

SemSorGrid4Env

FP7-223913



Deliverable

D2.1

Data Requirements, Data Management and Analysis Issues, and Query-Based Functionalities

Ixent Galpin (Editor), Alasdair J G Gray, Alvaro A A Fernandes, and Norman
W Paton
University of Manchester

Alexis Kotsifakos, Dimitris Kotsakos, and Dimitrios Gunopulos
National and Kapodistrian University of Athens

August 28, 2009



Executive Summary

A recent development, catalysed by advances in radio, processor and battery technology, is the emergence of *sensor networks* [KW05] as an economically viable hardware platform with which to observe or monitor phenomena *in situ*, possibly over a wide area of interest and at a finer grain of observation than was previously possible. However, it is currently not straightforward for a remote user to discover relevant sensor networks, and integrate them into applications on-the-fly (even if the sensor network is accessible via the Internet) due to the heterogeneity of their interfaces to the outside world, and the inability to resolve semantic inconsistencies between the data attributes collected by sensor networks. The potential exists, but currently cannot be exploited, for sensor networks to be added to *ad-hoc*, lightweight applications or *mashups*, possibly generated on-the-fly, that can underpin decision support, e.g. in a disaster response scenario.

The *Semantic Sensor Grid Rapid Application Development for Environmental Management* project (SemSorGrid4Env) aims to respond to the challenges alluded to in the previous paragraph by developing techniques to enable sensor networks to be easily discoverable, integrated with each other, and also integrated with other types of data sources (e.g. a relational database with historical data); and enabling mashups to be developed, in an impromptu manner, that combine independent sensor networks and other types of data sources. This document is focused on the requirements that other work packages within the SemSorGrid4Env project have for WP2, the data management work package assigned to the University of Manchester (UNIMAN) and the National Kapodistrian University of Athens (NKUA), and the work which will be undertaken within WP2 in order to fulfil these requirements.

Firstly, this deliverable identifies the functionality required by other work packages in the SemSorGrid4Env project, so that suitable software and techniques, comprising the WP2 contributions to SemSorGrid4Env, can be proposed. For the application use-cases, various requirements are identified, such as the ability to access data from both stored sources (e.g. historical data) and streaming sources (e.g. the SN deployed at the site of interest), the ability to evaluate queries over acquisitional streams within sensor networks, as well as event streams within robust networks, and the capability to perform data analysis techniques (e.g., outlier detection, for data cleaning, or to identify an event of interest such as a forest fire). For the technical work-packages, it is identified that data sources must register with, and provide metadata to, the registry (WP3); data sources must provide a common access mechanism for the semantic integrator developed as part of WP4; and for WP5, data sources may need to provide a REST API in addition to the service-oriented interface described in the SemSorGrid4Env architecture [GGF⁺09].

Based on the requirements identified from the other work packages, existing techniques and software are described that WP2 proposes to use as basis for its contributions to SemSorGrid4Env. These include data analysis algorithms, previously developed at NKUA, for outlier detection [SPP⁺06]; OGSA-DAI, off-the-shelf software that provides access to stored data sources via a service-based interface compliant with the WS-DAI standard [AAK⁺06, ACK⁺06], adopted by the SemSorGrid4Env architecture [GGF⁺09]; and SNEE, the sensor network query optimiser developed at UNIMAN, that evaluates queries expressed in SNEEqL, a rich, expressive continuous query language over sensor networks.

Finally, this deliverable presents a functional decomposition of tasks required in order to meet the requirements previously identified. These tasks include: Implementation of the D3 outlier detection algorithm as software for the SemSorGrid4Env infrastructure; Enabling query processing over both robust and sensor network nodes, and both acquisitional and event streams, as well as stored data; Enabling SNEE to define views to support data integration; Enabling QoS-awareness and multiple query execution; Integrating data analysis techniques so that they can be expressed seamlessly in the query language; and, developing appropriate network management techniques to support the above functionality. These tasks will be scheduled to ensure that WP2 contributes to the overall success of the SemSorGrid4Env project.



Note on Sources and Original Contributions

The SemSorGrid4Env consortium is an inter-disciplinary team, and in order to make deliverables self-contained and comprehensible to all partners, some deliverables thus necessarily include state-of-the-art surveys and associated critical assessment. Where there is no advantage to recreating such materials from first principles, partners follow standard scientific practice and occasionally make use of their own pre-existing intellectual property in such sections. In the interests of transparency, we here identify the main sources of such pre-existing materials in this deliverable:

- Section 4.1 contains material from [SPP⁺06]
- Section 4.3 contains material adapted from [GBJ⁺09].



Document Information






Contract Number	FP7-223913	Acronym	SemSorGrid4Env
Full title	SemSorGrid4Env: Semantic Sensor Grids for Rapid Application Development for Environmental Management		
Project URL	www.semsorgrid4env.eu		
Document URL	www.semsorgrid4env.eu/sem/viewTerm?content=instance.jsp&_sew_var_name=instance&_sew_instance=D2.1&_sew_instance_set=SemSorGrid4Env&_origin=%2Fhome.jsp		
EU Project Officer	Gaëlle Le Gars		

Deliverable	Number	D2.1	Name	Data Requirements, Data Management and Analysis Issues, and Query-Based Functionalities		
Task	Number	T2.1, T2.2	Name	T2.1: Develop Data Management and Analysis Techniques for Streams. T2.2: Generalise the current UNIMAN QP to Support Event Streams and View Definition Capabilities.		
Work package	Number			WP2		
Date of delivery	Contract	31 August 2009		Actual	31 August 2009	
Code name	D2.1		Status	draft <input type="checkbox"/>	final <input checked="" type="checkbox"/>	
Nature	Prototype <input type="checkbox"/> Report <input checked="" type="checkbox"/> Specification <input type="checkbox"/> Tool <input type="checkbox"/> Other <input type="checkbox"/>					
Distribution Type	Public <input checked="" type="checkbox"/> Restricted <input type="checkbox"/> Consortium <input type="checkbox"/>					
Authoring Partner	UNIMAN					
QA Partner	UPM					
Contact Person	Ixent Galpin					
	Email	Ixent@cs.man.ac.uk		Phone	+44 161 275 6132	Fax +44 161 275 6204
Abstract (for dissemination)	<p>The potential exists, but currently cannot be exploited, for sensor networks to be added to <i>ad-hoc</i>, lightweight applications or <i>mashups</i>, possibly generated on-the-fly, that can underpin decision support, e.g. in a disaster response scenario. The <i>Semantic Sensor Grid Rapid Application Development for Environmental Management</i> project (SemSorGrid4Env) aims to respond to this challenge, by developing techniques to enable sensor networks to be easily discoverable, integrated with each other, and also integrated with other types of data sources (e.g. a relational database with historical data); and enabling mashups to be developed, in an impromptu manner, that combine independent sensor networks and other types of data sources. This document is focused on the requirements that other work packages within the SemSorGrid4Env project have for WP2, the data management work package assigned to the University of Manchester (UNIMAN) and the National Kapodistrian University of Athens (NKUA), and the work which will be undertaken within WP2 in order to fulfil these requirements.</p> <p>This deliverable identifies the functionality required from WP2 by other work packages in the SemSorGrid4Env project, so that suitable software and techniques, comprising the WP2 contributions to SemSorGrid4Env, can be proposed. For the application use-cases, various requirements are identified, such as the ability to access stored and streaming data, the ability to evaluate queries over acquisitional and event streams, and the capability to perform data analysis techniques (e.g., outlier detection, for data cleaning, or to identify an event of interest such as a forest fire). In light of these requirements, existing techniques and software, such as the SNEE query optimiser for sensor networks and the D3 outlier detection algorithm are proposed as the basis for the WP2 contributions, and a functional decomposition of the tasks to be undertaken is presented.</p>					
Keywords	Data Management, Data Analysis					

Version log/Date	Change	Author
0.1 / 29 May 2009	Initial Draft	I. Galpin, A. Gray, A. Kotsifakos and D. Kotsakos
0.2 / 10 July 2009	Restructured document and revised text accordingly	I. Galpin, A. Gray, A. Kotsifakos and D. Kotsakos
0.3 / 31 July 2009	Edits based on feedback.	I. Galpin, A. Gray, A. Kotsifakos and D. Kotsakos
1.0 / 28 August 2009	Edits based on QA. Redraft of background chapter.	I. Galpin, A. Gray, A. Kotsifakos and D. Kotsakos

Project Information

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number FP7-223913. The Beneficiaries in this project are:

Partner	Acronym	Contact
Universidad Politécnica de Madrid (Coordinator)	UPM 	Prof. Dr. Asunción Gómez-Pérez Facultad de Informática Departamento de Inteligencia Artificial Campus de Montegancedo, sn Boadilla del Monte 28660 Spain #@ asun@fi.upm.es #t +34-91 336-7439, #f +34-91 352-4819
The University of Manchester	UNIMAN 	Prof. Carole Goble Department of Computer Science The University of Manchester Oxford Road Manchester, M13 9PL, United Kingdom #@ carole@cs.man.ac.uk #t +44-161-275-61 95, # +44-161-275 62 04
National and Kapodistrian University of Athens	NKUA 	Prof. Manolis Koubarakis University Campus, Ilissia Athina GR-15784 Greece #@ koubarak@di.uoa.gr #t +30 210 7275213, #f +30 210 7275214
University of Southampton	SOTON 	Prof. David De Roure University Road Southampton SO17 1BJ United Kingdom #@ dder@ecs.soton.ac.uk #t +44 23 80592418, #f +44 23 80595499
Deimos Space, S.L.	DMS 	Mr. Agustín Izquierdo Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2º Tres Cantos, Madrid – 28760 Spain #@agustin.izquierdo@deimos-space.com #t +34-91-8063450, #f +34-91-806-34-51
EMU Limited	EMU 	Dr. Bruce Tomlinson Mill Court, The Sawmills, Durley number 1 Southampton, SO32 2EJ, United Kingdom #@ bruce.tomlinson@emulimited.com #t +44 1489 860050, #f +44 1489 860051
TechIdeas Asesores Tecnológicos, S.L.	TI 	Mr. Jesús E. Gabaldón C/ Marie Curie 8-14 08042 Barcelona, Spain #@ jesus.gabaldon@techideas.es #t +34.93.291.77.27, #f +34.93.291.76.00



Contents

Glossary	ix
1 Introduction	1
2 Background	4
2.1 Data Management	4
2.1.1 Stored Data Sources	4
2.1.2 Streaming Data Sources	5
2.2 Query Processing	6
2.2.1 Classical Query Processing	6
2.2.2 Distributed Query Processing	7
2.2.3 Stream Query Processing	8
2.2.4 Data Integration	9
2.3 Data Analysis	10
2.3.1 Data Modelling	10
2.3.2 Data Classification	10
2.3.3 Data Clustering	11
2.3.4 Outlier Detection	12
2.3.5 Sampling	14
2.4 Summary	15
3 Requirements	16
3.1 Requirements from the Use Cases	16
3.1.1 Fire Risk Monitoring and Warning	16
3.1.2 Coastal and Estuarine Flood Warning in Southern UK	17
3.2 Requirements of the Technical Work Packages	18
3.2.1 Architecture and Infrastructure	18
3.2.2 Registry	18
3.2.3 Semantic Data Integration	19
3.2.4 High-level Application Programming Interfaces	19
4 Existing Work	20
4.1 In-network and Centralised Data Analysis	20
4.1.1 Motivation for Outlier Detection and Data Sampling	20
4.1.2 In-network Outlier Detection	20
4.1.3 Centralised Outlier Detection	27
4.2 Query Processing over Stored Data	28
4.3 In-network Query Processing	28
4.3.1 Query Language	29
4.3.2 Query Stack	30
4.4 Summary	35
5 Architecture	36
5.1 Architecture Overview	36
5.2 Data Management Interfaces	37
5.3 Data Management Services	38
5.3.1 Stored Data Service	38
5.3.2 Streaming Data Service	39
5.4 Meeting the Requirements	40
5.4.1 Meeting the Requirements from the Use Cases	41
5.4.2 Meeting the Requirements from the Technical Work Packages	41
5.5 Summary	42



6	Future Work	43
6.1	Implementation of Data Analysis Techniques within Sensor Networks and Robust Networks	44
6.2	Enabling Unified Query Processing over Sensor Network and Robust Networks . .	45
6.3	Enabling Unified Query Processing over Acquisitional Streams, Event Streams and Relations	46
6.4	Enabling Views over Streaming and Stored Data in Both Sensor and Robust Networks	46
6.5	Enabling QoS-aware Query Processing in Sensor and Robust Networks	47
6.6	Enabling Multiple Query Execution	48
6.7	Integrating Data Analysis Techniques into Continuous Queries	49
6.8	Design and Implement Sensor Network Management Capabilities to Support the Above	51
6.9	Summary	52



List of Figures

- 2.1 Classification of clustering methods. 12
- 4.1 Hierarchical Organisation of the Sensor Network 21
- 4.2 Distributed Deviation Detection (D3) algorithm 23
- 4.3 Leaf Process algorithm 23
- 4.4 Parent Process algorithm 23
- 4.5 Outlier algorithm 24
- 4.6 Example of the sampling and the counting neighbourhood. 25
- 4.7 For observation P , the counting neighbourhood is estimated by querying with $N(p, \alpha r)$.
The domain of the $1 - d$ observations is divided in intervals of width $2\alpha r$, and the
number of points in the i th interval is estimated with a range query $N(\alpha r(2i - 1), \alpha r)$. 26
- 4.8 Multi Granular Deviation Detection 26
- 4.9 Leaf Process algorithm 26
- 4.10 Black Process algorithm 26
- 4.11 Outlier algorithm 27
- 4.12 Example SNEEq schema definition. 29
- 4.13 Queries and QoS expectations. 29
- 4.14 SNEEq Compiler/Optimiser Stack. 31
- 4.15 The sensor network. 32
- 4.16 Schema sources. 32
- 4.17 Physical-algebraic form for $Q3$ 33
- 4.18 Routing tree for $Q3$ 33
- 4.19 Distributed-algebraic form for $Q3$ 34
- 4.20 The agenda for $Q3$ 35
- 5.1 The SemSorGrid4Env software architecture. 37
- 5.2 The interfaces exposed by the data tier services. 38
- 6.1 Model definitions, and queries which use the models to detect outliers. 50



List of Tables

2.1	Declarative query languages corresponding to stored data models.	7
3.1	Data Management Dimensions of SemSorGrid4Env Applications.	17



Glossary

API Application Programmer Interface. A set of defined method calls.

BIRCH Balanced Iterative Reducing and Clustering using Hierarchies.

CLARA Clustering Large Applications.

CLARANS Clustering Large Applications based upon RANdomized Search.

CLIQUE CLustering In QUest.

CSV Comma Separated Values. A file format for representing data.

CURE Clustering Using Representatives.

DAF Distributed Algebraic Form.

DBMS Database Management System.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

DDL Data definition language.

DENCLUE DENsity-based CLUstEring.

DQP Distributed Query Processor.

DSMS Data Stream Management System.

GAV Global-as-View. A mechanism for mapping a global schema to the data source schemas.

GLAV Global-Local-as-View. A mechanism for mapping a global schema to the data source schemas.

HTTP Hypertext Transfer Protocol.

JDBC Java Database Connectivity an API for accessing relational databases from Java applications.

k-NN k-nearest neighbours. A mechanism for classifying data.

LAV Local-as-View. A mechanism for mapping a global schema to the data source schemas.

NKUA National Kapodistrian University of Athens.

OGSA-DAI Open Grid Services Architecture Database Access and Integration Services.

OQL Object Query Language for querying object-oriented databases.

OWL The Web Ontology Language. A W3C standard for specifying ontologies.

PAF Physical Algebraic Form.

PAM Partitioning Around Medoids.

QEP Query Execution Plan.

QoS Quality of Service.



RDBMS Relational Database Management System.

RDF The Resource Description Framework. A W3C standard originally designed for modelling metadata about Web resources..

REST Representational State Transfer.

RT Routing Tree.

SNEE Sensor Network Engine. A query compiler for the SNEEqL language.

SNEEqL Sensor Network Evaluation Query Language for querying streams of data.

SPARQL SPARQL Protocol and RDF Query Language for querying RDF documents.

SQL Structured Query Language for querying relational databases.

STREAM The Stanford data stream management system.

stSPARQL Provides spatial and temporal extensions to SPARQL.

SVM Support Vector Machines. A mechanism for data classification.

UNIMAN University of Manchester.

URI Uniform Resource Identifier.

WP Work Package.

WS-* The group of Web service standards.

WS-DAI WS-Data Access and Integration [AAK⁺06].

WS-DAI-RDF The RDF realisation of WS-DAI [GGP08].

WS-DAI-Streaming The streaming realisation of WS-DAI.

WS-DAIR The relational realisation of WS-DAI [ACK⁺06].

WS-DAIX The XML realisation of WS-DAI [AHK⁺06].

WS-N WS-BaseNotification [GHM06].

WS-RF WS-ResourceFramework [Ban06].

XML Extensible Markup Language.



1. Introduction

A recent development, catalysed by advances in radio, processor and battery technology, is the emergence of *sensor networks* [KW05] as an economically viable hardware platform with which to observe or monitor phenomena *in situ*, possibly over a wide area of interest and at a finer grain of observation than was previously possible. Such networks are formed from sensor nodes working collaboratively on a task, which may involve data collection, event detection or entity tracking. Applications in a variety of domains have been proposed for sensor networks, including environmental monitoring, military, security and surveillance, precision agriculture, industrial applications, smart buildings, asset tracking and supply chain management, and health monitoring. The ability of sensor network nodes to gather information about their surrounding, process data, communicate with each other, and perform *actuation* (i.e. alter their surroundings in some way) has led to the idea that sensor networks conceptually enable the physical world itself to be viewed as a computing platform [BGS00].

Over recent years, the focus of research in sensor network data management has largely been focused at the intra-sensor network level. For example, numerous proposals have been made for energy-efficient techniques that perform data collection and transformation by exploiting the opportunity to process data at nodes *inside the sensor network* itself, thus trading off computations with more expensive radio communications, e.g. the TinyDB query processor [MFHH05] or the outlier detection techniques proposed in [SPP⁺06]. However, it is currently not straightforward for a remote user to discover relevant sensor networks, and integrate them into applications on-the-fly (even if the sensor network is accessible via the Internet) due to the heterogeneity of their interfaces to the outside world, and the inability to resolve semantic inconsistencies between the data attributes collected by sensor networks. The potential exists, but currently cannot be exploited, for sensor networks to be added to *ad-hoc*, lightweight applications or *mashups*, possibly generated on-the-fly, that can underpin decision support, e.g. in a disaster response scenario.

The *Semantic Sensor Grid Rapid Application Development for Environmental Management* project (SemSorGrid4Env) aims to respond to the challenges alluded to in the previous paragraph in a twofold manner:

1. By developing techniques to enable sensor networks to be easily discoverable, integrated with each other, and also integrated with other types of data sources (e.g. a relational database with historical data); and
2. Enabling mashups to be developed, in an impromptu manner, that combine independent sensor networks and other types of data sources.

