chemic_Infection (
  chinfect_code NUMBER(1)
  CONSTRAINT PK_ChemicInfec PRIMARY KEY,
  chinfect_desc VARCHAR2(25)
);

insert into Chemic_Infection values (1,'');
-- ...

/* Create existing Chemical_Type table & insert records */

CREATE TABLE Chemical_Type (
  chemical_code NUMBER(1)
  CONSTRAINT PK_ChemicType PRIMARY KEY,
  chemical_desc VARCHAR2(25)
);

insert into Chemical_Type values (1,'');
-- ...

/* Create Treatment_Place table & insert records */

CREATE TABLE Treatment_Place (
  tplace_code NUMBER(1)
  CONSTRAINT PK_TreatPlace PRIMARY KEY,
  tplace_desc VARCHAR2(25)
);

insert into Treatment_Place values (1,'');
-- ...

/* Create Treatment_Type table & insert records */

CREATE TABLE Treatment_Type (
  treatment_code NUMBER(1)
  CONSTRAINT PK_TreatType PRIMARY KEY,
  treatment_desc VARCHAR2(25)
);

insert into Treatment_Type values (1,'');
-- ...

```

The script that creates the tables of the data model explained in Section 3.4, TablesNest.SQL:

```

/* Create tables and constraints, primary key, foreign keys, and
check constraints */

CREATE TABLE Incident (
  incidentID NUMBER(15)
  CONSTRAINT PK_Incident PRIMARY KEY,
  location MDSYS.SDO_GEOMETRY NOT NULL,
  fenced_area MDSYS.SDO_GEOMETRY,
  start_time DATE,
  end_time DATE,
  description VARCHAR2(200)
);

CREATE TABLE Hypothetical (
  hypotheticalID NUMBER(15)

```

```

        CONSTRAINT FK_Hypothetical REFERENCES Incident,
        disaster_type NUMBER(2)
        CONSTRAINT REF_Disaster
        REFERENCES Disaster_Type,
        CONSTRAINT PK_Hypothetical
        PRIMARY KEY (hypotheticalID)
    );

CREATE TABLE RealIncident (
    realIncidentID NUMBER(15)
    CONSTRAINT FK_RealIncident REFERENCES Incident,
    disaster_type DYNAMIC_NUM,
    affected_area MOVING_REGION,
    threatened_area MDSYS.SDO_GEOMETRY,
    estimation_time TIMESTAMP,
    GRIPlevel DYNAMIC_NUM,
    scale DYNAMIC_NUM,
    evacuation_area MDSYS.SDO_GEOMETRY,
    escalation VARCHAR2(1000),
    CONSTRAINT PK_RealIncident
    PRIMARY KEY (realIncidentID)
)
NESTED TABLE affected_area STORE AS IncExtent,
NESTED TABLE disaster_type STORE AS IncDType,
NESTED TABLE scale STORE AS IncScale,
NESTED TABLE GRIPlevel STORE AS IncGRIP;

/* Disaster types and GRIP levels should be specialisation of
dynamic_num type, their respective values restricted to disaster
types (in look-up table) and grip levels */

CREATE TABLE Complaint (
    complaintID NUMBER(15)
    CONSTRAINT PK_Complaint PRIMARY KEY,
    call_time TIMESTAMP NOT NULL,
    location MDSYS.SDO_GEOMETRY,
    incidentID NUMBER(15) CONSTRAINT FK_Complaint_Incident
    REFERENCES RealIncident,
    report BLOB NOT NULL
);

CREATE TABLE Gasmal (
    incidentID NUMBER(15) CONSTRAINT FK_Gasmal_Incident
    REFERENCES RealIncident,
    shape MOVING_REGION,
    label VARCHAR2(15),
    prediction MOVING_REGION,
    CONSTRAINT PK_Gasmal PRIMARY KEY (incidentID)
)
NESTED TABLE shape STORE AS GasmalShape,
NESTED TABLE prediction STORE AS GasmalPredict;

CREATE TABLE Sectormal (
    incidentID NUMBER(15) CONSTRAINT FK_Sectormal_Incident
    REFERENCES RealIncident,
    sectors VARCHAR2(20) NOT NULL,
    label VARCHAR2(20),
    description VARCHAR2(1000),
    CONSTRAINT PK_Sectormal PRIMARY KEY (incidentID)
);

CREATE TABLE Process (
    incidentID NUMBER(15) CONSTRAINT FK_Process_Incident

```

```

REFERENCES RealIncident,
process_type NUMBER(2) CONSTRAINT REF_Process
REFERENCES Process_Type,
start_time TIMESTAMP,
end_time TIMESTAMP,
CONSTRAINT PK_Process
PRIMARY KEY (incidentID, process_type)
);

CREATE TABLE Department (
departmentID NUMBER(15)
CONSTRAINT PK_Department PRIMARY KEY,
dept_code VARCHAR2(10) NOT NULL,
safety_region NUMBER(2) CONSTRAINT REF_SafeRegions
REFERENCES Safety_Region,
location MDSYS.SDO_GEOMETRY
);

CREATE TABLE DeptResp4Proc (
departmentID NUMBER(15) CONSTRAINT FK_RespDept
REFERENCES Department,
incidentID NUMBER(15),
process_type NUMBER(2),
start_time TIMESTAMP,
end_time TIMESTAMP,
CONSTRAINT FK_ProcessCtrl FOREIGN KEY
(incidentID, process_type) REFERENCES Process,
CONSTRAINT PK_DeptResp4Proc PRIMARY KEY
(departmentID, incidentID, process_type)
);

CREATE TABLE DM_User (
userID NUMBER(15) CONSTRAINT PK_User PRIMARY KEY,
userBSN CHAR(9) NOT NULL,
deptID NUMBER(15) CONSTRAINT FK_UserDept
REFERENCES Department
);

CREATE TABLE UserInProcess (
userID NUMBER(15) CONSTRAINT FK_User_InProcess
REFERENCES DM_User,
incidentID NUMBER(15),
process_type NUMBER(2),
role NUMBER(3) CONSTRAINT REF_UserRole
REFERENCES User_Role,
start_time TIMESTAMP,
end_time TIMESTAMP,
CONSTRAINT FK_UserIn_Process FOREIGN KEY
(incidentID, process_type) REFERENCES Process
(incidentID, process_type),
CONSTRAINT PK_UserInProcess
PRIMARY KEY (userID, incidentID, process_type)
);

CREATE TABLE Vehicle (
vehicleID NUMBER(15) CONSTRAINT PK_Vehicle PRIMARY KEY,
vehicle_code VARCHAR2(10) NOT NULL,
deptID NUMBER(15) CONSTRAINT FK_Vehicle2Dept
REFERENCES Department,
calculatedETA TIMESTAMP
);

CREATE TABLE Team (
teamID NUMBER(20) CONSTRAINT PK_Team PRIMARY KEY,

```

```

name VARCHAR2(20),
team_type NUMBER(2) CONSTRAINT REF_Team
REFERENCES Team_Type,
no_members NUMBER(2),
leaderID NUMBER(15) CONSTRAINT FK_TeamLead
REFERENCES DM_SUser NOT NULL,
vehicleID NUMBER(15) CONSTRAINT FK_TravelWith
REFERENCES Vehicle,
location MOVING_POINT
)
NESTED TABLE position STORE AS TeamPosition;

CREATE TABLE Measurement (
taskID NUMBER(20) CONSTRAINT PK_Measurement PRIMARY KEY,
incidentID NUMBER(15) CONSTRAINT FK_Meas_Incident
REFERENCES RealIncident ON DELETE SET NULL,
AGSuserID NUMBER(15) CONSTRAINT FK_AGS_userID
REFERENCES DM_SUser,
location MDSYS.SDO_GEOMETRY NOT NULL,
assignment_time TIMESTAMP NOT NULL,
task_explosionLEL CHAR(1) CONSTRAINT CHK_explosion_task
CHECK (task_explosionLEL in ('Y', 'N')),
gas_tube BINARY_FLOAT,
task_automess CHAR(1) CONSTRAINT CHK_automess_task
CHECK (task_automess in ('Y', 'N')),
task_automess_sonda CHAR(1)
CONSTRAINT CHK_automess_sonda_task
CHECK (task_automess_sonda in ('Y', 'N')),
task_dosimeter CHAR(1) CONSTRAINT CHK_dosimeter_task
CHECK (task_dosimeter in ('Y', 'N')),
protection CHAR(1) CONSTRAINT CHK_protection_task
CHECK (protection in ('Y', 'N')),
teamID NUMBER(20) CONSTRAINT FK_Measurement_Team
REFERENCES Team ON DELETE SET NULL,
meas_time TIMESTAMP NOT NULL,
explosionLEL BINARY_FLOAT,
gas_tube_no NUMBER(2) CONSTRAINT REF_Bravo
REFERENCES Bravo_Measure(code),
pumping_no NUMBER(4),
concentration BINARY_FLOAT,
automess1 BINARY_FLOAT,
automess2 BINARY_FLOAT,
automess_sonda BINARY_FLOAT,
dosimeter BINARY_FLOAT,
fed2gasmal TIMESTAMP,
task_details VARCHAR2(400),
meas_details VARCHAR2(400),
CONSTRAINT CHK_explosion CHECK (task_explosionLEL <> 'Y' OR
explosionLEL IS NOT NULL) DISABLE,
CONSTRAINT CHK_gas_tube CHECK (gas_tube IS NULL OR
(gas_tube_no IS NOT NULL AND pumping_no IS NOT NULL
AND concentration IS NOT NULL)) DISABLE,
CONSTRAINT CHK_automess CHECK (task_automess <> 'Y' OR
(automess1 IS NOT NULL AND automess2 IS NOT NULL)) DISABLE,
CONSTRAINT CHK_automess_sonda CHECK
(task_automess_sonda <> 'Y'
OR automess_sonda IS NOT NULL) DISABLE,
CONSTRAINT CHK_dosimeter CHECK (task_dosimeter <> 'Y'
OR dosimeter IS NOT NULL) DISABLE
);

```

```

/* Data in this table will be stored in pieces: first the
measurement task, then the measurements for the given task id.
Constraints on the measurements are disabled because the fields

```

will be empty at first. They can be checked later with a command:
ALTER TABLE Measurement ENABLE VALIDATE CONSTRAINT CHK_explosion;
for the CHK_explosion constraint and similar for all others */

```
CREATE TABLE EventObject (  
  objectID NUMBER(20) CONSTRAINT PK_EventObject PRIMARY KEY,  
  incidentID NUMBER(15) CONSTRAINT FK_Object_Incident  
  REFERENCES RealIncident NOT NULL,  
  userID NUMBER(15) CONSTRAINT FK_Object_Creater  
  REFERENCES DM_SUser,  
  drawing_symbol NUMBER(2) CONSTRAINT REF_Symbol  
  REFERENCES Drawing_Symbol,  
  geometry MDSYS.SDO_GEOMETRY NOT NULL,  
  description VARCHAR2(200)  
);
```

```
CREATE TABLE ReliefCentre (  
  centreID NUMBER(10) CONSTRAINT PK_RCentre PRIMARY KEY,  
  centre_type NUMBER(1) CONSTRAINT REF_RCentre  
  REFERENCES Relief_Centre,  
  capacity NUMBER(20),  
  location MDSYS.SDO_GEOMETRY  
);
```

```
CREATE TABLE PatientCard (  
  patientID NUMBER(15) CONSTRAINT PK_PatientCard  
  PRIMARY KEY,  
  incidentID NUMBER(15) CONSTRAINT FK_Patient_Incident  
  REFERENCES RealIncident NOT NULL,  
  centreID NUMBER(10) CONSTRAINT FK_Patient_RCentre  
  REFERENCES ReliefCentre,  
  gender NUMBER(1) CONSTRAINT REF_Gender  
  REFERENCES Gender,  
  name VARCHAR2(50),  
  birthdate DATE,  
  address VARCHAR2(50),  
  IDcard VARCHAR2(50),  
  tel_family VARCHAR2(50),  
  place_found VARCHAR2(50),  
  allergy VARCHAR2(50),  
  medication VARCHAR2(50),  
  past_MR MEDICAL_HISTORY,  
  last_meal VARCHAR2(50),  
  decontamination CHAR(1) CONSTRAINT CHK_decontamination  
  CHECK (decontamination in ('Y', 'N')),  
  accident-mech VARCHAR2(50),  
  head-diagnose VARCHAR2(50),  
  injury NUMBER(1) CONSTRAINT REF_Injury  
  REFERENCES Injury_Type,  
  left_pupilR CHAR(1) CONSTRAINT CHK_LPupil  
  CHECK (left_pupilR in ('Y', 'N')),  
  right_pupilR CHAR(1) CONSTRAINT CHK_RPupil  
  CHECK (right_pupilR in ('Y', 'N')),  
  notes VARCHAR2(50),  
  triage_record TRIAGE,  
  triage_class NUMBER(1) CONSTRAINT REF_Triage  
  REFERENCES Triage_Val,  
  treatment TREATMENT,  
  remarks VARCHAR2(50)  
);
```

```
CREATE TABLE Casualty (  
  incidentID NUMBER(15) CONSTRAINT FK_Casualty_Incident
```