In order to address these aims, the SemSorGrid4Env project has the following objectives, each of which falls under a work package (WP):

- To develop a software architecture to allow interoperability between data sources, data integrators, registries, and the application layer (WP1);
- To develop data management functionality to provide useful abstractions for accessing stored data, and for performing data integration (WP2);
- To develop a mechanism that enables data sources to be registered and, subsequently, discovered (WP3);
- To enable a semantically consistent view across data sources, using ontologies, to allow data integration to take place (WP4);
- To provide high-level application programming interfaces (APIs) for application developers, that enable mashups to be developed quickly and on-the-fly (WP5).



In order to validate the infrastructure developed as part of the SemSorGrid4Env project, two example use-case scenarios will be used to show-case applications, viz.:

1. A decision support system for firefighters to predict and detect fires in a forest area in Castilla y León, Spain, that will combine data collected by a sensor network with satellite images of the area of interest (WP6); and
2. A coastal and estuarine flood observation system that will combine live sensor network data with historical sources (WP7).

This document is focused on:

1. The requirements that other work packages within the SemSorGrid4Env project have for WP2, the data management work package assigned to the University of Manchester (UNIMAN) and the National Kapodistrian University of Athens (NKUA); and
2. The work which will be undertaken within WP2 in order to fulfill these requirements.

Within this document, two categories of streaming data source are identified: acquisitional (where data values are obtained on demand, e.g. sensor networks), and event streams (where data values are assumed to just arrive, perhaps at unpredictable rates, e.g. stock tickers). Processing on these streams can either take place within the sensor network, referred to as *in-network* processing, or it can take place outwith the sensor network on one or more computers. Within this document, this latter case is referred to as *robust-network* processing.

The structure of this document is now described.

In Chapter 2, we provide a description of the technical concepts that are required to understand the approach undertaken by WP2. This includes different modes of query processing (i.e. classical, distributed and streaming), data integration, methods for modelling data (i.e. schemas and ontologies), and different types of data analysis techniques (i.e. data modelling, classification, clustering, outlier detection and sampling). Then, sensor networks are described as a distributed computing platform, and the previously described techniques are discussed specifically in the context of sensor networks.

In Chapter 3, we describe various requirements from the technical work packages (WP1-WP5) and the application work packages (WP6 and WP7) that WP2 needs to satisfy. It is identified that relevant data sources may be stored, acquisitional (sensed) streams, or event streams, so mechanisms need to be provided to access all these types of data sources. In order to provide support for on-the-fly applications, the use of query processing techniques is proposed, which enable *ad-hoc* data requests to be made. They are also a useful means to underpin data integration. An overview of query processing is given in Section 2.2. As a simpler alternative mechanism to accessing streaming data sources, it is also identified that the ability to subscribe to a stream needs to be supported. Data analysis techniques that are useful for the use-cases are also identified; e.g. outlier detection can be useful to detect abnormal events, or for data cleaning. A background to data analysis techniques is provided in Section 2.3.

In Chapter 4, currently existing work that will be used to address the requirements is described. This comprises both existing theoretical techniques, to be implemented as software, and existing software, that will be used off-the-shelf and/or extended as appropriate. Firstly, techniques that will be used for performing data analysis are proposed for both streaming data and stored data, described in Section 4.1. For retrieving stored data resources, there is OGSA-DAI [OGS09], described in Section 4.2. The SNEE query processor [GBJ⁺09], which performs query processing within sensor networks, will be used for accessing acquisitional streams (Section 4.3).

In Chapter 5, the SemSorGrid4Env architecture, developed as part of WP1, is described. It adopts a service-oriented approach and seeks to allow functionality provided by services such as data



access, the registry, and data integrator to be composed into complex orchestrations. It shows how existing functionality and future work, described in Chapters 4 and 6 respectively, will be bundled into data access services that expose operations specified by the SemSorGrid4Env architecture, in order to enable interactions with the other work packages in the project.

Future work, to ensure that the requirements described in Chapter 3 will be met, are outlined in Chapter 6, viz.:

- The data analysis techniques described Section 4.1 will be implemented as query operations;
- These data analysis techniques will then be integrated with SNEE, so that this type of functionality can be specified as declarative statements within the query language;
- Extending SNEE with QoS-awareness, so that it can generate query plans in response to different optimisation goals;
- Extending SNEE with the ability to handle event streams, outside the sensor network, so that it can evaluate queries from the flood use-case;
- Extending SNEE with view definition capabilities, to enable it to be integrated with other data sources;
- Extending SNEE with the ability to execute multiple queries simultaneously;
- Incorporating the required network management functionality into SNEE so that it can run on the sensor hardware selected for the fire use-case.

For each task, an overview is given of the approach to be taken. The chapter concludes the report with a summary of the contributions to be made by WP2 in the SemSorGrid4Env project.



2. Background

This chapter provides a brief overview of the major technical concepts of Data Management (Section 2.1), Query Processing (Section 2.2), and Data Analysis (Section 2.3) as required by the reader to understand the technical details described in the rest of this document. Each section will provide a high-level overview of its subject and pointers to the literature for further details.

2.1 Data Management

In the context of the SemSorGrid4Env project, one can usefully distinguish two types of data source, viz. *stored* (e.g. files and databases) and *streaming* (e.g. monitoring data and sensor data). The following sections describe these two types of data sources and the data management technologies that have been developed to store and/or process the data made available in either case. Note, however, that the distinction between the two types is based on user perception. Thus, it is possible to have the same data managed by both types of data management technologies.

2.1.1 Stored Data Sources

The term *stored data* is used to refer to traditional forms of data management (e.g. files, documents, databases, etc) where data is persistently stored on some form of medium (e.g. magnetic disc) and assumed not to change whilst the data is accessed, thus implying, amongst other properties, a fixed cardinality for the data set for a given interaction.

In its simplest form, stored data can be provided as a file, either in an application specific format in which case it can only be acted upon by that application, or in some conventional structured form (e.g. as comma separated values (CSV)). The use of delimiters to structure data in files provides a convenient way to exchange data between applications and computer platforms. However, they are more difficult to interact with efficiently and are not suitable for processing large volumes of data. Thus, they are not considered within SemSorGrid4Env.

Data management systems have seen widespread use and been an active area of research from the early days of computing, when hierarchical data models were used, through to the present when relational databases (DBMS or RDBMS) [Cod70, GMWU99], object databases [ABD⁺89, CBB⁺00], XML databases [eXi, Xin07], and triple data stores [MM04, BKvH02, McB02] are used.

Relational database management systems are capable of storing large volumes of structured data. The data is conceptually organised into a *schema* consisting of a set of *relations*, commonly referred to as tables. Each relation consists of a set of *attributes*, commonly referred to as fields, each of which has an associated data type, e.g. integer. For each relation, it is possible to specify a *primary key*, in the form of a subset of the attributes of the relation, used to uniquely identify each row of data and for relating information between relations. The relational database management system also stores metadata about each relation, e.g. the cardinality of each relation (i.e. the number of rows of data), which it stores in its *catalog* so that it can be used to inform decisions regarding the most efficient execution of operations on the data. Relational database systems can generally support multiple simultaneous interactions on the data, some of which update the data or add or delete data items.

XML database systems such as eXist [eXi] and Xindice [Xin07] conceptually organise data into a *collection* of XML documents, where each document conforms to some schema generally expressed in XSchema [FW04]. XML documents are primarily used for representing semi-structured data. The data in an XML document is hierarchically structured: each document has a single root node



which can have multiple children forming different branches. Note however, that the branches may contain links to other branches in the same document which are not their descendant.

Triple store systems such as Jena [McB02] and Sesame [BKvH02] have been developed specifically for data conforming to the Resource Description Framework (RDF) [MM04]. RDF represents data as triples of the form:

subject predicate object,

where **subject** identifies some entity using a URI, **predicate** states some property that the entity has using a URI, and **object** provides a data value or the subject URI of another entity. Conceptually, the triples form a graph of data. RDF was initially developed as a data model for capturing metadata but has proven useful for providing self-describing data and managing unstructured data.

2.1.2 Streaming Data Sources

The term *streaming data* is used to refer to a source of data where the (constantly) changing data values are of interest to the user. That is, the user is interested in receiving the most recent value or observing the change of values over time. Traditional data management systems such as databases are not able to meet these requirements as they must store data before they can process them, and for some operations, e.g. sorting data, they rely on all the data being read before generating a result, which is not possible with a data stream. As such, data stream management systems (DSMS) have been proposed [GBJ⁺09, MFHH05, KCC⁺03, ACC⁺03, AAB⁺05a, ABB⁺09]. These systems implement techniques specifically for processing data streams, often in close to real-time.

Generally, a data stream consists of an infinite sequence of timestamped tuples of the form

$$(s, \tau),$$

where s is a tuple conforming to some schema and τ is a timestamp indicating the time at which the tuple was received. Note that two tuples may have the same timestamp value. In such a situation, some systems also add a *rank* attribute to provide some order for tuples with the same timestamp, e.g. Aurora [ACC⁺03] and SNEE [GBJ⁺09], while others assume that no order can be imposed, e.g. STREAM [ABB⁺09].

It is possible to distinguish two categories of streaming data source: acquisitional (where data values are obtained on demand, e.g. sensor networks), and event streams¹ (where data values are assumed to just arrive, perhaps at unpredictable rates, e.g. stock tickers). These different types of stream raise different challenges for a DSMS and are discussed in the following sections.

Acquisitional Streams

For acquisitional data streams, the DSMS controls when data is obtained from the data generator, generally a sensor in a sensor network. Advances in processor technologies and wireless communications have enabled the deployment of small, low cost, and power-efficient sensor nodes which consist of sensors that measure temperature, movement, noise, etc., and produce streams consisting of these readings [KW05]. A sensor network consists of multiple nodes that have sensing and wireless communication abilities, and can be used to monitor the physical environment as per the SemSorGrid4Env use cases [LDI09, CHSR09]. Recent improvements in hardware technology have allowed for wide-scale distribution of such networks.

Sensor nodes are usually battery-powered and communicate using wireless radio which impose constraints that must be taken into account when developing systems that run in sensor networks [YG03]. The most important constraint is *energy consumption*. Because of their limited

¹Sometimes referred to as “classical” data streams or non-acquisitional streams.



energy supply, each system designed to execute on sensor nodes must preserve energy. Another constraint stems from limited *computing and storage capabilities*. This makes it challenging to design complex data management algorithms that make use of complex data structures. Sensor readings are also *uncertain*. This means that one has to be careful when manipulating sensor data, because it is possible that noise is a significant concern (see Section 2.3 for more details). Finally, an important constraint for sensor networks stems from *limited communication resources*, which makes the delivery of high quality of service difficult.

Note that since the DSMS controls when data is gathered, it can infer certain characteristics of the data stream, such as its arrival rate and therefore allocate memory accordingly. However, due to the (generally) constrained nature of the data gathering device, careful management of processing and energy consumption must be considered [RSZ05, KW05, SMZ07]. SNEE [GBJ⁺09] and TinyDB [MFHH05] are examples of acquisitional DSMS that are able to perform computation within the sensor network.

Event Streams

For event streams, the DSMS generally runs on one or more nodes in a robust network and must cope with potentially bursty or high-rate, and often unpredictable arrival rates. As such, it cannot predict the memory and processing requirements, and may have to shed tuples. Various mechanisms have been presented in the literature to address the ensuing challenges [BBD⁺02, GÖ03, CG06, Agg07, GGR09], some of which have been implemented in systems such as STREAM [ABB⁺09], Aurora [ACÇ⁺03], Borealis [AAB⁺05a], and TelegraphCQ [KCC⁺03].

Note that it is also possible to have event streams with a potentially known data rate. For example, a stream of data generated by a sensor that is expected to take measurements every 15 minutes. In such a situation, the DSMS does not have control of the arrival of data, nor does it need to consider the resources of the sensor, but it can make assumptions for the efficient processing of data.

2.2 Query Processing

This section provides details about querying data, either stored in a DBMS, or made available as a data stream and accessible through a DSMS. It first covers query processing over stored data in a centralised setting before considering distributed query processing over stored data i.e. over a robust network. Details of processing queries over data streams are then presented considering both the case of event streams processed in a robust network and acquisitional streams processed in a sensor network. Finally, details are given of how data from multiple data sources can be combined using data integration techniques that exploit query processing over distributed data and semantic techniques.

2.2.1 Classical Query Processing

Classical query processing enables the evaluation of a *declarative query* posed over the schema of a stored data source, where the data is physically located and managed on a central server, i.e. a single node in a robust network. A declarative query does not specify *how* the query is to be evaluated, only *what* should be returned. Queries are expressed in a query language corresponding to the data model of the data source, as exemplified in Table 2.1.

Although we focus on the relational case in the following, similar approaches are followed for all stored data sources. The processing of a query involves two steps:

Storage Data Model	Declarative Query Language
Relational	SQL [SQL92]
XML	XPath [BBC ⁺ 07], XQuery [BCF ⁺ 07]
Object	OQL [CBB ⁺ 00]
Triple	SPARQL [Pe08]

Table 2.1: Declarative query languages corresponding to stored data models.

1. Parsing and compiling the query into an optimised query execution plan; and
2. Evaluating the execution plan to return results to the client.

First the query is parsed to generate a *logical query plan*, which is an abstract syntax tree representing the structure of the query with logical operators as nodes and access operators as leaves. The *query optimiser* applies query rewritings that manipulate the logical query to reduce the size of intermediate results, e.g. by “pushing” selection operations as far down the query plan as possible. This helps to make the plan more efficient.

Next, several *physical query plans* are considered for the query by varying which algorithms are used for each operation. For example, there are several algorithms for performing a join between two relations, e.g. nested-loops join, hash join, merge join, etc. Each physical query plan is costed based on statistics about the relations (using metadata stored in the catalog such as the cardinality, value distribution, etc.), the available algorithms, and the ordering of relations in an operation. The plan with the lowest estimated cost is then evaluated to generate the query answer. Note that some operators, e.g. sort or join, are referred to as *blocking* operators as they must process their entire input before generating any output. If such operators occur in a query, then the entire data set must be seen before results are returned to the user. For more details about classical query processing in the relational setting see [GMWU99].

2.2.2 Distributed Query Processing

Classical query processing has proven very successful for centralised stored data. However, with the increase in the number and variety of distributed data sources, there is a need for query processing over data residing at multiple sites. A *distributed query processor* (DQP) operates over a distributed database, i.e. a database where both the data sources and the computational resource required to run a query evaluation plan are partitioned across various sites over a wide area, i.e. a robust network. A survey of the challenges and techniques developed can be found in [Kos00].

A query may make use of data from multiple sites. To support this, the metadata stored in the system catalog has additional information about where data items are placed and what capabilities each site has regarding the evaluation of query plans in order for the optimiser to generate the code that will make best use of resources. One popular approach for query compilation and optimisation in the distributed setting is the *two-phase optimisation paradigm* [Kos00]. This approach involves first generating a single site (classical) query execution plan and then *partitioning* the query plan, based on the content of the metadata in the catalog, into fragments that can be executed at particular sites.

Distributed query processing has been widely studied for relational data sources as surveyed in [Kos00]. Techniques have also been developed for executing XQuery and XPath over distributed sources [SFJ03]. Distributed query processing over triple stores is less mature, although there is currently a lot of interest in this topic [QL08, LBW07].



2.2.3 Stream Query Processing

Classical mechanisms for query processing are not directly applicable to data streams due to crucial differences between streaming and stored data sources. Traditional query processing assumes the data is stored such that knowledge about the data (e.g. its cardinality, distribution, etc.) is available. Data streams, and their usage, reflect a continuously changing world view. As such, a query may need to be re-evaluated every time that data is updated, e.g. in response to or as a reaction to, a change in the world as reflected in the newly-updated data. More importantly, from a query processing point of view, this implies that data streams have indefinite, possibly infinite, cardinality. Additionally, classical declarative query languages such as SQL [SQL92], XQuery [BCF⁺07], and SPARQL [Pe08] are also not suited for use with data streams. The semantics associated with these languages assumes that they are executed once over the entire data set, which is not possible with data streams, and they include the use of operators that process the entire data set to generate answers, e.g. sort, join, and aggregates. These limitations have led to new query languages and query processing techniques being developed.

A *continuous query* is a query that is evaluated over new data items as they arrive [TGNO92]. To enable these queries, several extensions for SQL have been proposed [ACQ⁺03, ABW06, KCC⁺03, BGFP08]. These query languages introduce the notion of a *window*, which defines a finite sub-sequence over a given data stream, to enable operations such as joins and aggregations, which are only well defined over the entire data set to be processed. Typically, the window refers to the most recent tuples either in time or in number of tuples, and can *slide* over the stream for each evaluation point. Similar query language extensions have been defined for XML query languages [FMSS07, MPC07] and for SPARQL [BGJ08, BBC⁺09].

Continuous queries still follow the classical query processing steps, i.e. the declarative query is parsed into a logical query plan which is then instantiated into a physical query plan. However, the evaluation of the physical query plan has required DSMSs to develop new techniques to respond to changing conditions at runtime which depend upon the processing environment and are discussed in the following sections.

Event Stream Processing

The majority of the DSMSs that have been developed are event stream processing systems which perform centralised query evaluation over relational data streams, e.g. Aurora [ACQ⁺03], STREAM [ABB⁺09], and TelegraphCQ [KCC⁺03]. Borealis is an example of distributed system for event streams [AAB⁺05a], i.e. one that processes queries over a robust network. Event stream DSMSs have developed techniques for dealing with high or bursty data rates, e.g. Eddies [AH00] adaptively route tuples during join evaluation, and load shedding policies select which tuples to discard when the data rate is too high [TcZ⁺03].

Event stream DSMS have also considered another common form of query over a data stream, a *history query* which is posed over the tuples that have already appeared on the stream [GNW07]. In order to answer such queries, the DSMS must store the tuples of the stream, for example in a database. A query just involving the history of a stream is referred to as a *one-off* query, and can be compiled and evaluated using classical query processing over a database. However, it is possible to combine a continuous query with a history query thereby requiring the combined use of both types of query evaluation.

Acquisitional Stream Query Processing

Acquisitional stream DSMSs such as Cougar [YG02], TinyDB [MFHH05], and SNEE [GBJ⁺09] distribute the query processing load over different nodes in the sensor network. The DSMS consists



of server software, which runs on the sensor network base station (a PC), and the sensor node software, which runs on the nodes in the sensor network [GM04]. The server software is responsible for parsing the queries, generating a query execution plan, and delivering the plan into the network for execution. The query plans are optimised by the server software, generally with the aim to reduce energy consumption, and control where, when, and how often data is acquired by the sensors. The execution plans are run by the sensor nodes to gather, process, and deliver the data that answers the query back to the base station. The use of in-network query processing has been shown to be more energy efficient than one that continually sends all data back [MFHH05].

Sensor networks can also be used to answer spatial and spatio-temporal queries. Spatial queries are important as they endow applications with the ability to answer questions such as “find the average temperature in an area” or “find the closest exit for which a safe path exists” in a fire evacuation scenario. Viewed as a distributed computing system, sensor networks differ from traditional centralised database systems, in which the field of spatial and spatio-temporal query processing have been extensively studied. Spatio-temporal query processing in sensor networks has been studied more recently in [DF03] and [SKG05].