```

REFERENCES RealIncident,
psychologicP DYNAMIC_NUM,
triage1 DYNAMIC_NUM,
triage2 DYNAMIC_NUM,
triage3 DYNAMIC_NUM,
triage3n DYNAMIC_NUM,
triage4 DYNAMIC_NUM,
dead DYNAMIC_NUM,
CONSTRAINT PK_Casualty PRIMARY KEY (incidentID)
)
NESTED TABLE psychologicP STORE AS DynPsychologic,
NESTED TABLE triage1 STORE AS DynTriage1,
NESTED TABLE triage2 STORE AS DynTriage2,
NESTED TABLE triage3 STORE AS DynTriage3,
NESTED TABLE triage3n STORE AS DynTriage3n,
NESTED TABLE triage4 STORE AS DynTriage4,
NESTED TABLE dead STORE AS DynDeath;

CREATE TABLE DamagePA (
incidentID NUMBER(15) CONSTRAINT FK_DamagePA_Incident
REFERENCES RealIncident,
trapped DYNAMIC_NUM,
missing DYNAMIC_NUM,
people2evac DYNAMIC_NUM,
people2decontam DYNAMIC_NUM,
animal2decontam DYNAMIC_NUM,
people4shelter DYNAMIC_NUM,
people2feed DYNAMIC_NUM,
animal2feed DYNAMIC_NUM
CONSTRAINT PK_DamagePA PRIMARY KEY (incidentID)
)
NESTED TABLE trapped STORE AS DynTrapped,
NESTED TABLE missing STORE AS DynMissing,
NESTED TABLE people2evac STORE AS Dynp2evac,
NESTED TABLE people2decontam STORE AS Dynp2decont,
NESTED TABLE animal2decontam STORE AS Dyna2decont,
NESTED TABLE people4shelter STORE AS Dynp4shelter,
NESTED TABLE people2feed STORE AS Dynp2feed,
NESTED TABLE animal2feed STORE AS Dyna2feed;

CREATE TABLE DamagedBuilding (
incidentID NUMBER(15) CONSTRAINT FK_DamageBuild_Incident
REFERENCES RealIncident,
BAGcode VARCHAR2(20),
CONSTRAINT PK_DamageBuild PRIMARY KEY (incidentID)
);

CREATE TABLE DamagedCar (
incidentID NUMBER(15) CONSTRAINT FK_DamageCar_Incident
REFERENCES RealIncident,
plate_no VARCHAR2(15),
CONSTRAINT PK_DamageCar PRIMARY KEY (incidentID)
);

```

The script that creates indices for all the foreign key columns and spatial indices, Indices.SQL:

```

/* Create indices for foreign keys */

CREATE INDEX IX_Complaint_Incident
ON Complaint (incidentID);

```

```

CREATE INDEX IX_Department_Code
ON Department (dept_code);

CREATE INDEX IX_DeptResp_Process
ON DeptResp4Proc (incidentID, process_type);

CREATE INDEX IX_DMuser_BSN ON DM_SUser (userBSN);

CREATE INDEX IX_DMuser_Dept ON DM_SUser (deptID);

CREATE INDEX IX_DMuser_inProc
ON User_In_Process (incidentID, process_type);

CREATE INDEX IX_Vehicle_Code
ON Vehicle (vehicle_code);

CREATE INDEX IX_Vehicle_Dept ON Vehicle (deptID);

CREATE INDEX IX_Team_Leader ON Team (leaderID);

CREATE INDEX IX_Team_Vehicle ON Team (vehicleID);

CREATE INDEX IX_Meas_incident
ON Measurement (incidentID);

CREATE INDEX IX_Meas_Team ON Measurement (teamID);

CREATE INDEX IX_Meas_AGS ON Measurement (AGSuserID);

CREATE INDEX IX_Event_Incident
ON EventObject (incidentID);

CREATE INDEX IX_Event_Type
ON EventObject (drawing_symbol);

CREATE INDEX IX_Patient_Incident
ON PatientCard (incidentID);

CREATE INDEX IX_Patient_Centre
ON PatientCard (centreID);

/* Spatial indices for Incident, RealIncident,
Complaint, Department, Measurement, EventObject, ReliefCentre.
Metadata should exist before creating the spatial index. The
values entered into SDO_DIM_ELEMENT are fictitious. Should be
UPDATED later with the real values. */

/* Metadata and spatial indices for Incident */

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('Incident',
'location',
SDO_DIM_ARRAY(
SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
90112);

CREATE INDEX SIX_Incident_loc ON Incident (location)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('Incident',
'fenced_area',

```

```

SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
    SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
90112);

CREATE INDEX SIX_Incident_fenced ON Incident (fenced_area)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

/* Metadata and spatial indices for RealIncident */

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('RealIncident',
 'threatened_area',
 SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
    SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
90112);

CREATE INDEX SIX_Incident_threatened
ON RealIncident (threatened_area)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('RealIncident',
 'evacuation_area',
 SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
    SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
90112);

CREATE INDEX SIX_Incident_evacuation
ON RealIncident (evacuation_area)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

/* Insert metadata & create spatial index of Complaint */

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('Complaint',
 'location',
 SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
    SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
90112);

CREATE INDEX SIX_Complaint_loc ON
Complaint (location) INDEXTYPE IS MDSYS.SPATIAL_INDEX;

/* Metadata & spatial index for Department */

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('Department',
 'location',
 SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
    SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
90112);

CREATE INDEX SIX_Department_loc ON Department (location)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

/* Metadata & spatial index for Measurement */
INSERT INTO USER_SDO_GEOM_METADATA VALUES
('Measurement',
 'location',

```

```

        SDO_DIM_ARRAY(
            SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
            SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
        90112);

CREATE INDEX SIX_Measurement_loc ON
Measurement (location) INDEXTYPE IS MDSYS.SPATIAL_INDEX;

/* Metadata & spatial index for Drawing Objects */

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('EventObject',
 'geometry',
 SDO_DIM_ARRAY(
     SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
     SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
 90112);

CREATE INDEX SIX_EventObject_geo ON
EventObject (geometry) INDEXTYPE IS MDSYS.SPATIAL_INDEX;

/* Metadata & spatial index for ReliefCentre */

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('ReliefCentre',
 'location',
 SDO_DIM_ARRAY(
     SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
     SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
 90112);

CREATE INDEX SIX_ReliefCentre_loc ON
ReliefCentre (location) INDEXTYPE IS MDSYS.SPATIAL_INDEX;

```

The script that creates views and fills their metadata, ViewsNesT.SQL:

```

/* Create views for the spatiotemporal data and metadata for the
views, which is needed for visualisation, e.g. in FME, but also if
we create later a functional (spatial) index in the field. ALL
METADATA entered from this scrip into SDO_DIM_ELEMENT are
FICTITIOUS. Should be UPDATED later with the real values. */

/* Create a view to see the change of extent of the current
incident, i.e. with maximal ID, then insert metadata for the view
*/

CREATE OR REPLACE VIEW IncidentExt (STime, Geometry) AS
SELECT ie.meas_time, ie.region_geo
FROM THE(SELECT affected_area
        FROM RealIncident
        WHERE realIncidentID = (SELECT MAX(realIncidentID)
                                FROM RealIncident)
        ) ie;

INSERT INTO USER_SDO_GEOM_METADATA VALUES
('IncidentExt',
 'Geometry',
 SDO_DIM_ARRAY(
     SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
     SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
 90112);

```

```
/* Create view for the Gasmal (changes) of the current incident,
then insert metadata for this view */
```

```
CREATE OR REPLACE VIEW GasmalExt (STime, Geometry) AS
  SELECT gs.meas_time, gs.region_geo
  FROM THE(SELECT shape
            FROM Gasmal
            WHERE IncidentId = (SELECT MAX(IncidentId)
                                FROM Gasmal)
          ) gs;
```

```
INSERT INTO USER_SDO_GEOM_METADATA VALUES
  ('GasmalExt',
   'Geometry',
   SDO_DIM_ARRAY(
     SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
     SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
   90112);
```

```
/* Create a view to track the position of a team.
It seems reasonable to look for teams involved in the current
incident. For the moment we do not keep a direct relation Team-
Incident in the data model, but we can get this through the
relation Team-DM_SUser, which means we will add a join in the
query: */
```

```
CREATE OR REPLACE VIEW TeamTracking AS
  SELECT t.teamid, t.team_type, t.name, t.no_members, p.*
  FROM Team t, TABLE(t.location) p;
```

```
INSERT INTO USER_SDO_GEOM_METADATA VALUES
  ('TeamTracking',
   'Geometry',
   SDO_DIM_ARRAY(
     SDO_DIM_ELEMENT ('X', 85000, 150000, 0.05),
     SDO_DIM_ELEMENT ('Y', 467000, 515000, 0.05)),
   90112);
```

```
/* More views may be needed; to be created as above. */
```


Appendix B: Oracle scripts creating the database schema with temporal data at record level

This appendix contains Oracle scripts with the DDL statements to create the structures for the data model in Oracle Spatial by employing only regular tables. The temporal and spatiotemporal data are handled at the record level. The DDL statements are again organised in several scripts with a main script, `CreatedMRegT.SQL`, that calls all the others: the script `Types.SQL` that creates the new data types needed for GHOR information; the script `LookupTables.SQL` to create and fill the look-up tables; a script, `TablesRegT.SQL`, that creates tables of the data model (Figure 3.13); and the script `Indices.SQL` to create spatial indices and indices for columns that are foreign keys. Below are given only the main script and the script that creates the tables. The other scripts called by the main script are provided in Appendix A.

The main script, `CreatedMRegT.SQL`:

```
/* Creates the data model for operational data of GDI4DM.
Every temporal or spatiotemporal attribute is stored
in a separate table. */

spool gdi4dm_struct.log
set trimspool on
set trim on
set echo on

/* Delete existing database, tables & records, types,
indices, views, metadata */
@ClearAll_RegT.SQL

/* Create new data types, needed for GHOR information */
@Types.SQL

/* Create and fill the look-up tables */
@LookupTables.SQL

/* Create tables */
@TablesRegT.SQL

/* Create indices: foreign keys and spatial indices */
@Indices.SQL

/* More metadata & spatial indices are to be created for the
regular tables that store the spatiotemporal data */

commit;

set trimspool off
```

The script that creates the tables of the data model explained in Section 3.4, `TablesRegT.SQL`:

```

/* Create tables and constraints, primary key, foreign keys, and
check constraints */

/* Tables for Incident, Hypothetical and RealIncident;
separate tables for temporal and spatiotemporal attributes:
affected_area, disaster_type, GRIPlevel, and scale. */

CREATE TABLE Incident (
  incidentID NUMBER(15)
  CONSTRAINT PK_Incident PRIMARY KEY,
  location MDSYS.SDO_GEOMETRY NOT NULL,
  fenced_area MDSYS.SDO_GEOMETRY,
  start_time DATE,
  end_time DATE,
  description VARCHAR2(200)
);
CREATE TABLE Hypothetical (
  hypotheticalID NUMBER(15)
  CONSTRAINT FK_Hypothetical REFERENCES Incident,
  disaster_type NUMBER(2)
  CONSTRAINT REF_Disaster
  REFERENCES Disaster_Type,
  CONSTRAINT PK_Hypothetical
  PRIMARY KEY (hypotheticalID)
);
CREATE TABLE RealIncident (
  realIncidentID NUMBER(15)
  CONSTRAINT FK_RealIncident REFERENCES Incident,
  threatened_area MDSYS.SDO_GEOMETRY,
  estimation_time TIMESTAMP,
  evacuation_area MDSYS.SDO_GEOMETRY,
  escalation VARCHAR2(1000),
  CONSTRAINT PK_RealIncident
  PRIMARY KEY (realIncidentID)
);
CREATE TABLE AffectedA (
  incidentID NUMBER(15) CONSTRAINT FK_Inc_AffectedA
  REFERENCES Incident,
  meas_time TIMESTAMP,
  affected_area MDSYS.SDO_GEOMETRY,
  CONSTRAINT PK_AffectedA PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE DisasterT (
  incidentID NUMBER(15) CONSTRAINT FK_Inc_DisasterT
  REFERENCES Incident,
  meas_time TIMESTAMP,
  disaster_type NUMBER(2) CONSTRAINT REF_Disaster
  REFERENCES Disaster_Type,
  CONSTRAINT PK_DisasterT PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE GRIPlevel (
  incidentID NUMBER(15) CONSTRAINT FK_Inc_GRIPlevel
  REFERENCES Incident,
  meas_time TIMESTAMP,
  grip_level NUMBER(1) CONSTRAINT CHK_GRIP
  CHECK (grip_level >= 0 AND grip_level <= 5),
  CONSTRAINT PK_GRIPlevel PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE IncScale (
  incidentID NUMBER(15) CONSTRAINT FK_IncScale
  REFERENCES Incident,
  meas_time TIMESTAMP,
  scale NUMBER(1) CONSTRAINT CHK_SCALE
  CHECK (scale >= 1 AND scale <= 10),

```

```

        CONSTRAINT PK_IncScale PRIMARY KEY (incidentID, meas_time)
    );