2.2.4 Data Integration

Data integration, or information integration, is a topic that has been investigated since the early 1990s and is based on the notion of a *mediator* to access multiple sources [Wie92]. Essentially, the idea is that a collection of heterogeneous, distributed, and autonomous data sources are presented as a single *virtual* data source with a *global schema*, also called a mediated schema, against which queries can be posed. The schemas of the data sources, called *local schemas*, are related to the global schema using mappings [Len02], the most common types being: “global-as-view” (GAV) where the global schema is expressed as views over the local schemas; “local-as-view” (LAV) where the local schemas are expressed as views over the global schema; and “global-local-as-view” (GLAV) which combines the previous two approaches. Upon receiving a query over the global schema, the integration system rewrites it into one or more queries over the underlying data sources to extract and join the relevant data to be returned to the consumer.

The majority of the work on data integration has focused on relational data sources. A good overview of the technical challenges and the available solutions is given in Halevy’s survey [Hal01], while his more recent paper gives a survey of implemented integration systems [HRO06].

More recently, there has been growing interest in exploiting the Semantic Web, and the use of ontologies, to integrate data sources. Ontologies are a mechanism for defining and modelling the concepts that exist in the real world by defining, “a *formal specification of a shared conceptualisation*” [SS04], where a conceptualisation is an abstract model of some aspect of the world. Ontologies are useful for explicitly defining the semantics that are associated with the schemas of data sources.

Several surveys, from different research perspectives, on integrating data using ontologies exist [Noy04, DH05, WVV⁺01]. Work in this area has focused on generating mappings between ontologies [ES07], and on exposing existing resources, such as relational databases, using semantic models [BC07, RCGP06, CGL⁺09]. This work is the focus of work package 4 and is covered in deliverable D4.1 [CC09].

The only work on integrating data streams is [CGN05, Gra07]. This work has only considered a limited query language—select-project queries—over event streams. The focus is on reducing the burden on original data sources by introducing data *republishers*, which give rise to a choice of where to collect data for query answering.

2.3 Data Analysis

Data analysis is the process of modelling and analysing data in order to support data mining, knowledge discovery and information retrieval tasks. The adopted techniques are closely related to the fields of machine learning and statistics. Every data analysis system (which normally is a subsystem of a larger infrastructure) has to go through some general steps before performing the core data analysis, which is based on specific requirements. These steps include data collection, data integration, data modelling and data cleaning. Data classification, data clustering, outlier detection and data sampling are some of the techniques that are widely used in these steps and will be briefly presented in the following sections.

2.3.1 Data Modelling

When considering problems that deal with real world data, a system has to map readings or input values onto physical reality. In this sense, a common way to address this problem is to use statistical modelling techniques. Having models that represent the world in the field of study and complement missing or erroneous data, provides data management and analysis systems with the ability to manage and process the collected data in a meaningful and efficient manner.

DBMSs use histograms and other summarising techniques in order to optimise the evaluation of various query types [Ioa03]. When a system deals with streaming data (as is the case in a sensor network), given that in many cases a data item is seen only once, an online modelling technique has to be adopted, in order to provide in each time instance, a view that approximates with high accuracy the data that had already been seen. While traditional database systems focus on providing precise answers to queries evaluated through stable query plans, as stated in [BBD⁺02], many sensor network systems focus on continuous queries using approximation and adaptivity due to the constraints they inherently have (as presented in section 2.1.2). In this sense, popular techniques for approximation that are widely used by systems built upon sensor networks include kernel density estimators [SPP⁺06], histograms [SLMJ07] and wavelets [ZC06].

2.3.2 Data Classification

Data classification has been thoroughly studied in statistics, machine learning, neural networks and expert systems, and is an important theme in data mining [CHYC96]. Data classification is a process which, given a set of objects in a database, finds the common properties and classifies the object into classes, according to a classification model. In order to construct the model, there is a training set whose tuples contain the same set of attributes as the tuples in the database, and have a known class identity associated with them. The classification task aims at analyzing the training data and developing an accurate description or a model for each class using the attributes available in the data. The class descriptions are then used to classify future data in the database or to develop classification rules for each class in the database. There are many applications of data classification such as medical diagnosis, performance prediction and selective marketing [CHYC96].

The best known methods for data classification are k-nearest neighbours (k-NN) [Alp04], decision trees [Qui86, Qui93], Naïve Bayes [Alp04], and Support Vector Machines (SVM) [Alp04]. The k-nearest neighbours algorithm (k-NN), which is one of the simplest of all machine learning algorithms, is a method for classifying objects based on the training examples that are closest to each other in the feature space. k-NN is a type of instance-based learning algorithm, where an object is classified by a majority vote of its neighbours, with the object being assigned to the class that is most common amongst its k-nearest neighbours. Thus, the algorithm is sensitive to the local structure of the data [Alp04].

A decision-tree-based classification method is a supervised learning method that constructs decision trees from a set of examples based on a function of the accuracy and the structure of the tree. In

each step of the method, an attribute is selected so as to reduce the number of unclassified tuples in each of the branches created according to the possible values of the attribute. Eventually, a tree is constructed in which each leaf carries a class name and each node specifies an attribute with a branch corresponding to each possible value of that attribute. Representative examples of decision tree learning systems are ID-3 [Qui86] and C4.5 [Qui93].

Another classification method uses Support Vector Machines (SVM). SVM [Alp04] include a set of related supervised learning methods used for classification and regression. Viewing input data as two sets of vectors in an n -dimensional space, an SVM will construct a separating hyperplane in that space, one which maximises the margin between the two data sets. To calculate the margin, two parallel hyperplanes are constructed, one on each side of the separating hyperplane, and a good separation is achieved by the hyperplane that has the largest distance to the neighbouring data points of both classes.

A Naïve Bayes classifier [Alp04] is used in Bayesian statistics and builds a simple probabilistic classifier using Bayes' theorem, under the simplifying assumption that the presence (or absence) of a particular feature in a class is unrelated to the presence (or absence) of any other feature. Naive Bayes classifiers often work well in complex real-world situations, and require only a small amount of training data to estimate the necessary parameters (means and variances of the variables) for classification.

2.3.3 Data Clustering

Clustering is one of the most important and popular data mining and knowledge discovery techniques. It is related to the identification of interesting patterns hidden in large datasets. Clustering is the process of partitioning the input dataset into groups of objects, called *clusters*, where the intra-group distance between objects is smaller than the inter-group distance, so that objects within a cluster have a strong relationship between them [HV05]. Data clustering is an often used statistical analysis method and is used in research fields like machine learning [Alp04], data mining [HK06], pattern recognition [TK03], etc. Clustering has many useful applications, like data reduction or cluster-based prediction [HV05] in fields like web-mining [JK98] and analysis of spatial, temporal and spatio-temporal data [KMB05].

In this section a short introduction to clustering methods will be presented, as well as a classification of these methods into different categories, based on the way the different approaches form the clusters. Objects, i.e. readings which may be one-dimensional or multi-dimensional points, are assigned to clusters based on the similarity (or distance) between them. It is the responsibility of a domain expert to determine which distance metric will be used, the Euclidean Distance being usually the default one. The chosen distance metric influences the semantic relationship that holds between objects belonging to the same cluster. For example, a cluster that has objects with similar sensor readings may represent one distinct neighbourhood in the area under study.

Figure 2.1 represents a hierarchy of clustering methods that have been proposed in the literature. At the highest level, clustering methods are divided into *hierarchical* and *partitional*. Hierarchical clustering methods produce the set of clusters one step at the time, with the clusters produced in step $i+1$ being based on the clusters produced in step i . Clustering Using Representatives (CURE) [GRS98] and Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [ZRL96] are the best known hierarchical algorithms.

Partitional clustering methods start with a set of clusters with the desired size. Then, in each step, the set of clusters is improved, until a specified threshold of quality is reached. Partitional methods are suitable for applications with extremely large input datasets. The most popular and widely used clustering algorithm, k-means [Mac67], belongs to the category of partitional clustering techniques. Other popular clustering algorithms that belong to the category of partitional algorithms are Partitioning Around Medoids (PAM) [KR90], Clustering Large Applications (CLARA) [KR90], and Clustering Large Applications based upon RANdomized Search (CLARANS) [NH94].

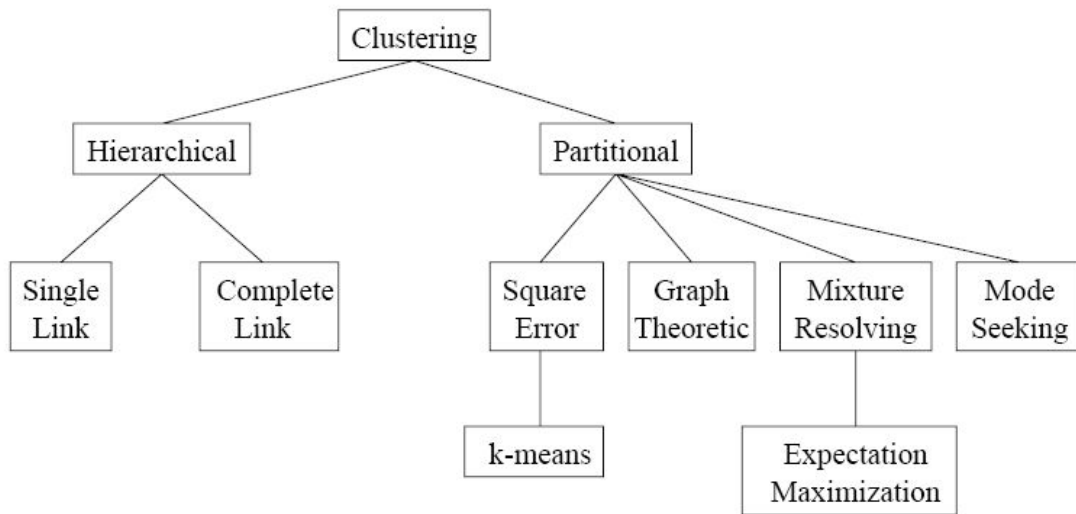


Figure 2.1: Classification of clustering methods.

Another important technique which identifies clusters included in subspaces of high multidimensional data is CLustering In QUest (CLIQUE) [AGGR99]. CLIQUE is scalable, does not make assumptions about the normal distribution of the data and is not sensitive to the insertion of records. Clustering algorithms based on the density of clusters have also been proposed, e.g. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [EKX95] and DENsity-based CLUstEring (DENCLUE) [HK98].

Data clustering in sensor network environments has to take into account the constraints that arise in these setting. More specifically, it must contend with such limitations, without impacting negatively on the operation of the network. Several techniques have been proposed in the literature, which try to address these constraints in the context of clustering tasks. In [ZCY⁺07], the proposed algorithms learn the distribution of the data, maximising the probability of clusters (expectation maximisation). In addition to that, in order to reduce the average processing cost, a strategy called “test-and-cluster” is used which is very efficient in clustering large data streams. In [ZLW07], an efficient incremental algorithm for the clustering of subgroups of multiple streams with the use of sliding windows is proposed. The algorithm captures the way a pattern of a subset of values of a subgroup changes in time, while preserving the future potential patterns which may appear. In [CMZ07] many algorithms that focus on the clustering of k centers are proposed. The algorithms vary depending on the centralised algorithm and on the clusters they maintain (either a single global clustering or many local clusters that can be merged). Moreover, the required communication for the collection of the data in a single central site is very low in comparison with a centralised algorithm. Another accurate, energy efficient and robust (regarding changes in the topology of the network) approach is proposed in [YG08]. A hierarchical organisation of sensor networks is used so as to reduce the energy consumption and achieve scalability. The technique initially compresses the time series produced by the sensor nodes in a compact representation and then based on dynamic time warping distances, hierarchically groups the approximated time series to a global clustering model in an incremental way.

2.3.4 Outlier Detection

One of the main knowledge discovery tasks that arises in many data management and analysis systems is *outlier detection*, which reports the discovery of unusual events and detects anomalies in the data. This task focuses on a very small percentage of data objects, which can often be discarded as noise. In many applications however, rare events are often more interesting than common ones. Such applications include the monitoring of environmental phenomena, of networks

or of health indicators, the detection of credit card fraud, the approval of loans, and the monitoring of criminal activities in electronic commerce [KN98].

According to Hawkins' definition [Haw80], an "outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism". Thus, outlier detection is also called deviation detection. Several outlier detection techniques have been proposed in the literature and fall into two broad categories: *centralised* and *distributed*. By centralised it is meant that there is a large dataset in a single site and the outlier detection algorithm is applied to the whole dataset in this site. Centralised approaches for outlier detection can be applied in sensor networks, where data from the sensor nodes is either collected at a sink before the outlier detection algorithm is applied, or the data remain in each sensor node and the outlier detection algorithm is called locally. In the second category, the outlier detection algorithms are executed in a distributed manner. The outlier detection techniques proposed in the literature so far can be classified into four types: a) distance-based, b) density-based, c) clustering-based and d) statistic-based.

In [SPP⁺06], a distance-based outlier is defined as follows: "A point P in a dataset T is a (D, r) -outlier if at most D of the points in T lie within distance r from P ", which means that an outlier is an observation that is *sufficiently* far from most other observations in the dataset. Centralised methods belonging to the distance-based category are proposed and more thoroughly described in [KN98] and [RRS00].

The definition of distance-based outliers focuses on identifying outlying values with respect to a global view of the dataset. However, in datasets with a more complex structure, there may be values that are outliers relative to their local neighbourhoods. These outliers can be identified by computing the densities of the neighbourhoods. Thus, a point is reported as an outlier if it has much lower density than that of its neighbours and is regarded as a local, or density-based, outlier. Some of the most interesting centralised algorithms for detecting density-based outliers can be found in [RWP04, PKGF03, BKNS00].

Regarding the clustering-based approach to outlier detection [JMF99], there are several known clustering algorithms that consider outliers, such as CLARANS [NH94], DBSCAN [EKX95], CURE [GRS98], and BIRCH [ZRL96]. However, the definition of outliers that they use is subjective to their specific needs and is related to the process of detecting clusters, in order to ensure that the outliers do not interfere with that process.

As mentioned above there is a statistics-based definition of outliers [BL94]. This approach uses a distribution or probability model that corresponds to the given dataset and then identifies outliers with respect to the model using a discordancy test. A discordancy test is a process that checks whether there is statistically significant evidence that there are values in the dataset that do not conform to the assumed distribution model. To perform this test the knowledge of the assumed data distribution, as well as the knowledge of the distribution parameters (e.g. the mean and the variance), is required [HK06].

The problem of detecting outliers in sensor networks is especially important, because it can be used to identify faulty sensors and to filter spurious reports from different sensors. Even if we are certain of the quality of measurements reported by the sensors, the identification of outliers provides an efficient way to focus on the interesting events in the sensor network. To further emphasise the importance of outlier detection in sensor networks, consider the following example. Assume a machine that is fitted with sensors that monitor its operation. These sensors measure quantities such as temperature, pressure, and vibration amplitude for the different parts of the machine. If there is some malfunction or any other abnormality, some of these readings will deviate significantly from the norm. For example, it may be the case that a small part of the engine is overheated when compared to the rest of the engine, or that the entire engine is overheated when compared to the rest of the machine. Therefore, the ability to identify outliers as early as possible can allow engineers to correct them during operation and prevent the engine from further damage. Another motivating example for outlier detection in sensor networks is the following. Assume a forest



with sensor nodes that measure temperature. If there are some sensor nodes located in the same neighbourhood that report outliers regarding the temperature, then the monitoring system can infer that the possibility of fire in this neighbourhood is higher than the usual one.

Many techniques for outlier detection in sensor networks have been proposed [SPP⁺06, SLMJ07, BSG⁺06, ZC06, ZWL07]. While some (e.g. [AJA04]) are centralised, most of the approaches work in a distributed manner. For example, if we want to detect outliers in a sensor network, one approach would be for the sensor nodes to process their data and that of their children and then send the results towards their parent nodes. The results to the upper levels of the hierarchy that is built within the sensor network. This requires communication between the nodes of the network in order to detect the outliers from the values produced. In [SLMJ07], the problem of outlier detection in sensor networks is addressed by creating a model that characterises the normal behavior and outliers are detected as the deviations. However, there is a dependence on highly capable sensors to manage groups of other sensors and to perform outlier detection. In [BSG⁺06], a non-parametric, unsupervised algorithm is presented that has the following properties: a) it works in-network with communication load proportional to the outcome, b) it works for many outlier detection heuristics, because it is generic, c) it is robust regarding the data and network changes, and d) it reveals the results to all the sensors of the network.

2.3.5 Sampling

Many large scale applications produce high volumes of streaming data arriving at very fast rates, which can also be stored as large datasets in traditional databases. Such applications include network monitoring, financial data analysis and some sensor network applications. Moreover, for streaming data, one has to take into account that a data item may be seen only once. Due to this fact, there is a need for finding sampling methods to reduce the amount of data which has to be processed by the data analysis algorithms. Consequently, the results of the algorithms performed on these data streams will be rough approximations of the real values.

Sampling large databases has been a well studied problem for many years in database research [OR95]. The functional requirements of some applications (e.g. data mining, or decision support applications) often do not depend on finding exact answers to queries; an estimate is sufficient. Consequently, these applications can make use of approximate queries. Database systems that support this type of query usually build a synopsis of the database tables in an offline (pre-processing) fashion, making extensive use of histograms [PGI99], wavelet-based summaries [CGRS04], and various sampling techniques [GM98]. Later on, after a rewriting step, query evaluation is performed over the synopsis rather than original database relations. The idea behind sampling is that a small sample of the database often represents the entire data with a satisfactory accuracy. A fast approximate query is executed over the sample and the result is then scaled to the size of the database.

When sampling data streams, some additional constraints arise. As mentioned above, one such constraint is that usually streaming data may only be seen once. This makes sampling of distinct items extremely difficult. Additionally, the size of a stream is generally unknown, which complicates the extraction of fixed-sized samples. During the past few years many sophisticated sampling methods have been proposed [WMN04, JMR05]. Each of them has a different purpose, thereby covering most of the needs for different aggregates (e.g. quantiles [MRL99, BW01, GK01], distinct counts [CL03, CLKB04], adaptive geometric sampling [HS08] and reservoir sampling [DLK08]).

Sensor networks provide an environment where, potentially, large quantities of streaming data are generated. This, combined with the fact that one of the major constraints in sensor networks is limited energy, creates the need to reduce the energy consumed by radio transmissions from sensor nodes (given that radio transmissions dominate energy consumption). Thus, most applications in data mining take into consideration sophisticated sampling techniques for preserving data characteristics that would be otherwise unobtainable through traditional exact solutions [LAGD08, JC04, MS03]. In [DLK08] some indices and algorithms are proposed for evaluating



historical queries in flash-equipped sensor nodes in an efficient way. In that approach the need for sophisticated methods for random sampling is emphasised, because in many applications the amount of data being generated is more than the amount of data that can be stored. The proposed methods maintain a random sample in order to answer approximate queries and perform data analysis, employing the Reservoir Sampling technique [DLK08], which will be further analyzed in Section 4.1.2.