/* assuming agreed scale levels 1-10 */

CREATE TABLE Complaint (
    complaintID NUMBER(15) CONSTRAINT PK_Complaint PRIMARY KEY,
    call_time    TIMESTAMP NOT NULL,
    location     MDSYS.SDO_GEOMETRY,
    incidentID  NUMBER(15) CONSTRAINT FK_Complaint_Incident
    REFERENCES RealIncident,
    report      BLOB NOT NULL
);

/* Gasmal tables */

CREATE TABLE Gasmal (
    incidentID NUMBER(15) CONSTRAINT FK_Gasmal_Incident
    REFERENCES RealIncident,
    label      VARCHAR2(15),
    CONSTRAINT PK_Gasmal PRIMARY KEY (incidentID)
);

CREATE TABLE GasmalShape (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_GasmalShape
    REFERENCES RealIncident,
    meas_time  TIMESTAMP,
    shape      MDSYS.SDO_GEOMETRY,
    CONSTRAINT PK_GasmalShape PRIMARY KEY (incidentID,
    meas_time)
);

CREATE TABLE GasmalPred (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_GasmalPred
    REFERENCES RealIncident,
    meas_time  TIMESTAMP,
    prediction MDSYS.SDO_GEOMETRY,
    CONSTRAINT PK_GasmalPred PRIMARY KEY (incidentID, meas_time)
);

CREATE TABLE Sectormal (
    incidentID NUMBER(15) CONSTRAINT FK_Sectormal_Incident
    REFERENCES RealIncident,
    sectors    VARCHAR2(20) NOT NULL,
    label      VARCHAR2(20),
    description VARCHAR2(1000),
    CONSTRAINT PK_Sectormal PRIMARY KEY (incidentID)
);

CREATE TABLE Process (
    incidentID NUMBER(15) CONSTRAINT FK_Process_Incident
    REFERENCES RealIncident,
    process_type NUMBER(2) CONSTRAINT REF_Process
    REFERENCES Process_Type,
    start_time  TIMESTAMP,
    end_time    TIMESTAMP,
    CONSTRAINT PK_Process PRIMARY KEY (incidentID, process_type)
);

CREATE TABLE Department (
    departmentID NUMBER(15) CONSTRAINT
    PK_Department PRIMARY KEY,
    dept_code    VARCHAR2(10) NOT NULL,
    safety_region NUMBER(2) CONSTRAINT REF_SafeRegions
    REFERENCES Safety_Region,
    location     MDSYS.SDO_GEOMETRY

```

```

);

CREATE TABLE DeptResp4Proc (
    departmentID      NUMBER(15) CONSTRAINT FK_RespDept
REFERENCES Department,
    incidentID        NUMBER(15),
    process_type      NUMBER(2),
    start_time        TIMESTAMP,
    end_time          TIMESTAMP,
    CONSTRAINT FK_ProcessCtrl FOREIGN KEY
    (incidentID, process_type) REFERENCES Process,
    CONSTRAINT PK_DeptResp4Proc PRIMARY KEY
    (departmentID, incidentID, process_type)
);

CREATE TABLE DM_SUser (
    userID            NUMBER(15) CONSTRAINT PK_User PRIMARY KEY,
    userBSN          CHAR(9) NOT NULL,
    deptID           NUMBER(15) CONSTRAINT FK_UserDept
REFERENCES Department
);

CREATE TABLE User_In_Process (
    userID            NUMBER(15) CONSTRAINT FK_User_InProcess
REFERENCES DM_SUser,
    incidentID       NUMBER(15),
    process_type     NUMBER(2),
    role             NUMBER(3) CONSTRAINT REF_UserRole
REFERENCES User_Role,
    start_time       TIMESTAMP,
    end_time         TIMESTAMP,
    CONSTRAINT FK_UserIn_Process FOREIGN KEY
    (incidentID, process_type) REFERENCES Process,
    CONSTRAINT PK_UserInProcess
    PRIMARY KEY (userID, incidentID, process_type)
);

CREATE TABLE Vehicle (
    vehicleID        NUMBER(15) CONSTRAINT PK_Vehicle PRIMARY KEY,
    vehicle_code     VARCHAR2(10) NOT NULL,
    deptID           NUMBER(15) CONSTRAINT FK_Vehicle2Dept
REFERENCES Department,
    calculatedETA    TIMESTAMP
);

/* Team tables */

CREATE TABLE Team (
    teamID           NUMBER(20) CONSTRAINT PK_Team PRIMARY KEY,
    name            VARCHAR2(20),
    team_type       NUMBER(2) CONSTRAINT REF_Team
REFERENCES Team_Type,
    no_members      NUMBER(2),
    leaderID        NUMBER(15) CONSTRAINT FK_Team_Leader
REFERENCES DM_SUser NOT NULL,
    vehicleID       NUMBER(15) CONSTRAINT FK_Travel_With
REFERENCES Vehicle
);

CREATE TABLE TeamPosition (
    teamID           NUMBER(15) CONSTRAINT FK_Team_Position
REFERENCES Team,
    meas_time       TIMESTAMP,
    position         MDSYS.SDO_GEOMETRY,
    CONSTRAINT PK_TeamPosition PRIMARY KEY (teamID, meas_time)
);

```

```

);

CREATE TABLE Measurement (
    taskID          NUMBER(20)    CONSTRAINT PK_Measurement PRIMARY
KEY,
    incidentID     NUMBER(15)    CONSTRAINT FK_Measurement_Incident
REFERENCES RealIncident ON DELETE SET NULL,
    AGSuserID      NUMBER(15)    CONSTRAINT FK_AGS_userID
REFERENCES DM_SUser,
    location       MDSYS.SDO_GEOMETRY NOT NULL,
    assignment_time  TIMESTAMP NOT NULL,
    task_explosionLEL CHAR(1)    CONSTRAINT CHK_explosion_task
CHECK (task_explosionLEL in ('Y', 'N')),
    gas_tube       BINARY_FLOAT,
    task_automess   CHAR(1)    CONSTRAINT CHK_automess_task
CHECK (task_automess in ('Y', 'N')),
    task_automess_sonda CHAR(1)
CONSTRAINT CHK_automess_sonda_task
CHECK (task_automess_sonda in ('Y', 'N')),
    task_dosimeter CHAR(1)    CONSTRAINT CHK_dosimeter_task
CHECK (task_dosimeter in ('Y', 'N')),
    protection     CHAR(1)    CONSTRAINT CHK_protection_task
CHECK (protection in ('Y', 'N')),
    teamID         NUMBER(20)    CONSTRAINT FK_Measurement_Team
REFERENCES Team ON DELETE SET NULL,
    measurement_time  TIMESTAMP NOT NULL,
    explosionLEL     BINARY_FLOAT,
    gas_tube_no      NUMBER(2)    CONSTRAINT REF_Bravo
REFERENCES Bravo_Measure(code),
    pumping_no      NUMBER(4),
    concentration   BINARY_FLOAT,
    automess1       BINARY_FLOAT,
    automess2       BINARY_FLOAT,
    automess_sonda  BINARY_FLOAT,
    dosimeter       BINARY_FLOAT,
    fed2gasmal     TIMESTAMP,
    task_details    VARCHAR2(400),
    meas_details    VARCHAR2(400),
    CONSTRAINT CHK_explosion CHECK (task_explosionLEL <> 'Y' OR
explosionLEL IS NOT NULL) DISABLE,
    CONSTRAINT CHK_gas_tube CHECK (gas_tube IS NULL OR
(gas_tube_no IS NOT NULL AND pumping_no IS NOT NULL AND
concentration IS NOT NULL)) DISABLE,
    CONSTRAINT CHK_automess CHECK (task_automess <> 'Y' OR
(automess1 IS NOT NULL AND automess2 IS NOT NULL)) DISABLE,
    CONSTRAINT CHK_automess_sonda CHECK
(task_automess_sonda <> 'Y'
OR automess_sonda IS NOT NULL) DISABLE,
    CONSTRAINT CHK_dosimeter CHECK (task_dosimeter <> 'Y' OR
dosimeter IS NOT NULL) DISABLE
);