2.4 Summary

This chapter has given a brief overview of data management, query processing, and data analysis, highlighting some of the major issues in these areas. It has focused on two types of data source, stored relational databases and relational data streams with a particular focus on streams generated by sensor networks and the constraints on energy, processing, and memory that this entails. Techniques for managing and analyzing data in both centralised and distributed settings have been presented.

3. Requirements

This chapter describes the functionality required from WP2 in the context of the SemSorGrid4Env project. Firstly, the requirements from the application use-cases are described (Section 3.1). Then, the requirements from the technical work packages are described (Section 3.2). The requirements in this chapter, together with the currently existing technology/concepts described in the next chapter, are used to identify the future work for WP2 in Chapter 6.

3.1 Requirements from the Use Cases

3.1.1 Fire Risk Monitoring and Warning

This section describes the requirements of WP2 which arise in the fire-use case scenario, as based on the WP6 requirements specification [LDI09], and following discussions with the relevant stakeholders. These are summarised in Table 3.1.

- *Types of data sources.* Data will be provided from both stored sources (e.g. historical data) and streaming sources (e.g. the SN deployed at the site of interest).
- *Location of query evaluation.* Since it will be possible to change the functionality of the sensor network once it has been deployed, the ability to execute queries in the sensor network as well as the centrally at a base station will preserve scarce energy resources.
- *Stored data request mode.* Historical stored data may be requested by posing a query.
- *Stored data delivery mode.* Historical stored data may be delivered either directly by the data-source, or by means of an intermediary. The latter may be useful in the case of resource-intensive queries.
- *Streaming data request mode.* Streaming data may be requested by posing a query, or by subscription.
- *Streaming data access mode.* Streaming data may be accessed either in push mode (i.e. the data is sent to the data consumer without an explicit request) or pull mode (i.e. data is only sent to the consumer when explicitly requested).
- *Streaming data delivery mode.* Streaming data may be delivered either directly (i.e. by the source itself) or indirectly (i.e. by means of an intermediary – useful in the case of resource-intensive queries).
- *Extents handled by query language/optimiser.* Since there will be both streaming and stored data sources, the query language needs to handle both stream and relation extents.
- *Multiple query execution.* This is required as more than one query may need to be posed simultaneously.
- *QoS-awareness.* This is required in order to obtain a short delivery time for query results when there is a high fire risk, and to preserve energy when the risk of a fire occurring is lower.
- *Simple temporal reasoning in query language.* This is required in order to correlate data collected at different times.
- *Simple spatial reasoning in query language.* This is required in order to correlate data collected at different locations.

Dimension	WP6	WP7
<i>Stored data</i>		
Request mode	$\in \{\text{Query, Attribute-value pairs}\}$	{Query}
Delivery mode	$\in \{\text{Direct, Indirect}\}$	{Direct, Indirect}
<i>Streaming data</i>		
Request mode	$\in \{\text{Query, Event Subscription}\}$	{Query, Event Subscription}
Access mode	$\in \{\text{Pull, Push}\}$	{Pull, Push}
Delivery mode	$\in \{\text{Direct, Indirect}\}$	{Direct, Indirect}
<i>Query Processing</i>		
Location of query evaluation	$\in \{\text{Sensor Network, Robust Network}\}$	{Robust Network}
Extents	$\in \{\text{Relations, Streams}\}$	{Relations, Streams}
Multiple-query execution	Yes	Yes
QoS-awareness	Yes	No
Simple temporal reasoning	Yes	Yes
Simple spatial reasoning	Yes	Yes
<i>Data Analysis</i>		
Data Modelling	No	No
Data Classification	No	No
Data Clustering	No	Yes
Outlier detection	Yes	No
Sampling	Yes	Yes

Table 3.1: Data Management Dimensions of SemSorGrid4Env Applications.

- *Outlier detection.* This is useful in order to either identify data for a faulty sensor (i.e. data cleaning), or to identify an event of interest (e.g. a forest fire).
- *Sampling.* This is required to perform outlier detection in an efficient way. Other than that, it is useful since streaming data in in-network processing can be seen only once and cannot be stored locally.

3.1.2 Coastal and Estuarine Flood Warning in Southern UK

According to the WP7 requirements specification [CHSR09], and discussions with the relevant stakeholders, the following requirements have been identified as summarised in Table 3.1. The list below explains each requirement in more detail:

- *Types of data sources.* Data will be provided from both stored sources (e.g. historical data) and streaming sources (e.g. the SN deployed at the site of interest).
- *Location of query evaluation.* The sensor network to be deployed will have a fixed functionality and will not support in-network processing. Therefore, query evaluation will only take place in a robust network.
- *Stored data request mode.* Historical stored data may be requested by posing a query.
- *Stored data delivery mode.* Historical stored data may be delivered either directly by the data-source, or by means of an intermediary. The latter may be useful in the case of resource-intensive queries.
- *Streaming data request mode.* Streaming data may be requested by posing a query, or by subscription.

- *Streaming data access mode.* Streaming data may be accessed either in push mode (i.e. the data is sent to the data consumer without an explicit request) or pull mode (i.e. data is only sent to the consumer when explicitly requested).
- *Streaming data delivery mode.* Streaming data may be delivered either directly (i.e. by the data source itself) or indirectly (i.e. by means of an intermediary – useful in the case of resource-intensive queries).
- *Extents handled by query language/optimiser.* Since there will be both streaming and stored data sources, the query language needs to handle both stream and relation extents.
- *Multiple query execution.* This is required as more than one query may need to be posed simultaneously.
- *QoS-awareness.* Since queries will not be executed on the sensor network, there is limited scope for this.
- *Simple temporal reasoning in query language.* This is required in order to correlate data collected at different times.
- *Simple spatial reasoning in query language.* This is required in order to correlate data collected at different locations.
- *Data Clustering.* Clustering time-stamped sequences of values is useful, considering the benefits from the identification of spatial locations that show similar behavior, or identification of different time intervals over time that also exhibit similarities (for example identification periodicities).
- *Sampling.* This is useful since the data analysis computations on large data sets, e.g. consisting of historical archives of sensor readings, will be performed more efficiently.

3.2 Requirements of the Technical Work Packages

3.2.1 Architecture and Infrastructure

The architecture for the SemSorGrid4Env project is defined in Deliverable D1.3v1 [GGF⁺09], and is discussed in Chapter 5 of this document. The SemSorGrid4Env architecture places certain requirements on the publishing of data as Web services. It is expected that the data management services developed in WP 2 will conform to the Web service interfaces defined in D1.3v1, both in terms of the operations that they make available and the metadata that they publish. That is, the data streams generated (either acquisitional or event) will expose the interfaces of the Streaming Data Service, and stored extents will expose the interfaces of the Stored Data Service.

3.2.2 Registry

The proposed registry for the SemSorGrid4Env project is described in Deliverable D3.1v1 [KKK09], and is part of WP3. At initialisation, the registry requires data sources to register with it, and for metadata to be provided. Examples of metadata include the schema of the data source, and for example, if the source is a sensor network, the region being monitored, types of sensor readings that may be made, and whether it is able to evaluate queries within the network, or it behaves in a fixed, predetermined manner. The mechanism whereby a data source can register and provide metadata about itself is described in Section 5.3. The registry then enables sources to be discovered on-the-fly by applications, as envisioned by the SemSorGrid4Env project.



3.2.3 Semantic Data Integration

The semantic integrator developed as part of WP4 will enable independent and heterogeneous data sources, both streaming and stored, to be combined in an arbitrary fashion on-the-fly. In order for this to happen, data sources should be discoverable via the registry, and the interfaces defined by the SemSorGrid4Env architecture should be exposed by data sources, so that there is a common access mechanism for data retrieval. The capability to process queries to support a Global-as-View or a Local-as-View approach to data integration is also required.

3.2.4 High-level Application Programming Interfaces

The high-level application programming interfaces (APIs) to be designed and developed by WP5, will enable semantic mashups to be developed. Initial thoughts on these are sketched in Deliverable D5.1 [PRMS09]. In that document, the use of the more recent, resource-oriented approach based on representational state transfer (REST) for rapid development of lightweight applications is proposed, in conjunction with the service-oriented approach that emerged from the SemSorGrid4Env architecture. It may therefore be necessary for data sources to provide a REST API in addition to the service-oriented interface described in the SemSorGrid4Env architecture.



4. Existing Work

This chapter describes existing work, comprising both theoretical techniques and software, which WP2 is going to use as a basis for SemSorGrid4Env project. In Section 4.1, we describe data analysis algorithms, previously developed at NKUA, for outlier detection, which will be implemented as part of future work and incorporated into query processing techniques. OGSA-DAI [AAB⁺05b] is the off-the-shelf software chosen for SemSorGrid4Env to provide access to stored data sources as it is a mature existing implementation and provides a service-based interface compliant with the WS-DAI standard [AAK⁺06, ACK⁺06], adopted by the SemSorGrid4Env architecture (see Chapter 5). Finally, this chapter describes SNEE, the sensor network query optimiser, developed at UNIMAN, that evaluates queries expressed in SNEEq, a rich, expressive continuous query language, within the sensor network.

4.1 In-network and Centralised Data Analysis

In Section 2.3, some of the most important data analysis techniques were presented, particularly with respect to their applications in sensor network environments. In this section, existing work regarding outlier detection and data sampling is described. This work will be used and integrated with UNIMAN's SNEE.

4.1.1 Motivation for Outlier Detection and Data Sampling

Motivations for data sampling and mining outliers in wireless sensor networks include:

1. Wireless sensor networks can generate large amounts of streaming data that arrive at high rates and hence may only be seen once;
2. The sensing devices are battery powered and, consequently, their performance deteriorates as their power is exhausted;
3. Data collection is done from the real world, using devices which are imperfect;
4. The number of sensors in real deployments of sensor networks may be large, thus increasing the probability of errors, since this probability increases with network size.

The first two properties indicate the importance of data sampling in sensor networks, while the last two suggest that there is a need to detect outliers in sensor networks. As mentioned in Section 2.3.4, there are two general approaches regarding mining outliers in sensor networks: a centralised and an in-network one. In the centralised approach, either all data is collected to a central sink that is responsible for performing all computations, or else the data remains in each sensor node and the outlier detection algorithm is called locally. In in-network processing, the outlier detection algorithms are executed in a distributed manner. This distinction holds for most data analysis techniques, ranging from data clustering to query processing and outlier detection.

4.1.2 In-network Outlier Detection

A centralised approach for outlier mining in wireless sensor networks is highly energy consuming, since the amount of data can be large. Obviously, this will lead to the quick depletion of the sensor batteries, followed by network malfunction. Consequently, more efficient methods require as little transmission as possible, while still achieving the goals of outlier detection over as large an amount of data as possible that is collected in a distributed fashion by a wireless sensor network, which in many cases may be composed of tens or thousands of sensors.

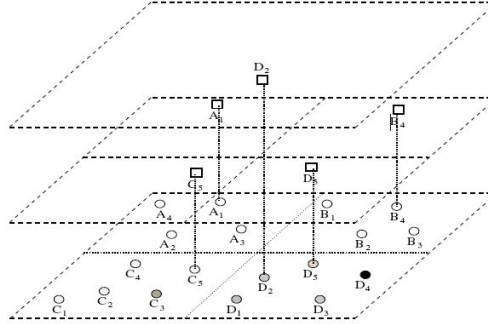


Figure 4.1: Hierarchical Organisation of the Sensor Network

Mining Outliers using Non-parametric Models

Stemming from the reasons mentioned in Section 4.1.1, there are two different in-network outlier detection algorithms which we consider suitable for the needs of WP2 in SemSorGrid4Env. The algorithms chosen to be implemented and integrated with UNIMAN’s SNEE are proposed in [SPP⁺06] and will be presented and analyzed in this section. The integration and implementation plan will be presented in Chapter 6.

In [SPP⁺06] there are two proposed methods for online distributed outlier detection in wireless sensor networks. The first one uses a fast distance-based algorithm called *Distributed Deviation Detection - D3*, while the second one employs a more robust, local metrics-based approach, called *Multi-Granular Deviation Detection - MGDD*.

In order to achieve scalability, a hierarchical organisation of the sensor nodes is supposed in [SPP⁺06], using overlapping virtual grids, each of which corresponds to a different level of granularity. The lowest tier represents the finest level of granularity and is partitioned in cells identifying small local areas. The highest tier represents the entire area covered by the sensor network. For each cell in the lowest tier there is a leadership role that rotates among the sensor nodes of the cell in an energy-efficient manner. The leader node of a cell collects values from the sensor nodes of its cell and processes them. Moving up the hierarchy the leaders collect values from the leaders of the sub-cells (see Figure 4.1).

The hierarchical organisation presented above is different from the sensor network organisation that is assumed by SNEE, which will be presented in Section 4.3. Also, we have to mention that the hierarchical topology may not be realisable easily in TinyOS.

The proposed approach uses kernel density estimators [Sco92] to approximate the sensor data distribution. A kernel estimator is a generalised form of sampling. In the basic step, a uniform random sample is produced that distributes the weight of a point in the space around it. By summing up all the kernel functions, a density function for the dataset can be obtained.

More formally, assume that we have a static relation, T , that stores the d -dimensional values $t = (t_1, \dots, t_d)$, whose distribution we want to approximate. The recorded values must fall in the interval $[0, 1]^d$. Let R be a random sample of T , and $k(x)$ a d -dimensional function of $x = (x_1, \dots, x_d)$, such that $\int_{[0,1]^d} k(x) dx = 1$, for all tuples in R . We call $k(x)$ the *kernel function*, which is decided to be the Epanechnikov kernel. The underlying distribution $f(x)$, according to which the values in T were generated can be approximated using the following function:

$$f(x) = \frac{1}{|T|} \sum_{t_i \in R} k(x_1 - t_{i1}, \dots, x_d - t_{id}) \quad (4.1)$$

In the sensor network setting, we require that each sensor maintains a model for the distribution of values it generates. Since we are not interested in the entire history of the values produced by the sensors, it suffices to consider the values in a sliding window W of size N . Then, T holds only the values in this sliding window, i.e. the N most recent values. At each point in time, we are interested in approximating the distribution of the data values within the sliding window. The intuition is to use the kernel estimators for computing an approximation of the new data distribution at each point in time. The first step for creating a kernel estimator for a sliding window W is to maintain online a random sample of the set T that contains the values in the most recent window W . The other quantity we need for the kernel estimator is the standard deviation of the values in the sliding window W . Both of these operations can be efficiently supported in a data streaming environment.

Sampling can be done in various ways, but the most useful ones for the setting proposed in [SPP⁺06] are “Chain Sampling” [BDM02] and “Reservoir Sampling” [DLK08].

The “Chain Sampling” algorithm starts with an initial random sample and proceeds as follows. For each point in the sample, it picks at random the next element from the data stream that will replace it. The only restriction is that the new value must replace the old one, before the old one expires from the sliding window (that is, the two should not be more than N values apart in the stream). The memory requirements are $O(d|R|)$.

In the “Reservoir Sampling” algorithm the intention is to maintain a sample of size k . In order to achieve this, the first k arriving data records are inserted into the reservoir and when the n th ($n > k$) data record arrives, it will be included in the reservoir with probability $\frac{k}{n}$, by replacing a random victim in the reservoir; with probability $1 - \frac{k}{n}$, the arriving data record is not included in the reservoir.

The estimate for the standard deviation of the sliding window is computed using a concise histogram along the time axis. The estimate of the standard deviation is derived by combining the statistical information stored in all the buckets of the histogram. The required memory is $O(\frac{d}{\varepsilon^2} \log|W|)$ where ε is the maximum relative error we wish to tolerate in the estimation, and $|W|$ is the size of the sliding window.

Each sensor maintains a density distribution function $f(x)$. For each new observation P , the sensor uses this function to estimate the number of the values that can be considered neighbours of P when using a maximum neighbourhood radius of r . More specifically, a number $N(P, r)$ is estimated. $N(P, r)$ corresponds to the number of values that fall in the interval $[P - r, P + r]$. If $N(P, r)$ is below an application-specific threshold t , P is marked as a distance based outlier.

The process of identifying outliers in a *parent* node is different from the one presented above. Each parent node collects in a pool all the values measured by its children. This means that, in a parent process, outliers are identified in the light of this new dataset. In order to find the outliers in its level of granularity, a parent has to process only the outliers reported by its children. Any value that is not reported by a node in the lower tier (meaning that this value was not an outlier at a finer granularity) cannot possibly be an outlier at the parent’s level. Consequently, all the other data values can be ignored by the parent, a fact justified by the following theorem [SPP⁺06]:

Theorem 4.1.1 *The nodes n_1, \dots, n_l are the children of the parent node n_p and produce the data streams S_1, \dots, S_l for the corresponding sliding windows W_1, \dots, W_l . The sliding window of the parent node is defined as $W_p = \cup_{i=1}^l W_i$. Let at some point in time, O_1, \dots, O_l be the sets of distance-based outliers corresponding to the l sliding windows. Then, for the set O_p of outliers in W_p (representing the outliers in the tier of the parent node) it holds that $O_p \subseteq \cup_{i=1}^l O_i$.*

The outliers are a subset of the outliers reported by the children nodes. Moreover, the theorem reduces the amount of information that needs to be transferred over the sensor network, which takes into account the *communication cost* constraint stated in Section 2.1.2.

Let W^w and W^b be the sliding windows of the leaf and parent nodes,
 Let R^w and R^b be the samples on W^w and W^b
 Let σ^w and σ^b be the standard deviations on W^w and W^b
 Let f be the fraction of the sample propagated from a child to its parent

D3()

- 1 Assign one leaf node to each one of the input streams;
- 2 Configure all parent nodes in a hierarchy on top of the leaf nodes;
- 3 Initiate `ParentProcess` for each parent node;
- 4 Initiate `LeafProcess` for each leaf node;
- 5 **return** ;

Figure 4.2: Distributed Deviation Detection (D3) algorithm

LEAF PROCESS()

- 1 **if** a new value $S(i)$ arrives from the stream
- 2 **then** Update R^w, σ^w ;
- 3 **if** ($S(i)$ included in R^w)
- 4 **then** Send $S(i)$ to parent with probability f ;
- 5 `IsOutlier`($R^w, \sigma^w, S(i)$);
- 6 **if** ($S(i)$ is an outlier)
- 7 **then** Report $S(i)$ as an outlier; Send $S(i)$ to parent;
- 8 **return** ;

Figure 4.3: Leaf Process algorithm

PARENT PROCESS()

- 1 **if** a new message from a child node arrives
- 2 **then if** (message is new outlier P)
- 3 **then** `IsOutlier`(R^b, σ^b, P);
- 4 **if** (P is an outlier)
- 5 **then** Report P as an outlier; Send P to parent;
- 6 **if** (message is a new value from child l)
- 7 **then** Update σ^b and R^b
- 8 **if** (the new value is included in R^b)
- 9 **then** Send new value to parent with probability f ;
- 10 **return** ;

Figure 4.4: Parent Process algorithm

The Distributed Deviation Detection algorithm is presented in Figure 4.2. The algorithm's first step is to initialise the *LeafProcess* procedure in each node belonging to the lowest tier, the *leaf nodes*. Leaf nodes maintain a kernel density estimation model for approximating the input data distribution and based on this model they report distance-based outliers. The values marked as outliers are transmitted to the parent node and are checked with the model maintained there to determine whether they can be marked as outliers in the above level. This procedure continues recursively until the top level is reached. Each sensor node has the possibility to send to the parent node its current sample as well.

Due to the fact that the data points may exhibit different densities in different regions of the data space or across time, it is not always appropriate to use a single threshold value r to determine the outliers. This fact is addressed by more robust statistical techniques, such as *MDEF*, which detect

ISOOUTLIER(SAMPLE R , STDDEV σ , POINT P)

- 1 Use R and σ to estimate $N(P, r)$;
- 2 **if** ($N(P, r) < t$)
- 3 **then** Mark P as an outlier;
- 4 **return** ;

Figure 4.5: Outlier algorithm

the outliers by using multi-granular local metrics that compare the relative differences between the observed points.

In the second approach presented below, named MGDD, only leaf nodes report outliers with respect to the rest of the observations of their data streams and the observations of the entire region where they belong. Thus, the leaf sensors have a local estimation model of their input data stream, a copy of the *global* probability density function and detect MDEF-based outliers with respect to the *global* estimation model. Note that an outlier in a parent node does not have to be an outlier in any of its children nodes.

MDEF was introduced in [PKG03] and stands for Multi-granularity DEviation Factor. It can cope with variations in the feature space and can detect both isolated outliers and outlying clusters. In order to detect a point as an outlier for both [PKG03] and [SPP⁺06], several definitions have to be given, as follows.

At first, Papadimitriou *et al.* define the r -neighbourhood of an object P to be the set of objects within distance r of P , which is notated as $n(P, r)$ and also referred to as the *sampling neighbourhood*. Also, it is obvious that $n(P, \alpha r)$ is the number of objects in the αr -neighbourhood of P , which is also referred to as the *counting neighbourhood*, where $0 < \alpha < 1$. Additionally, they define $\hat{n}(P_i, r, \alpha)$ to be the average of $n(P, \alpha r)$ of all the objects that belong to the r -neighbourhood of P (Figure 4.6). The *local correlation integral* is then defined as the function $\hat{n}(P_i, \alpha, r)$ over all r . Based on the above definitions, for any P , r , and α , the MDEF at radius r is given from the following equation: $MDEF(P_i, r, \alpha) = 1 - \frac{n(P_i, \alpha r)}{\hat{n}(P_i, \alpha, r)}$. Intuitively, the MDEF is the relative deviation of the local neighbourhood density of a point from the average local neighbourhood density of the points that belong to the r -neighbourhood of P , which essentially introduces the concept of locality in outlier detection. As a result, an object which is actually an outlier will have MDEF far from 0. In [PKG03] and [SPP⁺06], in order to detect a point as an outlier, apart from MDEF, $\sigma_{MDEF} = \frac{\sigma_{\hat{n}(P_i, r, \alpha)}}{\hat{n}(P_i, r, \alpha)}$ has to be computed, which is the normalised standard deviation $\sigma_{\hat{n}}(P_i, r, \alpha)$ of $n(P, \alpha r)$ for $P \in n(P_i, r)$. Having defined MDEF and σ_{MDEF} , Papadimitriou *et al.* in [PKG03] detect a point as an outlier, according to the following definition: “A point is flagged as an outlier, if for any $r \in [r_{min}, r_{max}]$ its MDEF is sufficiently large, i.e., $MDEF(P_i, r, \alpha) > k_{\sigma} * \sigma_{MDEF}(P_i, r, \alpha)$ ”¹. The LOCI algorithm [PKG03], which stands for Local Correlation Integral, computes exact MDEF and σ_{MDEF} values for all objects of a dataset, and reports outliers based on the above definition.

In MGDD, apart from LOCI, *aLOCI* is also used, which is introduced in [PKG03] and linearly approximates the LOCI algorithm. *aLOCI* makes the following assumptions: (a) objects belong to a k -dimensional vector space, i.e. $P_i = (P_i^1, P_i^2, \dots, P_i^k)$, and (b) uses the L_{∞} norm, which is defined as $\|P_i - P_j\| = \max_{1 \leq m \leq k} |P_i^m - P_j^m|$.

In order to quickly estimate the $\hat{n}(P_i, r, \alpha)$ in *aLOCI*, the authors propose the creation of a grid of cells with side $2\alpha r$ over the set of points, and count for each cell the total number of neighbours over all objects in that cell. This number is defined as c_j^2 for cell C_j which has c_j objects in it. By denoting as $C(P_i, r, \alpha)$ the set of cells that lie within r distance from a point P , this can be used as an approximation for the r -neighbourhood of P_i . Furthermore, $S_q(P_i, r, \alpha)$

¹ k_{σ} is usually set to 3.

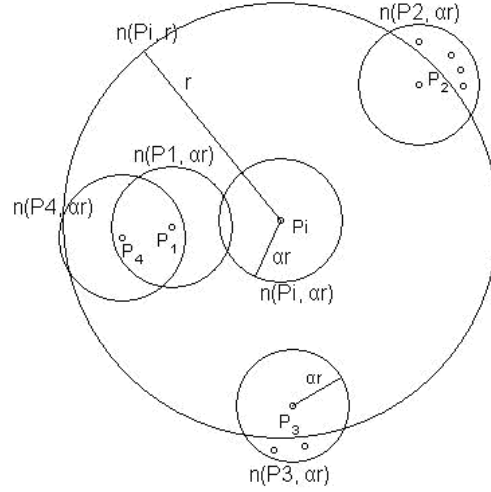


Figure 4.6: Example of the sampling and the counting neighbourhood.

is defined as $S_q(P_i, r, \alpha) \equiv \sum_{C_j \in C(P_i, r, \alpha)} c_j^q$. From this definition, it can be easily seen that $S_2(P_i, r, \alpha)$ is the total number of neighbours, and $S_1(P_i, r, \alpha)$ is the total number of objects in the r -neighbourhood of P_i . Thus, the approximate average neighbour count is given by the equation $\hat{n}(P_i, r, \alpha) = \frac{S_2(P_i, r, \alpha)}{S_1(P_i, r, \alpha)}$ and the standard deviation of the neighbour count is approximately given by $\sigma_{\hat{n}}(P_i, r, \alpha) = \sqrt{\frac{S_3(P_i, r, \alpha)}{S_1(P_i, r, \alpha)} - \frac{S_2^2(P_i, r, \alpha)}{S_1^2(P_i, r, \alpha)}}$. Note that the value of α that is typically used is $\frac{1}{16}$.

As mentioned above, each leaf sensor has a local estimator model and a copy of the global probability density function that is computed by a leader node at the highest level of the hierarchy formed by the sensor network. According to [SPP⁺06], every new observation that is added to the local estimator model at the lowest level sensors is transmitted to the leaders of the sensors with a specified probability f . Iteratively, the leaders transmit the received values with the same probability. If a new observation is finally added to the kernel sample maintained at the leader node of the highest level, the update is transmitted down to the leaf sensor nodes through the intermediate leaders, only if the current estimator model of these leader nodes is significantly different from the estimator model that was last sent to its children.

The function *isMDEFOutlier()*, which is used in the procedure *IsOutlier()* in the MGDD algorithm presented in Figure 4.8, is the aLOCI algorithm [PKGf03]. The two values that have to be computed based on the kernel estimation model, so as to decide if an observation is an outlier, are the following: the counting neighbour of the observation and the number of observations in each of the $2\alpha r$ intervals of the domain (Figure 4.7). In MGDD, it is assumed that the observations are one-dimensional, though the algorithm can be used for observations of greater dimensions as well. After these two computations have been performed, the *MDEF* and σ_{MDEF} can also be computed so as to determine if the examined observation is an actual outlier, according to the definition provided above.

Other Existing Work

In order to build a generic query optimiser that has the ability to include outlier detection facilities either as a Quality of Service (QoS) or as an operator of the query language, some extensions to the methods presented above are possible and are under study. Some examples include the use of multiple data models, instead of using only kernel density estimators to approximate the underlying distribution of data, such as histograms, wavelets or kalman-filters.

²In approximation equations, α equals to 2^{-l} for some positive integer l .

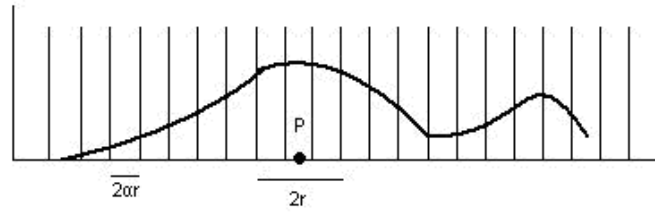


Figure 4.7: For observation P , the counting neighbourhood is estimated by querying with $N(p, \alpha r)$. The domain of the $1 - d$ observations is divided in intervals of width $2\alpha r$, and the number of points in the i th interval is estimated with a range query $N(\alpha r(2i - 1), \alpha r)$.

Let R^g and σ^g be the sample and standard deviation at the leader of the highest level;

MGDD()

- 1 Assign one leaf node to each one of the input streams;
- 2 Configure all parent nodes in a hierarchy on top of the white nodes;
 - ▷ black nodes represent the nodes which simply receive values from their children
- 3 Initiate ParentProcess() for each black node;
 - ▷ white nodes represent the nodes which acquire values from their input stream
- 4 Initiate LeafProcess() for each white node;
- 5 **return** ;

Figure 4.8: Multi Granular Deviation Detection

LEAF PROCESS()

- 1 **if** a new value $S(i)$ arrives from the stream
- 2 **then** Update R^w, σ^w ;
- 3 IsOutlier($R^g, \sigma^g, S(i)$);
- 4 **if** ($S(i)$ is added to R^w);
- 5 **then** Send $S(i)$ to parent with probability f ;
- 6 **if** a new update of R^g and σ^g is received
- 7 **then** Update R^g and σ^g
- 8 **return** ;

Figure 4.9: Leaf Process algorithm

BLACK PROCESS()

- 1 **if** a new message from a child node arrives
- 2 **then if** (message is added to R^b)
- 3 **then** Send message to parent with probability f ;
- 4 **if** (leader at the highest level)
- 5 **then** Send updates of R^b and σ^b to all the children;
- 6 **return** ;

Figure 4.10: Black Process algorithm

In order to specify the requirements posed by these data models, several other outlier detection algorithms have to be further examined. The most useful ones are presented in [SLMJ07, ZC06, ZWL07].

In [SLMJ07], Sheng *et al.* propose a distance-based outlier detection approach that extracts hints

ISOULIER(SAMPLE R , STDDEV σ , POINT P)

- 1 Use R and σ to estimate $N(p, ar)$ and $N(ar(2j - 1), ar)$ for each interval j ;
- 2 **if** (isMDEFOutlier())
- 3 **then** Mark P as an outlier;
- 4 **return** ;

Figure 4.11: Outlier algorithm

from the network by using histograms so as to abstract the distribution. The goal is to find global outliers over one dimensional data collected by all sensors, rather than detecting abnormal sensor readings in local proximity. The authors propose two schemes based on two definitions of outliers as stated in [KN98] and [RRS00]. The first scheme is based on the first definition and detects a point as an outlier if the distance from its k th nearest neighbour, D^k , is not smaller than a user specified distance d . The second proposed scheme which uses equi-width histograms, is based on the definition of outliers presented in [RRS00], where a point P is an outlier if no more than $n - 1$ points have a distance from their k th neighbour which is greater than the distance from the point's P k th neighbour. Both of these schemes can be extended to support online outlier detection, since each sensor can report the necessary histogram changes due to the newly generated data and the sink can update the histogram periodically.

In [ZC06] Zhuang *et al.* define two kinds of outliers, the short simple outlier, and the long segmental outliers. According to them, a “short simple outlier is a high frequency noise or error. It is usually represented as an abnormal sudden burst and depression, which is dissimilar to the other part of the sensing series” and a “long segmental outlier, is the erroneous sensed readings that last for a certain time period”. The outlier cleaning in [ZC06] includes outlier correction and outlier removal, which are performed by each sensor along the routing path to the sink, starting from the leaf nodes, and takes advantage of the similarity in both the temporal and the spatial dimension. As the authors mention, the proposed approach can be applied to a centralised environment too. The main disadvantage of this approach is that the same distribution for the data of the whole area is required. Consequently, a sophisticated statistical model is required, such as the one proposed in [SPP⁺06]. As an extension to [ZC06], in [ZWL07] Zuang *et al.* propose a method for cleaning sensor data in a hierarchical topology of sensor nodes. The proposed method maintains a weighted moving average that quickly reflects rapid changes in the data distribution. The weighted moving average is computed at the sink and provides clean data which can be used for query answering. This algorithm does not identify outliers and does not provide a formal definition about what an outlier is. However, by using Kalman filters and linear regression as predictors, the algorithm turns out to be more capable of following anomalies in the data and recovering after they disappear than simple moving average approaches that have been proposed.

4.1.3 Centralised Outlier Detection

A general and simple solution for mining outliers is to follow the centralised data analysis approach and collect all the data in a single site, the sink. Afterwards, any of the proposed algorithms for centralised outlier detection can be effectively executed. Such algorithms are presented in [KN98, RRS00, RWP04, PKGF03, BKNS00, GPO06, BS03, RKV95].

All these algorithms and techniques can also operate in a centralised manner, over a database with data consisting of sensor readings. We consider this fact important, because it enables these techniques to be used in cases where there is no access to the sensor network infrastructure, as happens in the case of flood warning in Southern UK use case scenario. Thus, no in-network processing can take place and all the data analysis techniques have to be executed centrally in a database. Other than that, data analysis in centralised settings provides us with the ability to apply the outlier detection and sampling algorithms on historical archives except for only analyzing



real-time generated data, as happens in in-network processing. This fact turns out to be of great importance and usefulness when one wants to compare environmental phenomena in various periods of time or just to observe the behavior of the monitored values in the past. Based on the data analysis techniques existing in the literature and presented in this section, the requirements of WP7, mentioned in Section 3.1.2, can be efficiently addressed.

4.2 Query Processing over Stored Data

Stored data will be accessed in the SemSorGrid4Env project using the reference implementations of the WS-DAI realisations for stored data [AAK⁺06, ACK⁺06, AHK⁺06, GGP08]. That is, the OGSA-DAI implementations of WS-DAIR and WS-DAIX [AAB⁺05b], and the reference implementation of WS-DAI-RDF [GGP08]. The following will discuss the OGSA-DAI approach since the WS-DAI-RDF implementation is still in a state of flux, but should be broadly similar.

OGSA-DAI exposes data sources, such as relational databases and XML document stores, as Web services in a standardised manner, enabling heterogeneous sources (at the level of schema and underlying vendor) to be queried, updated and integrated in a uniform manner. This is done by reconciling differences in the methods of accessing data, data format and result delivery mechanism.

The key features provided by OGSA-DAI are:

- Standardised interfaces for accessing data;
- Service metadata with information about underlying DBMS (such as the vendor type), capabilities of the DBMS, capabilities of the service itself, and the database schema;
- Managing database sessions for each consumer of the service;
- Different data delivery modes. For example, with synchronous delivery, a consumer sends a request, receives the results directly from the OGSA-DAI service. With asynchronous delivery, results may be sent to a different service, or stored until retrieval by the consumer.

The latest implementation of OGSA-DAI (version 3.1) is compliant with the Web Services Resource Framework (WS-RF) specification [GT06] and the WS-DAI specifications [AAK⁺06, ACK⁺06, AHK⁺06], and is available at [OGS09]. Note that OGSA-DAI can be used for relational databases, XML document stores, or files. This is supported by implementations of the relevant WS-DAI realisation, e.g. WS-DAIR for relational sources, and WS-DAIX for XML document stores.

In Section 5.3.1, a detailed description is given of the operations exposed by OGSA-DAI.

4.3 In-network Query Processing

The SNEE (SNEE for Sensor NETwork Engine) query optimiser is a novel query optimiser for sensor networks which combines a rich, expressive query language with a software architecture based on traditional distributed query processing techniques. The optimisation steps cover all the query optimisation phases that are required to map a declarative query in the SNEEq_l language to executable code. Firstly, a brief description of the SNEEq_l language is provided, with some example queries (Section 4.3.1), and then a brief overview of the SNEE query stack is presented (Section 4.3.2), with an illustration of the progression of compilation for an example query.

```
tree:sensed (id:int, locx:int, locy:int, ts:time, smoke:boolean,
            temperature:float, rhumidity:float)
soil:sensed (id:int, locx:int, locy:int, ts:time, moisture:float)
wind:sensed (id:int, locx:int, locy:int, ts:time, speed:float,
            direction:float)
rain:sensed (id:int, locx:int, locy:int, ts:time, level:float)
```

Figure 4.12: Example SNEEqI schema definition.

```
Q1:      SELECT RSTREAM w.id, w.speed, w.direction
         FROM   wind[NOW] w;

Q2:      SELECT RSTREAM wavg.wsp / (10 - 0.25* (tavg.temp - tavg.rh))
         FROM   (SELECT avg(w.speed) as wsp
                 FROM   wind[NOW] w) wavg,
                 (SELECT avg(t.temperature) as temp, avg(t.rhumidity) as rh
                 FROM   tree[now] t) tavg;

Q3:      SELECT RSTREAM t.id, w.speed, w.direction
         FROM   wind[NOW] w, tree[NOW] t
         WHERE  t.smoke > 0
         AND    sqrt((t.locx - w.locx)^2 + (t.locy - w.locy)^2) <= 40;

QoS 1:   {Acquisition interval = 1 min,
         Delivery Time <= 1 min}

QoS 2:   {Acquisition interval <= 15 min,
         Delivery Time <= 15 min}
```

Figure 4.13: Queries and QoS expectations.

4.3.1 Query Language

SNEEqI [BGFP08] is a declarative query language for sensor networks inspired by expressive classical stream query languages such as CQL [ABB⁺03]. A rich language is used even though our target delivery platform consists of limited devices. As with classical databases, a schema is defined using a data definition language (DDL), described in [BGFP08] in the case of SNEEqI, that defines the *extents*, or schema objects, that queries may be posed against. In SNEEqI, extents may be acquisitional streams, event streams or stored relations. In a schema, the name and type of extent is specified, as are the names and types of the attributes that comprise the extent. Figure 4.12 presents a schema defining four acquisitional streams, viz., *tree*, *soil*, *wind* and *rain*, which comprise sets of sensors with different sensing modalities, inspired by the SemSorGrid4Env fire use-case application scenario [LDI09]. Each acquisitional stream is bound to a subset of physical nodes in the sensor network, decoupled from the the schema definition which is at the logical level. The first four attributes in the tuple types defined contain information relating to the location and time of the sensor reading. The *id* attribute contains a unique identifier of the source node that performed the sensor readings. Co-ordinates representing the location are represented by the *locx* and *locy* attributes, and a timestamp denoting the time that the sensor reading was taken is contained in the *ts* attribute. The remaining attributes in the tuples correspond to sensed values, i.e. smoke, relative humidity, wind speed etc.