/* Data in this table will be stored in pieces: first the
measurement task, then the measurements for the given task id.
Constraints on the measurements are disabled because the fields
will be empty at first. They can be checked later with ALTER TABLE
Measurement ENABLE VALIDATE CONSTRAINT CHK_explosion for the
CHK_explosion constraint; similar for all others */

CREATE TABLE EventObject (
    objectId      NUMBER(20)    CONSTRAINT PK_EventObject PRIMARY
KEY,
    incidentID    NUMBER(15)    CONSTRAINT FK_Object_Incident
REFERENCES RealIncident NOT NULL,
    userID        NUMBER(15)    CONSTRAINT FK_Object_Creater

```

```

REFERENCES DM_SUser,
drawing_symbol NUMBER(2) CONSTRAINT REF_Symbol
REFERENCES Drawing_Symbol,
geometry MDSYS.SDO_GEOMETRY NOT NULL,
description VARCHAR2(200)
);

CREATE TABLE ReliefCentre (
centreID NUMBER(10) CONSTRAINT PK_RCentre PRIMARY KEY,
centre_type NUMBER(1) CONSTRAINT REF_RCentre
REFERENCES Relief_Centre,
capacity NUMBER(20),
location MDSYS.SDO_GEOMETRY
);

CREATE TABLE PatientCard (
patientID NUMBER(15) CONSTRAINT PK_PatientCard
PRIMARY KEY,
incidentID NUMBER(15) CONSTRAINT FK_Patient_Incident
REFERENCES RealIncident NOT NULL,
centreID NUMBER(10) CONSTRAINT FK_Patient_RCentre
REFERENCES ReliefCentre,
gender NUMBER(1) CONSTRAINT REF_Gender
REFERENCES Gender,
name VARCHAR2(50),
birthdate DATE,
address VARCHAR2(50),
IDcard VARCHAR2(50),
tel_family VARCHAR2(50),
place_found VARCHAR2(50),
allergy VARCHAR2(50),
medication VARCHAR2(50),
past_MR MEDICAL_HISTORY,
last_meal VARCHAR2(50),
decontamination CHAR(1) CONSTRAINT CHK_decontamination
CHECK (decontamination in ('Y', 'N')),
accident-mech VARCHAR2(50),
head-diagnose VARCHAR2(50),
injury NUMBER(1) CONSTRAINT REF_Injury
REFERENCES Injury_Type,
left_pupilR CHAR(1) CONSTRAINT CHK_LPupil
CHECK (left_pupilR in ('Y', 'N')),
right_pupilR CHAR(1) CONSTRAINT CHK_RPupil
CHECK (right_pupilR in ('Y', 'N')),
notes VARCHAR2(50),
triage_record TRIAGE,
triage_class NUMBER(1) CONSTRAINT REF_Triage
REFERENCES Triage_Val,
treatment TREATMENT,
remarks VARCHAR2(50)
);

/* Casualty tables */

CREATE TABLE PsychologyPat (
incidentID NUMBER(15) CONSTRAINT FK_IncPsychologicP
REFERENCES RealIncident,
meas_time TIMESTAMP,
psychologicP NUMBER(15),
CONSTRAINT PK_Psychologic PRIMARY KEY (incidentID, meas_time)
);

CREATE TABLE Triage1 (
incidentID NUMBER(15) CONSTRAINT FK_Inc_Triage1
REFERENCES RealIncident,

```

```

        meas_time    TIMESTAMP,
        triage1      NUMBER(15),
        CONSTRAINT PK_Triage1 PRIMARY KEY (incidentID, meas_time)
    );
CREATE TABLE Triage2 (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Triage2
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    triage2      NUMBER(15),
    CONSTRAINT PK_Triage2 PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE Triage3 (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Triage3
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    triage3      NUMBER(15),
    CONSTRAINT PK_Triage3 PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE Triage3n (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Triage3n
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    triage3n     NUMBER(15),
    CONSTRAINT PK_Triage3n PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE Triage4 (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Triage4
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    triage4      NUMBER(15),
    CONSTRAINT PK_Triage4 PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE Death (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Death
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    death        NUMBER(15),
    CONSTRAINT PK_Death PRIMARY KEY (incidentID, meas_time)
);

/* Tables of damage on people&animals */

CREATE TABLE Trapped (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Trapped
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    trapped      NUMBER(15),
    CONSTRAINT PK_Trapped PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE Missing (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_Missing
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    missing      NUMBER(15),
    CONSTRAINT PK_Missing PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE P2Evacuate (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_P2Evacuate
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    people2evac NUMBER(15),
    CONSTRAINT PK_P2Evacuate PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE P2decont (

```



```

        incidentID NUMBER(15) CONSTRAINT FK_Inc_P2decont
        REFERENCES RealIncident,
        meas_time    TIMESTAMP,
        people2decontam    NUMBER(15),
        CONSTRAINT PK_P2decont PRIMARY KEY (incidentID, meas_time)
    );
CREATE TABLE A2decont (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_A2decont
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    animal2decontam    NUMBER(15),
    CONSTRAINT PK_A2decont PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE P4shelter (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_P4shelter
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    people4shelter    NUMBER(15),
    CONSTRAINT PK_P4shelter PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE P2feed (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_P2feed
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    people2feed NUMBER(15),
    CONSTRAINT PK_P2feed PRIMARY KEY (incidentID, meas_time)
);
CREATE TABLE A2feed (
    incidentID NUMBER(15) CONSTRAINT FK_Inc_A2feed
    REFERENCES RealIncident,
    meas_time    TIMESTAMP,
    animal2feed NUMBER(15),
    CONSTRAINT PK_A2feed PRIMARY KEY (incidentID, meas_time)
);