Example SNEEqI queries are shown in Figure 4.13 against the schema in Figure 4.12, based on the forest fire monitoring scenario described in Section 3.1.1. *Q1* requests the node identifier, speed and direction readings for the wind sensors in the sensor network deployment. *Q2* calculates a fire risk rating index, as proposed in [SMWG09], used to obtain an indication of the risk of a fire,



and its destructive potential. In this case, the danger rating D is computed as $D = \frac{U}{10-0.25(T-H)}$, where D is the danger rating, U is the wind-speed (km/h), T is the temperature ($^{\circ}\text{C}$) and H is the relative humidity (%). This index uses the fact that certain climatic conditions, i.e. higher temperature, higher wind speed, and lower humidity, lead to increased risk of fire and destructive potential of a fire. The average for each of the temperature, humidity and wind speed variables is first obtained using subqueries, and then the formula is applied. $Q3$ computes, for each location where smoke has been detected, the wind speed/direction recorded nearby, in order to predict in what direction the fire is likely to spread. This query takes readings from the wind and tree streams, and joins them if smoke is detected at the given tree, and the distance between the tree sensor and the wind sensor is 40m or less.

In SNEEqL, the only structured type is `tuple`. The primitive collection types in SNEEqL are: `relation`, an instance of which is a bag of tuples with definite cardinality; `window`, an instance of which is a relation whose content may implicitly vary between evaluation episodes; and `stream`, an instance of which is a bag of tuples with indefinite cardinality whose content may implicitly vary throughout query evaluation. As in CQL, operations construct windows out of streams and vice-versa.

In SNEEqL, windows are used to convert from streams to relations, relational operators act on those relations, and stream operators add the resulting tuples into the output stream. Window definitions are of the form `WindowDimension [SLIDE] [Units]`, where the `WindowDimension` is of the form `NOW` or `FROM Start TO End`, where the former contains all the tuples with the current time stamp, and the latter contains all the tuples that fall within the given range. The `Start` and `End` of a range are of the form `NOW` or `NOW -Literal`, where the `Literal` represents some number of `Units`, which is either `ROWS` or a time unit (`HOURS`, `MINUTES` or `SECONDS`). The optional `SLIDE` indicates the gap in `Units` between the `Start` of successive windows; this defaults to the acquisition rate specified. The results of relational operators are added to the result stream using the relation-to-stream operators from CQL, namely `ISTREAM`, `DSTREAM` and `RSTREAM`, denoting *inserted-only*, *deleted-only* and *all-tuples*, respectively.

Two examples of Quality of Service (QoS) specifications which may be associated with a query are also shown in Figure 4.13. *QoS 1* states that the acquisition interval (i.e. time interval between sensor readings) should be one minute, and that data must be received by the user within 1 minute of acquisition. With *QoS 2*, sensor readings must be made at least every 15 minutes, and tuples are required by the user within 15 minutes of acquisition. As discussed in Section 6.5, work is nearing completion to enable the SNEE optimiser to be fully QoS-aware, in which more QoS metrics will be able to be included in constraints, and also, an optimisation goal will be able to be specified in the OS specification. Currently, the implicit optimisation goal is to maximise network lifetime.

A detailed description of the SNEEqL language, including a formal semantics, is given in [BGFP08].

4.3.2 Query Stack

The SNEEqL compilation/optimisation stack is illustrated in Figure 4.14, and comprises three phases. The first two are similar to those familiar from the two phase-optimisation approach, namely *Single-Site* (comprising Steps 1-3, in gray boxes) and *Multi-Site* (comprising steps 4-6, in white, solid boxes). The *Code Generation* phase grounds the execution on the concrete software and hardware platforms available in the network/computing fabric and is performed in a single step, Step 7 (in a white, dashed box), which generates executable code for each site based on the distributed Query Execution Plan (QEP), routing tree and agenda. The current implementation of SNEEqL generates nesC [GLvB⁺03] code for execution in TinyOS [HSW⁺00], a component-based, event-driven runtime environment designed for wireless SNs. nesC is a C-based language for writing programs over a library of TinyOS components. The optimiser consists of 15K lines of Java. A detailed description of the SNEE optimiser is given in [GBJ⁺09].

Metadata required by the optimiser to inform the generation of a query plan includes the classical kinds of metadata used by query optimisers (e.g. schematic information, statistics, etc.) as well

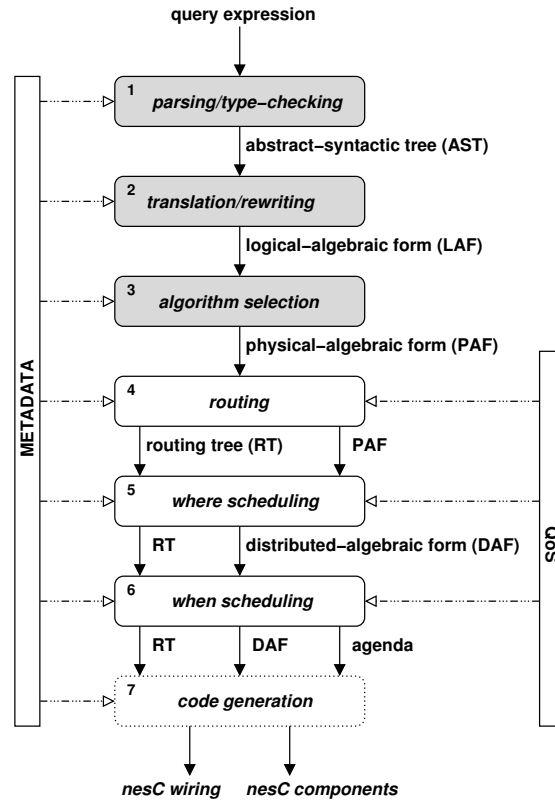


Figure 4.14: SNEEqI Compiler/Optimiser Stack.

as a detailed description of the network fabric. Figure 4.15 depicts an example network which depicts a sensor network deployment in a forest used for fire prediction/detection, as described in Section 3.1.1. Nodes are assumed to be located evenly on a grid, 20m apart from each other. The sink node, where query results are to be delivered to, is node 0 in the top-left hand corner. A graph $N = \langle V, E \rangle$ is derived from the network geometry in Figure 4.15, where V is the set of vertices in the graph (i.e. the sites of the sensor network), and E is the set of edges (i.e. the pairs of nodes between which direct interaction is possible). Note that each edge $(u, v, energy) \in N.E$ connecting site u to site v has an energy weighting associated with it. The *energy* weighting reflects the radio transmission power for the radio in site u to transmit to site v , and is expected to increase proportionally with the square of the distance. It is assumed that this is collected and updated by code components with network management capabilities.

The streams generated by the sensors in the deployment correspond to logical extents in the schema described in Figure 4.12, and the source nodes for these schemas are described in Figure 4.16. For example, the *Wind* stream, which contains the sensed attributes *speed* and *direction*, is acquired at sites 4, 10, 22 and 24, each labeled with **W** in Figure 4.15. The *id* attribute is the identifier unique to each site, *locx* and *locy* provide the co-ordinates in geometric space of the source site, and *time* is a timestamp corresponding to the sensor reading. The second set of attributes provide a spatio-temporal context for the sensed attributes (i.e. *speed* and *direction* in this case). Note that all sites may contribute computing (e.g. processing intermediate results) or communication (e.g. relaying data) capabilities. Metadata about the network is assumed to be populated by a network management layer, such as Moteworks (<http://www.xbow.com>), which also provides services such as network formation, self-healing, code dissemination etc. Throughout the steps in the query stack, size-, memory-, energy- and time-cost models are used by the optimiser to reach motivated decisions. These cost models are described in more detail in [BGFP09].

Single-site optimisation is decomposed into components that are familiar from classical, centralised query optimisers, using well-established techniques. In essence: *Step 1* checks the validity of the query with respect to syntax and the use of types, and builds an abstract syntax tree to represent

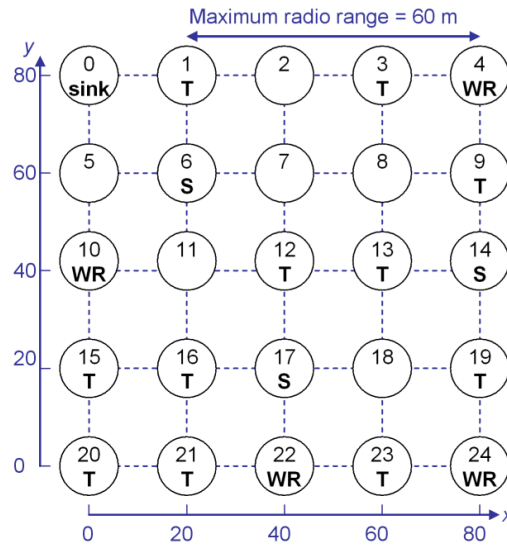


Figure 4.15: The sensor network.

```
Stream sources
=====
tree {1, 3, 9, 12, 13, 15, 16, 19, 20, 21}
soil {6, 14, 17}
wind {4, 10, 22, 24}
rain {4, 10, 22, 24}
```

Figure 4.16: Schema sources.

the query; *Step 2* translates the abstract syntax tree into a logical algebra, the operators of which are reordered to reduce the size of intermediate results; and *Step 3* translates the logical algebra into a physical algebra, which, e.g. makes explicit the algorithms used to implement the operators. The output of this phase is the *physical-algebraic form*, depicted in Figure 4.17 for the example query compilation. As can be seen in the figure, aggregates (such as average) are computed in three phases (*initialise*, *merge* and *evaluate*), each implemented as a separate physical operator, as is done in TinyDB. This helps reduce radio traffic and allows the computation of different operands in the algebraic expression to be scheduled separately.

Multi-site optimisation involves generating a distributed plan by breaking up the the physical-algebraic form (PAF) into QEP fragments for evaluation on specific nodes in the network. During this phase, consideration is given to routing (the means by which data travels between nodes within the network), the placement of operators in the PAF, and duty cycling (when nodes transition from being switched on and engaged in specific tasks, and being asleep, or in power-saving modes). The following paragraphs provide more details about each of these steps.

Routing. Step 4 determines a routing tree (RT) for communication links that the data flows in the PAF can then rely on. This is achieved by computing a Steiner tree, i.e. a tree of minimal cost derived from the network topology graph with a required set of nodes, using any additional nodes which are necessary. The resulting routing tree for *Q3* over the network given in Figure 4.15 is depicted in Figure 4.18. It can be seen that it includes all the sources nodes required by the query (i.e. the sources of the *wind* and *tree* extents), and the sink node (node 0), plus additional relay nodes (e.g. node 11).

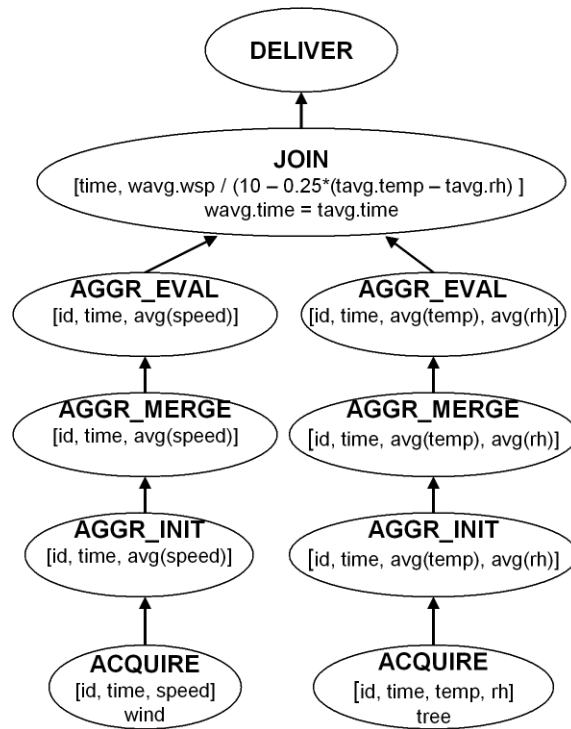


Figure 4.17: Physical-algebraic form for $Q3$.

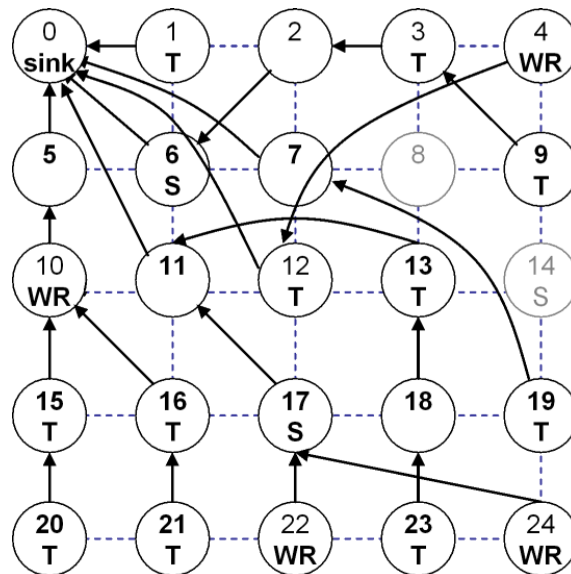
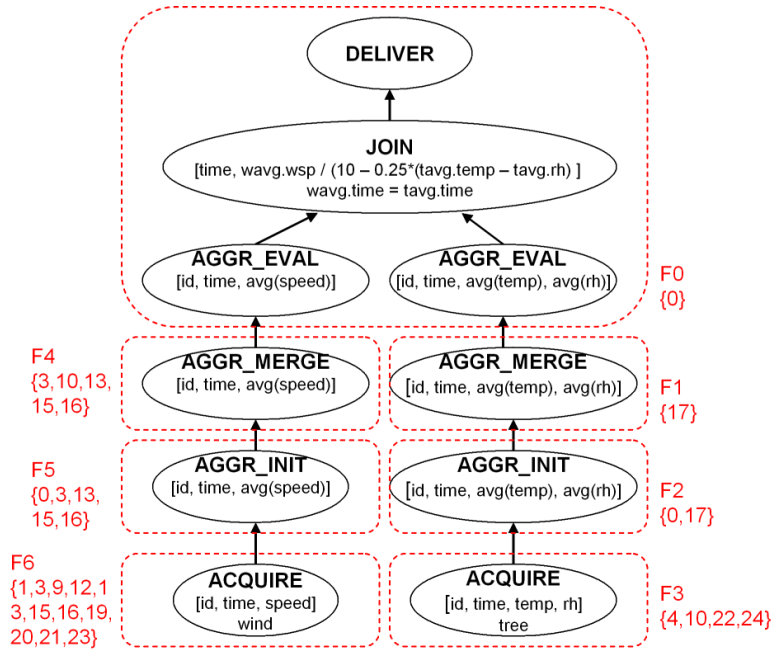


Figure 4.18: Routing tree for $Q3$


 Figure 4.19: Distributed-algebraic form for Q_3 .

Where-scheduling. Step 5 breaks up the QEP into fragments by inserting exchange operators [Gra90], and decides which QEP fragments are to run on which RT nodes. An exchange operator encapsulates all of control flow, data distribution and inter-process communication and is implemented in two parts, referred to as producer and consumer. This step is carried out using a heuristic algorithm that places fragments with the aim to reduce the amount of data transmitted. This step results in the distributed-algebraic form (DAF), in which each fragment is allocated to a set of sites. It can be observed that instances of F_3 and F_6 have been created at multiple sites, as these fragments contain location-sensitive ACQUIRE operators, whose placement is dictated by the schema definition in Figure 4.16. Also, a single instance of attribute-sensitive F_0 has been created, combining fragments $F_{0_i} \dots$ to $F_{0_{i_v}}$ into a single fragment as they all execute on the same site. F_0 is assigned to the sink site, the deepest confluence site where tuples from both F_1 and F_4 are available (as it is a non-location-sensitive fragment and has been placed according to its expected output size, to reduce communication). Note also the absence of site 11 in Figure 4.19 wrt. Figure 4.18. This is because site 11 is only a relay node in the routing tree.

When-scheduling. Step 6 stipulates execution times for each fragment. The approach adopted is to build an agenda that, insofar as permitted by the memory available at the site, and given the acquisition rate and the maximum expected delivery time for the query, buffers as many tuples as possible before transmitting. This step is carried out using a heuristic algorithm that repeatedly schedules the acquisitional (leaf) fragments until the maximum possible level of buffering is reached, and schedules remaining tasks according to the precedence constraints implied by the DAF and RT. The aim is to be economical with respect to both the time in which a site needs to be active and the amount of radio traffic that is generated. The result of this step is an agenda. The agenda can be conceptualised as a matrix, in which the rows, identified by a relative time point, denote concurrent tasks in the sites which identify the columns. For Figure 4.19, the computed agenda is shown in Figure 4.20, where $\alpha = 60s$, $\beta = 10$ and $\delta \leq 10$ min. Thus, a non-empty cell (t, s) with value a , denotes that task a starts at time t in site s . In an agenda, there is a column for each site and a row for each time when some task is started. Thus, if cell $(t, s) = a$, then at time t in site s , task a is started. A task is either the evaluation of a fragment (which subsumes sensing), denoted by F_n in Figure 4.20, where n is the fragment number, or a communication event, shown in a shaded cell, and denoted by tx or rx , i.e. respectively, tuple transmission, or tuple reception. Leaf fragments F_3 and F_6 are evaluated β times in each agenda evaluation, as indicated by the

start time	site 20	site 15	site 21	site 16	site 10	---	Site 3	Site 2	site 6	site 1	site 0
0:00:00.000	F6 ₁	F6 ₁	F6 ₁	F6 ₁	F3 ₁	---	F6 ₁			F6 ₁	
0:01:00.000	F6 ₂	F6 ₂	F6 ₂	F6 ₂	F3 ₂	---	F6 ₂			F6 ₂	
---	---	---	---	---	---	---	---	---	---	---	---
0:09:00.000	F6 ₁₀	F6 ₁₀	F6 ₁₀	F6 ₁₀	F3 ₁₀	---	F6 ₁₀			F6 ₁₀	
0:09:00.042	tx	rx				---					
0:09:00.087		F5				---					
0:09:00.117		F4				---					
0:09:00.134		tx			rx	---					
0:09:00.170			tx	rx		---					
---	---	---	---	---	---	---	---	---	---	---	---
0:09:04.227						---			tx		rx
0:09:05.245						---				tx	rx
0:09:06.310						---					F2
0:09:06.375						---					F5
0:09:06.450						---					F0
0:09:09.600						---					

$\alpha = 1 \text{ min}$
 $\beta = 10$
 $\delta \leq 10 \text{ min}$

Figure 4.20: The agenda for $Q3$.

subscript on the leaf fragment tasks. Blank cells denote the lack of a task to be performed at that time for the site, in which case, an OS-level power management component is delegated the task of deciding whether to enter a energy-saving state. Note that, due to the large size of the agenda for this query, in Figure 4.20, some rows and columns are omitted from the agenda, as indicated by the dashes.

4.4 Summary

This chapter has described existing work which is being used as a basis for WP2's contributions to the SemSorGrid4Env project to meet the requirements of the other work packages described in Chapter 3. The extensions required to meet these requirements are described in Chapter 6.



5. Architecture

This chapter describes the SemSorGrid4Env architecture, and how the functionality described in Chapters 4 and 6 is going to be bundled up into services compliant with the SemSorGrid4Env architecture so that they can interact with components from the other work packages. Specifically, it will indicate how the requirements identified in Chapter 3 are satisfied by the proposed implementation of the data services.

5.1 Architecture Overview

The SemSorGrid4Env software architecture defined in Deliverable D1.3v1 [GGF⁺09] specifies a Web services architecture where services may be made available as REST services [Fie00] or traditional service-oriented Web services [ACKM04]. The Web services and their interactions are depicted in Figure 5.1. For simplicity the diagram depicts a generalised Data Service, however two separate types of data service have been defined: a Stored Data Service and a Streaming Data Service. The SemSorGrid4Env architecture defines the service-oriented interfaces of four core Web services. The core services together with application specific services which have been identified in Deliverable D5.1 [PRMS09] have been classified into three tiers—application, middleware, and data. Note that any service or application may call any other service regardless of which tier it is nominally classified in.

Data Tier. The data tier provides two Web services—the Stored Data Service and the Streaming Data Service—which make it possible to publish data stored in traditional data sources, e.g. databases, and streaming data, e.g. generated by a sensor network. The data services, be they streaming or stored, *virtualise* a real data source which the data service accesses through a connectivity bridge. For the case of a relational database this would most likely be JDBC [Sun09]. A similar API, which abstracts the intricacies of interacting with the hardware and software combinations of sensor devices, does not exist for sensor networks. Currently bespoke mechanisms need to be developed, but there is scope to develop a Sensor Network Connectivity Bridge to provide similar abstractions for sensor networks as JDBC does for database management systems. Details about the functionality provided by the data source tier, and technology used, will be provided in subsequent sections. The data tier also provides an Intermediary Service which can be used to support consumers of event subscriptions.

Middleware Tier. The middleware tier provides two services that provide additional functionality to the data published by the data tier services. The Registration and Discovery Service provides mechanisms by which data sources, and services which process data, can be found according to their capabilities or data content. The Integration and Query Processing Service allows the data from one or more data service to be accessed as a *virtual data source* using a different model of the data, e.g. an ontology of the domain of interest.

Application Tier. The application tier provides functionality specific to a particular application domain. For example, a frontend to the Registration and Discovery Service will be developed that allows flood use case mash-up developers to easily discover suitable data sources without needing to learn the `stSPARQL` query language used by the registry implementation [KKK09]. The domain specific services have been identified in Deliverable D5.1 [PRMS09].

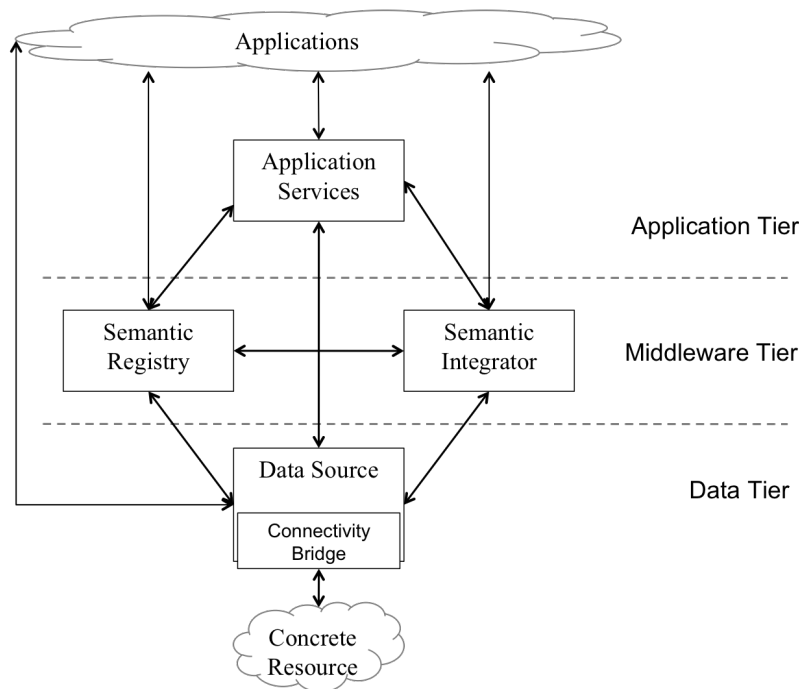


Figure 5.1: The SemSorGrid4Env software architecture.

5.2 Data Management Interfaces

The SemSorGrid4Env architecture defines a set of interfaces which allow the Web services to interact. The Web services are defined by the interfaces that they make available. To enable “*well-formed and well-informed*” service calls between the Web services, all services are required to provide the **Service Interface** for discovery of the methods, and the associated metadata for the service. The interfaces provided by the Web services of the data tier are shown in Figure 5.2. The interfaces with solid boxes **MUST** be provided by the Web service while the interfaces with dashed boxes are **OPTIONAL**.

The specification of the interfaces can be found in D1.3v1 [GGF⁺09]. Where appropriate, the interfaces have been directly taken from the relevant WS-* standards: WS-ResourceFramework (WS-RF) for resource and lifetime management [Ban06]; WS-Data Access and Integration (WS-DAI) for data access and query processing [AAK⁺06]; and WS-BaseNotification (WS-N) for subscriptions and notification messages [GHM06]. A brief overview of the functionality provided is given below.

Service Interface: Provides the operations of the WS-ResourceFramework so that clients, which may be other services or applications, can discover the operations and properties of the Web service. This enables clients to make “*well-formed*” and “*well-informed*” service calls, i.e. they can discover what operations are made available, and the input parameters required, as well as discover the properties of the specific implementation, e.g. whether a **Stored Data Service** supports concurrent access.

Query Interface: Provides the operations of the WS-DAI standard to allow data to be retrieved using a formal query language such as SNEEqL, SPARQL, SQL, etc. The operations, which are provided by the specific realisations of the WS-DAI standard (i.e. WS-DAIR, WS-DAI-RDF, and WS-DAI-Streaming), enable two mechanisms for returning data. For the case of the **Stored Data Service** the answer to a query can either be sent directly back in response to the query message or indirectly through a data resource that exposes the **Data Access Interface**, see below. For the **Streaming Data Service** the current answer to the query can

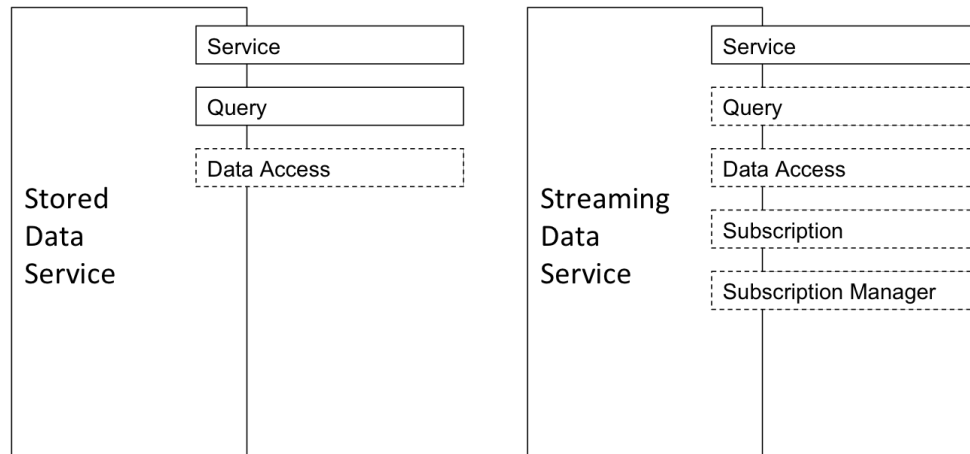


Figure 5.2: The interfaces exposed by the data tier services.

be returned directly and the execution of the query is assumed to cease, or if a continuous response is desired then it must use an indirect mechanism to return the query answers. This can either be through a *pull-based* mechanism using the Data Access Interface, or a *push-based* mechanism using the Subscription and Subscription Manager Interfaces.

Data Access Interface: Provides the operations of the WS-DAI standard for retrieving and iterating over a dataset that is stored in a specific data model, e.g. relational or RDF. The operations are drawn from specific realisations of WS-DAI, e.g. the relational realisation WS-DAIR [ACK⁺06] or the RDF realisation WS-DAI-RDF [GGP08]. For the Streaming Data Service provided by the WS-DAI-Streaming realisation [GGFP09], the Data Access Interface is used to provide a *pull-based* delivery mechanism, i.e. a data consumer can periodically request data using this interface.

Subscription Interface: Provides the WS-BaseNotification operation for subscribing to notifications about events of interest. The interface enables a data consumer to receive a stream of data which is *pushed* to the consumer.

Subscription Manager Interface: Provides the WS-BaseNotification operations for managing a subscription, e.g. the lifetime of the subscription. This interface supports the Subscription interface and MUST be provided whenever the Subscription interface is provided.

5.3 Data Management Services

This section describes the implementations of the data tier services that will be used in the SemSorGrid4Env project for the publication of data sources. Note that the implementation of the Stored Data Service exists already and can just be used in the project. However, the Streaming Data Service still needs to be implemented.

5.3.1 Stored Data Service

The interfaces exposed by the Stored Data Service directly correspond to the WS-* standards used in the OGSA-DAI implementations of the WS-DAI standard [AAB⁺05b, OGS09]. Thus, the OGSA-DAI implementation will be used to provide the Stored Data Service. This section describes how the relational realisation of the OGSA-DAI service responds to calls from the various operations it implements: the XML interaction is similar but uses operations from the WS-DAIX realisation. In the context of this section, an *OGSA-DAI service* is a service which exposes the



interfaces of a stored data service and handles requests to query it, a *client* refers to an entity which sends the initial data request to the OGSA-DAI service, an *intermediary* is a service temporarily created by the OGSA-DAI service in order to deliver results, and a *consumer* is an entity different to the client which receives the results.

Data Source Properties. To retrieve the properties document of a data source, the client sends a `GetDataResourcePropertyDocument` message. The OGSA-DAI service requests the required information from the underlying database (such as the database schema), and returns a property document to the client containing metadata about the underlying database.

Direct Query Execution. A client can use the `SQLExecute` operation to pose a query over the relational data source and have the answer sent directly back. The `SQLExecute` request contains the SQL query, and any parameters needed, together with the format that the answer should be returned in. The OGSA-DAI service passes the query to the underlying database, which is optimised and subsequently evaluated. Results are passed back to the OGSA-DAI service, which sends them directly to the client in the format requested.

Indirect Query Execution. A client can use the `SQLExecuteFactory` operation to pose a query over the relational data source and have the answer stored in an intermediary data resource from which it can later access it. The `SQLExecuteFactory` request contains the SQL query, and any parameters, together with details of how the later data access should be conducted. The OGSA-DAI service creates an intermediary which will be used to return the results using the Data Access Interface, and returns a handle of the intermediary to the client. The OGSA-DAI service then passes the query to the underlying database, which is optimised and subsequently evaluated. Results are passed back to the OGSA-DAI service, which sends them directly to the intermediary.

Data Access. At the desired time, the consumer service (which may not necessarily be the client) retrieves the results from the intermediary using the `SQLResponse` operations defined in the WS-DAIR standard [ACK⁺06]. From the point of view of SemSorGrid4Env, where most external data will only be read, the most important of these operations are:

- `GetSQLResponsePropertyDocument`: which allows the properties of the response set to be retrieved;
- `GetSQLResponseItem`: which returns a specified number of items from the answer set;
- `GetSQLRowset`: which gets a specified number of rows from the answer set.

5.3.2 Streaming Data Service

The streaming data service will be implemented as part of the SemSorGrid4Env project and will be based on the SNEE continuous query compiler and evaluator, described in Section 4.3 and the extensions that are described in Section 6. It is anticipated that the SNEE compiler can be extended to operate in a unified way over both acquisition streams, e.g. coming from sensor networks with in-network query processing capabilities, and non-acquisition streams, e.g. coming from sensor networks which simply gather data or external data sources. That is, there is a single SNEE implementation for both types of data stream. The Streaming Data Service will provide a wrapper which will receive and process the Web service messages and then pass the query onto the SNEE query evaluation engine. More details of the intended implementation of the extended SNEE compiler can be found in Chapter 6.

The Web service provides a streaming realisation of the WS-DAI standard (WS-DAI-Streaming) [GGFP09], and will support the SNEEql continuous query language [BGFP08]. The interaction with these operations are described in the following paragraphs.



Data Source Properties. To retrieve the properties document of the data source, the client sends a `GetDataResourcePropertyDocument` message. The Web service gathers the required information from the underlying processing engine, and returns a property document to the client containing metadata about the underlying data source.

Direct Query Execution. A client can use the `StreamExecute` operation to pose a continuous query over the streaming data source and receive the answer directly. The request contains the SNEEqL query that is to be executed, together with any parameters (e.g. QoS). Due to the semantics of continuous queries, only a limited set of queries can be supported for one-off execution, returning the entire answer set in response to the service call. The service implementation does this on a best effort basis or returns a fault if it cannot execute the supplied query in this manner. The recommended mechanism for querying a streaming source is the indirect query execution message.

Indirect Query Execution. A client can use the `StreamExecuteFactory` operation to pose a continuous query over the streaming data source. The request contains the SNEEqL query, together with any parameters (e.g. QoS), and declares the mechanism by which the answer stream should be delivered. If a pull-based mechanism is used, then the client receives the endpoint reference of an intermediary data resource from which it can later access it using the Data Access operations described below. If a push-based mechanism is used, then the client receives the endpoint reference for a `SubscriptionManager` resource which will manage the delivery of notification messages to the notification endpoint reference declared in the initial request. In both cases, the Web service passes the SNEEqL query and its parameters to the query evaluation engine. When the query evaluation engine starts generating query answers, these are made available through the Data Access interface or pushed to the client as appropriate.

Data Access. If the client has chosen to use pull-based delivery then it can use the operations of the Data Access interface to retrieve tuples. Currently, the stream realisation of the data access interface offers the following operations:

- `GetStreamResponsePropertyDocument`: which allows the properties of the response set to be retrieved, e.g. details of the tuple dropping policy if tuples are generated at a faster rate than the client requests them;
- `GetStreamOldestItem`: which returns a specified number of items, either in time (e.g. the oldest 30 seconds worth of tuples) or a number of tuples, from the answer set, starting with the oldest items. If the optional `keep` argument is present then the items are not removed from the answer set, i.e. they will remain available for a future read;
- `GetStreamNewestItem`: which returns a specified number of items, either in time (e.g. the newest 30 seconds worth of tuples) or a number of tuples, from the answer set, starting with the newest items. If the optional `keep` argument is present then the items are not removed from the answer set, i.e. they will remain available for a future read.

5.4 Meeting the Requirements

This section discusses how the requirements identified in Chapter 3 have been met by the SemSorGrid4Env architecture. Following the structure of Chapter 3, this discussion is split into “Meeting the Requirements of the Use Cases”, and “Meeting the Requirements of the Technical Work Packages.”



5.4.1 Meeting the Requirements from the Use Cases

This section explains how the requirements identified in Section 3.1, summarised in Table 3.1, have been met by the data tier services. Specifically, it addresses the Stored and Streaming Data dimensions in Table 3.1. The requirements for the Query Processing and Data Analysis dimensions in Table 3.1 are met by the implementations wrapped by the data source services: the details of which are discussed in Chapter 6.

Stored Data

The SemSorGrid4Env **Stored Data Service** enables existing stored sources, such as databases with historical sensor data or related information, to be published and queried. The **Stored Data Service** supports direct delivery mode through the **GenericQuery** operation, which for the relational realisation is implemented by the **SQLExecute** operation. Indirect delivery is supported by the **GenericQueryFactory** operation, which for the relational realisation is implemented by the **SQLExecuteFactory** operation.

Streaming Data

The SemSorGrid4Env **Streaming Data Service** enables streaming data sources, such as sensor networks, to have their data streams published and queried. The **Streaming Data Service** supports query requests through the **Query Interface**. The **Query Interface** provides support for both direct delivery of answers through the **StreamExecute** operation, and indirect delivery through the **StreamExecuteFactory** operation. Query answer streams can be delivered through pull-based mechanisms using the **Data Access Interface** or push-based mechanisms supported by the **Subscription** and **Subscription Manager** interfaces.

It is also possible to access an event stream using either a pull-based or push-based mechanism. These are also supported by the **Data Access Interface** or the **Subscription** and **Subscription Manager** interfaces respectively. Indirect delivery for event streams is supported by the **Intermediary Service**.

5.4.2 Meeting the Requirements from the Technical Work Packages

This section explains how the requirements identified in Section 3.2 have been met by the architecture.

Architecture and Infrastructure

The architecture and infrastructure place certain expectations on the Web service interfaces that will be offered by the data source services, as described in Sections 5.2 and 5.3. The implementations of the services will provide these interfaces.

Registry

The specification of the architecture interfaces in D1.3 [GGF⁺09] provides a mechanism for publishing metadata about the implementing Web service, i.e. information about the area of coverage of a data source, and for linking the metadata to concepts in an ontology. This information is made available by the **Service Interface** which all SemSorGrid4Env Web service implement. Thus, the data source services are able to publish information about themselves that can be incorporated into the **Registry Service**.



Semantic Data Integrator

The semantic data integrator requires that data sources are discoverable through the registry and provide a common mechanism for querying and retrieving data, both stored and streaming. The discovery of the data sources in the registry is covered above. The data services, both stored and streaming, provide a common mechanism for querying data through the **Query Interface**. Query answers may be retrieved directly or indirectly, and streams can be accessed either through pull-based or push-based mechanisms.

High-level Application Programming Interfaces

There is potentially the requirement within SemSorGrid4Env for data sources to make their data available for resource-oriented access through representational state transfer (REST). The interface to such services are limited to the HTTP operations GET, PUT, POST, and DELETE. The client of such services needs to interact with the service to discover what resources are available. This mechanism relies on the content of the specific source and the REST service providing links between resource representations. As such, the architecture deliverable [GGF⁺09] does not define REST services for SemSorGrid4Env, and neither are they discussed in this document. However, it should be straightforward for a data source publisher to provide such an interface for their content.

5.5 Summary

This chapter has given a brief overview of the SemSorGrid4Env architecture. It then described the data tier Web services that will be used to publish data sources and the implementations of these services that will be provided in the SemSorGrid4Env project. The section then discussed how the requirements for data management and analysis have been fulfilled by these services.

6. Future Work

In Chapter 4, existing relevant work was presented which will be extended by WP2 to meet the requirements of the other work packages described in Chapter 3. In this chapter, we describe the current limitations of the existing work, where applicable, and propose a series of tasks which will be undertaken to fulfil the requirements of SemSorGrid4Env. We consider in turn the case of data analysis, query processing over stored data sources, and query processing over streaming data sources. For each task, we present a brief reminder of the motivation for the task, a summary of work done so far, the steps required to carry out the task, as well as any dependencies that the task has in terms of the other tasks.