CREATE TABLE DamagedBuilding (
    incidentID NUMBER(15) CONSTRAINT FK_DamageBuild_Incident
    REFERENCES RealIncident,
    BAGcode VARCHAR2(20),
    CONSTRAINT PK_DamageBuild PRIMARY KEY (incidentID)
);

CREATE TABLE DamagedCar (
    incidentID NUMBER(15) CONSTRAINT FK_DamageCar_Incident
    REFERENCES RealIncident,
    plate_no VARCHAR2(15),
    CONSTRAINT PK_DamageCar PRIMARY KEY (incidentID)
);

```

Reports published before in this series

1. GISSt Report No. 1, Oosterom, P.J. van, Research issues in integrated querying of geometric and thematic cadastral information (1), Delft University of Technology, Rapport aan Concernstaf Kadaster, Delft 2000, 29 p.p.
2. GISSt Report No. 2, Stoter, J.E., Considerations for a 3D Cadastre, Delft University of Technology, Rapport aan Concernstaf Kadaster, Delft 2000, 30.p.
3. GISSt Report No. 3, Fendel, E.M. en A.B. Smits (eds.), Java GIS Seminar, Opening GDMC, Delft 15 November 2000, Delft University of Technology, GISSt. No. 3, 25 p.p.
4. GISSt Report No. 4, Oosterom, P.J.M. van, Research issues in integrated querying of geometric and thematic cadastral information (2), Delft University of Technology, Rapport aan Concernstaf Kadaster, Delft 2000, 29 p.p.
5. GISSt Report No. 5, Oosterom, P.J.M. van, C.W. Quak, J.E. Stoter, T.P.M. Tijssen en M.E. de Vries, Objectgerichtheid TOP10vector: Achtergrond en commentaar op de gebruikersspecificaties en het conceptuele gegevensmodel, Rapport aan Topografische Dienst Nederland, E.M. Fendel (eds.), Delft University of Technology, Delft 2000, 18 p.p.
6. GISSt Report No. 6, Quak, C.W., An implementation of a classification algorithm for houses, Rapport aan Concernstaf Kadaster, Delft 2001, 13.p.
7. GISSt Report No. 7, Tijssen, T.P.M., C.W. Quak and P.J.M. van Oosterom, Spatial DBMS testing with data from the Cadastre and TNO NITG, Delft 2001, 119 p.
8. GISSt Report No. 8, Vries, M.E. de en E. Verbree, Internet GIS met ArcIMS, Delft 2001, 38 p.
9. GISSt Report No. 9, Vries, M.E. de, T.P.M. Tijssen, J.E. Stoter, C.W. Quak and P.J.M. van Oosterom, The GML prototype of the new TOP10vector object model, Report for the Topographic Service, Delft 2001, 132 p.
10. GISSt Report No. 10, Stoter, J.E., Nauwkeurig bepalen van grondverzet op basis van CAD ontgravingsprofielen en GIS, een haalbaarheidsstudie, Rapport aan de Bouwdienst van Rijkswaterstaat, Delft 2001, 23 p.
11. GISSt Report No. 11, Geo DBMS, De basis van GIS-toepassingen, KvAG/AGGN Themamiddag, 14 november 2001, J. Flim (eds.), Delft 2001, 37 p.
12. GISSt Report No. 12, Vries, M.E. de, T.P.M. Tijssen, J.E. Stoter, C.W. Quak and P.J.M. van Oosterom, The second GML prototype of the new TOP10vector object model, Report for the Topographic Service, Delft 2002, Part 1, Main text, 63 p. and Part 2, Appendices B and C, 85 p.
13. GISSt Report No. 13, Vries, M.E. de, T.P.M. Tijssen en P.J.M. van Oosterom, Comparing the storage of Shell data in Oracle spatial and in Oracle/ArcSDE compressed binary format, Delft 2002, .72 p. (Confidential)
14. GISSt Report No. 14, Stoter, J.E., 3D Cadastre, Progress Report, Report to Concernstaf Kadaster, Delft 2002, 16 p.
15. GISSt Report No. 15, Zlatanova, S., Research Project on the Usability of Oracle Spatial within the RWS Organisation, Detailed Project Plan (MD-NR. 3215), Report to Meetkundige Dienst – Rijkswaterstaat, Delft 2002, 13 p.
16. GISSt Report No. 16, Verbree, E., Driedimensionale Topografische Terreinmodellering op basis van Tetraëder Netwerken: Top10-3D, Report aan Topografische Dienst Nederland, Delft 2002, 15 p.
17. GISSt Report No. 17, Zlatanova, S. Augmented Reality Technology, Report to SURFnet bv, Delft 2002, 72 p.
18. GISSt Report No. 18, Vries, M.E. de, Ontsluiting van Geo-informatie via netwerken, Plan van aanpak, Delft 2002, 17p.
19. GISSt Report No. 19, Tijssen, T.P.M., Testing Informix DBMS with spatial data from the cadastre, Delft 2002, 62 p.
20. GISSt Report No. 20, Oosterom, P.J.M. van, Vision for the next decade of GIS technology, A research agenda for the TU Delft the Netherlands, Delft 2003, 55 p.
21. GISSt Report No. 21, Zlatanova, S., T.P.M. Tijssen, P.J.M. van Oosterom and C.W. Quak, Research on usability of Oracle Spatial within the RWS organisation, (AGI-GAG-2003-21), Report to Meetkundige Dienst – Rijkswaterstaat, Delft 2003, 74 p.
22. GISSt Report No. 22, Verbree, E., Kartografische hoogtevoorstelling TOP10vector, Report aan Topografische Dienst Nederland, Delft 2003, 28 p.
23. GISSt Report No. 23, Tijssen, T.P.M., M.E. de Vries and P.J.M. van Oosterom, Comparing the storage of Shell data in Oracle SDO_Geometry version 9i and version 10g Beta 2 (in the context of ArcGIS 8.3), Delft 2003, 20 p. (Confidential)
24. GISSt Report No. 24, Stoter, J.E., 3D aspects of property transactions: Comparison of registration of 3D properties in the Netherlands and Denmark, Report on the short-term scientific mission in the CIST – G9 framework at the Department of Development and Planning, Center of 3D geo-information, Aalborg, Denmark, Delft 2003, 22 p.
25. GISSt Report No. 25, Verbree, E., Comparison Gridding with ArcGIS 8.2 versus CPS/3, Report to Shell International Exploration and Production B.V., Delft 2004, 14 p. (confidential).
26. GISSt Report No. 26, Penninga, F., Oracle 10g Topology, Testing Oracle 10g Topology with cadastral data, Delft 2004, 48 p.
27. GISSt Report No. 27, Penninga, F., 3D Topography, Realization of a three dimensional topographic terrain representation in a feature-based integrated TIN/TEN model, Delft 2004, 27 p.
28. GISSt Report No. 28, Penninga, F., Kartografische hoogtevoorstelling binnen TOP10NL, Inventarisatie mogelijkheden op basis van TOP10NL uitgebreid met een Digitaal Hoogtemodel, Delft 2004, 29 p.

29. GISSt Report No. 29, Verbree, E. en S.Zlatanova, 3D-Modeling with respect to boundary representations within geo-DBMS, Delft 2004, 30 p.
30. GISSt Report No. 30, Penninga, F., Introductie van de 3e dimensie in de TOP10NL; Voorstel voor een onderzoekstraject naar het stapsgewijs introduceren van 3D data in de TOP10NL, Delft 2005, 25 p.
31. GISSt Report No. 31, P. van Asperen, M. Grothe, S. Zlatanova, M. de Vries, T. Tijssen, P. van Oosterom and A. Kabamba, Specificatie datamodel Beheerkaart Nat, RWS-AGI report/GISSt Report, Delft, 2005, 130 p.
32. GISSt Report No. 32, E.M. Fendel, Looking back at Gi4DM, Delft 2005, 22 p.
33. GISSt Report No. 33, P. van Oosterom, T. Tijssen and F. Penninga, Topology Storage and the Use in the context of consistent data management, Delft 2005, 35 p.
34. GISSt Report No. 34, E. Verbree en F. Penninga, RGI 3D Topo - DP 1-1, Inventarisatie huidige toegankelijkheid, gebruik en mogelijke toepassingen 3D topografische informatie en systemen, 3D Topo Report No. RGI-011-01/GISSt Report No. 34, Delft 2005, 29 p.
35. GISSt Report No. 35, E. Verbree, F. Penninga en S. Zlatanova, Datamodellering en datastructurering voor 3D topografie, 3D Topo Report No. RGI-011-02/GISSt Report No. 35, Delft 2005, 44 p.
36. GISSt Report No. 36, W. Looijen, M. Uitentuis en P. Bange, RGI-026: LBS-24-7, Tussenrapportage DP-1: Gebruikerswensen LBS onder redactie van E. Verbree en E. Fendel, RGI LBS-026-01/GISSt Rapport No. 36, Delft 2005, 21 p.
37. GISSt Report No. 37, C. van Strien, W. Looijen, P. Bange, A. Wilcsinszky, J. Steenbruggen en E. Verbree, RGI-026: LBS-24-7, Tussenrapportage DP-2: Inventarisatie geo-informatie en -services onder redactie van E. Verbree en E. Fendel, RGI LBS-026-02/GISSt Rapport No. 37, Delft 2005, 21 p.
38. GISSt Report No. 38, E. Verbree, S. Zlatanova en E. Wisse, RGI-026: LBS-24-7, Tussenrapportage DP-3: Specifieke wensen en eisen op het gebied van plaatsbepaling, privacy en beeldvorming, onder redactie van E. Verbree en E. Fendel, RGI LBS-026-03/GISSt Rapport No. 38, Delft 2005, 15 p.
39. GISSt Report No. 39, E. Verbree, E. Fendel, M. Uitentuis, P. Bange, W. Looijen, C. van Strien, E. Wisse en A. Wilcsinszky en E. Verbree, RGI-026: LBS-24-7, Eindrapportage DP-4: Workshop 28-07-2005 Geo-informatie voor politie, brandweer en hulpverlening ter plaatse, RGI LBS-026-04/GISSt Rapport No. 39, Delft 2005, 18 p.
40. GISSt Report No. 40, P.J.M. van Oosterom, F. Penninga and M.E. de Vries, Trendrapport GIS, GISSt Report No. 40 / RWS Report AGI-2005-GAB-01, Delft, 2005, 48 p.
41. GISSt Report No. 41, R. Thompson, Proof of Assertions in the Investigation of the Regular Polytope, GISSt Report No. 41 / NRM-ISS090, Delft, 2005, 44 p.
42. GISSt Report No. 42, F. Penninga and P. van Oosterom, Kabel- en leidingnetwerken in de kadastrale registratie (in Dutch) GISSt Report No. 42, Delft, 2006, 38 p.
43. GISSt Report No. 43, F. Penninga and P.J.M. van Oosterom, Editing Features in a TEN-based DBMS approach for 3D Topographic Data Modelling, Technical Report, Delft, 2006, 21 p.
44. GISSt Report No. 44, M.E. de Vries, Open source clients voor UMN MapServer: PHP/Mapscript, JavaScript, Flash of Google (in Dutch), Delft, 2007, 13 p.
45. GISSt Report No. 45, W. Tegtmeier, Harmonization of geo-information related to the lifecycle of civil engineering objects – with focus on uncertainty and quality of surveyed data and derived real world representations, Delft, 2007, 40 p.
46. GISSt Report No. 46, W. Xu, Geo-information and formal semantics for disaster management, Delft, 2007, 31 p.
47. GISSt Report No. 47, E. Verbree and E.M. Fendel, GIS technology – Trend Report, Delft, 2007, 30 p.
48. GISSt Report No. 48, B.M. Meijers, Variable-Scale Geo-Information, Delft, 2008, 30 p.
49. GISSt Report No. 48, Maja Bitenc, Kajsa Dahlberg, Fatih Doner, Bas van Goort, Kai Lin, Yi Yin, Xiaoyu Yuan and Sisi Zlatanova, Utility Registration, Delft, 2008, 35 p.
50. GISSt Report No 50, T.P.M. Tijssen en S. Zlatanova, Oracle Spatial 11g en ArcGIS 9.2 voor het beheer van puntenwolken (Confidential), Delft, 2008, 16 p.
51. GISSt Report No. 51, S. Zlatanova, Geo-information for Crisis Management, Delft, 2008, 24 p.
52. GISSt Report No. 52, P.J.M. van Oosterom, INSPIRE activiteiten in het jaar 2008 (partly in Dutch), Delft, 2009, 142 p.
53. GISSt Report No. 53, P.J.M. van Oosterom with input of and feedback by Rod Thompson and Steve Huch (Department of Environment and Resource Management, Queensland Government), Delft, 2010, 60 p.