Data Analysis. With regard to the data analysis algorithms described in Section 4.1, the D3 outlier detection algorithm will be implemented as software for the SemSorGrid4Env infrastructure. Algorithms for other data analysis techniques may also be implemented, such as MGDD that is described in 4.1, or the ones proposed in [SLMJ07, ZC06, ZWL07], but their compatibility with SNEE is yet to be determined. This task is described in Section 6.1. They will then be integrated into the SNEE optimiser, as discussed in Section 6.7.

Query Processing over Stored Data Sources. For accessing stored data sources, OGSA-DAI will be deployed in order to provide virtualised access to these resources. It is not expected that any significant development will be required. It will constitute a Stored Data Service as defined by the SemSorGrid4Env architecture [GGF⁺09]. SNEE will also be extended to support relations, so that stored and streaming data can be combined within a single query as described in Section 6.3. SNEE will use OGSA-DAI or JDBC to access external stored relations.

Query Processing over Streaming Data Sources. With regards to streaming data sources, the SNEE optimiser will be used. Currently, it only supports query evaluation within the sensor network, for acquisitional streams. In Section 6.2, we describe how SNEE will be generalised into a query processor capable of evaluating fragments of the query plan over both robust and sensor network nodes. It will need to be able to contend with both acquisitional and event streams. In order to enable data integration to take place, the ability for SNEE to define views will also be added (Section 6.4).

A limitation of state-of-the-art query processors, including SNEE, is that currently, query plans are optimised considering a single optimisation goal, viz., network lifetime. Work is nearing completion on adding QoS-awareness in the optimiser, which will enable queries over acquisitional streams to be optimised for other goals, e.g. delivery time, an important concern for queries where data needs to be received quickly (e.g. an emergency response scenario). For event streams, the relevant QoS metrics will also need to be considered. The issue of QoS awareness is described in more detail in Section 6.5. Also, currently only a single query can be executed in SNEE. The ability to enable SNEE to compile multiple queries is also identified as being useful in Chapter 3 and the work this entails is outlined in Section 6.6.

Finally, appropriate network management functionality will need to be developed to support query processing and data analysis so that it can be deployed successfully for the fire use-case deployment. This is described in Section 6.8.

6.1 Implementation of Data Analysis Techniques within Sensor Networks and Robust Networks

The data analysis algorithms presented in Section 4.1 meet some of the requirements of the two use case scenarios regarding outlier detection in data generated from sensor networks. Some of them operate in an in-network manner, while some others follow centralised approaches. The algorithm that is chosen to be implemented in the context of WP2 in SemSorGrid4Env is named Distributed Deviation Detection (D3) and is presented in [SPP⁺06]. The main reason to choose D3 over other algorithms is the fact that that D3 uses a tree routing structure and is therefore compatible with SNEE. D3 performs online in-network outlier detection in wireless sensor networks and identifies distance-based outliers given a threshold value r , which represents a minimum distance that one point P must have from other points in order to be considered as an outlier. As mentioned in Section 4.1.2, due to the fact that the data points may exhibit different densities in different regions of the data space or across time, it is not always appropriate to use a single threshold value r to determine the outliers.

In our existing work, the whole implementation was based on a TAG-based simulator and required 5000 lines of Java code, including the necessary modifications to enable the hierarchical organisation of the nodes in the sensor network. However, the code that implements the D3 algorithm has a very small footprint. For instance, the kernel density estimation and outlier detection modules (that is, the code that would have to run on a sensor to implement the algorithm) required a total of 150 lines of Java code.

The outlier detection algorithm that will be implemented includes a sampling component. For the purposes of sampling we will implement two different sampling algorithms, namely “Chain Sampling” [BDM02] and “Reservoir Sampling” [DLK08], which are presented in Section 4.1.2.

Before testing these data analysis techniques on the motes of the sensor networks that will be deployed, some extensive simulations have to be run. The simulator which will be used for this purpose is TOSSIM [LLWC03], which is a TinyOS mote simulator. TinyOS is a component-based operating system programmed in nesC, which is also the programming language used in the development of SNEE. Consequently, everything described above will be implemented in nesC, in order to be compatible with TinyOS and SNEE.

Moreover, as mentioned above, some more algorithms regarding outlier detection or data cleaning may be implemented. These algorithms are proposed in [SLMJ07, ZC06, ZWL07] and described in 4.1. For these techniques effort has to be made in order to determine if they are compatible with SNEE and how difficult the integration will be.

To conclude this section, regarding the implementation of in-network data analysis techniques, in order to implement the D3 algorithm, the following components are chosen to be implemented:

- Chain Sample, which maintains a running sample of the sensor readings in a sliding window;
- Reservoir Sample, which maintains a sample of fixed size;
- Variance estimator, which maintains a running estimate of the standard deviation of sensor readings in a sliding window;
- Kernel density estimator, which is used to approximate the data distributions, and also contains the required machinery for using these approximations to answer queries.

The algorithms presented in Chapter 4 regarding in-network and centralised outlier detection and sampling, will be implemented in order to operate in a centralised manner as well, over a database with data consisting of sensor readings. This is required in the flood warning use case, in which the sensor network is viewed as a resource for which there is no in-network processing capability. Thus, the ability to efficiently employ our centralised techniques is of great importance and useful.



Approach. This task involves:

- Implementing Sensor network versions of the outlier detection operators (e.g. in nesC);
- Implementing Robust network versions of the outlier detection operators (e.g. in Java);
- Doing the above for other data analysis operations, (e.g. sampling);
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- None.

6.2 Enabling Unified Query Processing over Sensor Network and Robust Networks

The SemSorGrid4Env project demonstrators will source data from two types of sensor networks. In the fire use-case, in-network query processing will be supported, and the evaluation of SNEE queries as well as data analysis will take place within the sensor network over an acquisitional stream. In the flood use-case, the sensor nodes have a predetermined functionality, and the sensor network therefore emits an event stream (from the point of view of the query processor which will receive these tuples). Currently, the SNEE optimiser currently only supports the former, as the query plan comprises generated code executables for sensor network nodes. In order to process the streaming data from the flood use-case, query evaluation and data analysis will need to take place on robust network nodes (i.e. at servers between the sensor network sources and the application tier, as described in Chapter 5).

The SNEE optimiser will therefore be extended to process event streams in the robust network, as well as acquisitional streams within sensor networks that support in-network query processing. This will involve the design and implementation of an event stream evaluation engine, in order to interpret query plans expressed (rather than executable code for the in-network case), on relatively well-resourced robust network nodes. This is already underway as part of an MSc project at UNIMAN. Also, in the case of sensor networks that support in-network processing, optimisation policies will have to be implemented to allocate fragments to the sensor network or the robust network nodes as appropriate. For example, certain resource intensive fragments may not be able to be evaluated in the sensor network, whereas in other cases, evaluating a fragment within the sensor network may be advantageous if it reduces the amount of communication sent via the radio. This implies that where-scheduling decisions will need to be able to contend with these trade-offs.

Approach. This task involves:

- Designing and implementing query language syntax to reference event streams and the capability to register event streams in the SNEE schema metadata;
- Designing and implementing a query evaluation engine for executing query plan fragments over event streams on robust network nodes;
- Designing and implementing Optimisation policies for scheduling query plan fragments to robust network nodes and sensor network nodes as appropriate;
- Evaluating the above design and implementation;
- Writing up the results obtained.



Dependencies. This task has the following dependencies:

- None.

6.3 Enabling Unified Query Processing over Acquisitional Streams, Event Streams and Relations

The SNEEq_l query language can contend with acquisitional streams, event streams, and also relations. The semantics of the SNEEq_l query language defined in [BGF08] cater for that. However, support for the latter two is currently not implemented in the SNEE optimiser or evaluator. Support for stored relations is motivated by the need to query historical data sources for both demonstrators in the SemSorGrid4Env project. For example, it may be useful to correlate a stream describing an event which is currently unfolding with past data in order to identify any similarities or trends.

Relations may be located within sensor networks (e.g. stored on persistent flash memory of sensor nodes), or at external data sources (e.g. which expose a WS-DAIR service-based interface). SNEE may therefore either use OGSA-DAI and/or JDBC for scanning and materialising relations located on external, stored data sources.

Approach. This task involves designing and implementing:

- Query language syntax to create and reference relations the capability to register relations in the SNEE schema metadata;
- Operators for scanning from, and materialising to, relations;
- Optimisation policies for scheduling query plan fragments containing these operators to robust network nodes and sensor network nodes, as appropriate.
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- *Enabling Unified Query Processing over Sensor Network and Robust Networks* (Section 6.2).

6.4 Enabling Views over Streaming and Stored Data in Both Sensor and Robust Networks

A view is a named query, whose result may be a relation or stream. Views will be registered into the schema metadata, and can be incorporated into subsequent queries.



Approach. This task involves:

- Designing and implementing a data definition language (DDL) syntax for defining views, query language syntax for making use of them, and the capability to register views in the SNEE schema metadata;
- Designing and implementing extensions to the optimiser to make use of views;
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- *Enabling Unified Query Processing over Sensor Network and Robust Networks* (Section 6.2);
- *Enabling Unified Query Processing over Acquisitional Streams, Event Streams and Relations* (Section 6.3).

6.5 Enabling QoS-aware Query Processing in Sensor and Robust Networks

Sensor Network Query processors developed to date, e.g. TinyDB [MFHH05] and the current version of SNEE, are concerned with a single optimisation goal, viz., maximising sensor network lifetime. This is a consequence of the resource constrained nature of sensor network hardware, with regards to memory, processing power, communication capability and, in particular, energy availability. These query processors (and many other sensor network data management techniques proposed in the literature) trade-off processing and energy consumption, and operate in duty-cycles whereby nodes are asleep for as long as possible and only perform computations and communications in short bursts of activity. Their aim is to be as economical as possible with energy consumption, so that the need to replace batteries in sensor nodes (an expensive, and sometimes impossible task) can be avoided for as long as possible.

Note that maximising sensor network lifetime at the expense of other QoS desiderata such as acquisition interval or total energy may not always be desirable. For example, in the fire-use case scenario, if a fire is detected, delivery time will be the main QoS metric of concern, not sensor network lifetime (e.g. consider query $Q3$ in Figure 4.13). Conversely, while performing routine monitoring of the forest humidity level, e.g. to obtain a danger rating index, as shown in $Q2$ in Figure 4.13, delivery time is likely to be less of a concern, and prolonging network lifetime should be the main QoS of concern. The current version of the SNEE optimiser has only limited QoS-awareness (it is focused on lifetime, although the user may specify an upper bound on the delivery time). Work is almost completed on developing a QoS-aware version over acquisitional streams so that diverse optimisation goals and constraints expressed in terms of QoS metrics will be reflected in the query plans generated. The QoS metrics being considered are:

- The frequency that sensors read data (the *acquisition interval*);
- The maximum delay tolerable for data to reach the user (the *delivery time*);
- The *total energy consumption* of the nodes in the sensor network; and
- The time taken for the sensor network to become non-operational due to energy depletion (the *network lifetime*).

For event streams, the issue of QoS still needs to be considered.



Approach. This task involves:

- Identifying different query planning decision-making behaviours for different QoS goals for acquisitional streams over sensor networks (complete);
- Designing and implementing QoS-aware algorithms for the *multi-site* steps in the SNEE query stack for acquisitional streams over sensor networks (complete);
- Doing the above for event streams over robust networks;
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- None.

6.6 Enabling Multiple Query Execution

Currently, SNEE is only able to compile a single query at a time. However, there are many cases when more than one query may be executing simultaneously. For example, a continuous model may be incrementally updated at the same time as another query is using the model to check for outliers, e.g. if $M1$ and $Q4$ from Figure 6.1 are executing simultaneously. Preliminary work in this area has been done [Naj08]. This will be extended and incorporated into SNEE.

Approach. This task involves:

- Identifying query plan sharing opportunities;
- Designing and implementing algorithms for merging different query plan agendas;
- Extending the entire optimiser for making use of multiple queries;
- Designing and implementing mechanism for adding a query to an existing query workload;
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- None.

6.7 Integrating Data Analysis Techniques into Continuous Queries

The advantages of integrating query processing techniques with data analysis algorithms are twofold. Firstly, enabling users to include data analysis operations seamlessly within query language statements will provide a convenient means for users to express data retrieval and analysis requests in a declarative manner using the same language. Furthermore, if data analysis functionality is bundled up as operators in the physical algebra, decisions relating to their placement and timing can be delegated to the query optimiser. Thus the query optimiser will be able to decide at which (sensor network or robust network) nodes to carry out data analysis.

A proposal for a syntax for incorporating data analysis functionality into SNEEqI has already been sketched, and some examples are shown in Figure 6.1. Since the outlier detection algorithms that will be implemented require a model describing the data, two statements, $M1$ and $M2$, are presented which would cause SNEE to build a model. $M1$ is a continuously updated model and will be used to identify extreme sensor readings that may indicate the existence of an event such as a fire. The predefined `treeData` view (statement $V1$, which assumes that view definition capabilities as described in Section 6.4 have been implemented) is used to do this. This results in the definition of $M1$, the `treeDataModel`, which defines a kernel-density-estimator-based model using the view `treeData`. The model is continuously updated, as it has been created using a SNEEqI view whose output is a stream. Note that `REL_ERROR` is the maximum relative error we wish to tolerate in the estimation and `F` is the probability that a tuple will be included in the sample. These correspond to the ϵ and f parameters described in Section 4.1.2. These parameters would lead to different requirements in terms of the resources used, and the accuracy exhibited by the model and would be determined empirically.

Statement $M2$ creates a model, based on historical data, from the relation defined by statement $R1$, which collects temperature and humidity readings over the month of July. (This assumes that support for relations, as described in Section 6.2, has been implemented.) $M2$ defines a historical histogram model using a SNEEqI relation. This model is created only once, since the corresponding SNEEqI query is evaluated for a fixed period of time and since the result is stored in the relation. Note that `COUNT_N` is the sampling neighbourhood and `SAMPLE_N` is the counting neighbourhood, and correspond to the r and αr parameters defined in Section 4.1.2.

Statement $Q4$ is a query posed to obtain data from the tree stream, with a predicate that will result in answers being delivered to the user if they deviate from the model, i.e. are outliers, thereby indicating the possible presence of a fire. Note that as the model building and query are executing at the same time, this assumes that support for multiple query execution has been implemented (Section 6.6). $Q4$ continuously displays tuples which are outliers according to continuous model `treeDataModel` when the temperature attribute value is greater than 30. Note that D and r are parameters of the `isOutlier` function.

A problem with $Q4$ is that it may not return any results, and the user may be unsure whether this indicates that there is no fire, or that there is a malfunction in the sensor network. $Q5$ addresses this by displaying all tuples produced, and shows an additional flag indicating whether the tuple is an outlier or not. $Q6$ continuously displays tuples which are not outliers (according to model `julyTreeDataModel`, which is based on historical data). $Q7$ returns tuples which are not outliers by considering the temperature attribute only, using model $M2$.

Approach. This task involves:

- Designing and implementing a syntax to incorporate data analysis techniques into SNEEqI;
- Designing and implementing algebraic operators for data analysis algorithms;



```
V1: CREATE VIEW treeData AS
      SELECT RSTREAM t.locx, t.locy, t.temperature, t.rhumidity
      FROM tree[NOW-1 HOUR TO NOW] t;

M1: CREATE KDE MODEL treeDataModel AS
      SELECT *
      FROM treeData
      WITH REL_ERROR = ? AND F = ?

R1: CREATE TABLE julyTreeDataLog AS
      SELECT t.locx, t.locy, t.temperature, t.rhumidity
      FROM tree[FROM 1 Jul 2009 00:00 GMT TO 31 Jul 2009 13:00 GMT]

M2: CREATE HISTOGRAM MODEL julyTreeDataModel AS
      SELECT *
      FROM julyTreeDataLog
      WITH COUNT_N = ? AND SAMPLE_N = ?

Q4: SELECT RSTREAM time, t.locx, t.locy, t.temperature, t.rhumidity
      FROM tree[FROM NOW] t, treeDataModel m
      WHERE m.isOutlier[D, r](t.locx, t.locy, t.temperature, t.humidity)
      AND t.temperature > 30

Q5: SELECT RSTREAM t.time, t.locx, t.locy, t.temperature, t.rhumidity,
      m.isOutlier[D, r](t.locx, t.locy, t.temperature, t.rhumidity)
      FROM tree[FROM NOW] t, treeDataModel m
      WHERE temperature > 30

Q6: SELECT RSTREAM time, t.locx, t.locy, t.temperature, t.rhumidity
      FROM tree[FROM NOW] t, julyTreeDataModel m
      WHERE NOT m.isOutlier[](t.locx, t.locy, t.temperature, t.humidity)

Q7: SELECT time, t.locx, t.locy, t.temperature, t.rhumidity
      FROM treeDataLog, julyTreeDataModel
      WHERE NOT julyTreeDataModel.isOutlier[](temperature)
```

Figure 6.1: Model definitions, and queries which use the models to detect outliers.

- Designing and implementing rules for SNEE optimiser where-scheduling decisions;
- Designing and implementing rules for SNEE optimiser when-scheduling decisions;
- Extending the code generation phase accordingly;
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- *Implementation of Data Analysis Techniques within Sensor Networks and Robust Networks* (Section 6.1);
- *Enabling Multiple Query Execution*, as in many cases the model building and query which references the model will be evaluated simultaneously (Section 6.6);
- *Enable Unified Query Processing over Sensor Network and Robust Networks* (Section 6.2), to be able to perform data analysis on event streams;
- *Enabling Unified Query Processing over Acquisitional Streams, Event Streams and Relations* (Section 6.3), to be able to perform data analysis over historical data sources.

6.8 Design and Implement Sensor Network Management Capabilities to Support the Above

The current version of SNEE has, to date, only been evaluated using simulators. In these simulators, certain capabilities, broadly referred to as network management functionality, are not a concern as it comes for free, or is provided by the programmer as a configuration parameter to the simulator. Consequently, in order to deploy SNEE in the fire use-case sensor network, several aspects of network management functionality will need to be incorporated into SNEE. Examples of this type of functionality include:

- Network formation and topology maintenance/self-healing. By self-healing it is meant that if a node dies in the network, messages will be routed automatically via an alternative route. Similarly, if a node dies, the fragments of the query plan it is executing need to be taken over by an alternative node in a timely manner.
- Obtaining a network connectivity graph from the sensor network (needed by SNEE to devise a routing tree).
- Synchronisation between nodes in the network. By synchronisation it is meant that the clocks on each node should have roughly the same time, so that sensor readings and radio communications can take place in a co-ordinated manner. For SNEE, this does not need to be very accurate—10ms will suffice.
- A mechanism for query plan dissemination. The SNEE query optimiser generates a different executable code image for each node, which needs to be sent across the airwaves to the corresponding node.
- End-to-end reliable message transmission. This may be required because, in certain situations, e.g. if a fire is detected, it is worthwhile to expend additional energy to ensure that query results are received by end-users.
- A mechanism for sending commands or messages to a subset of sensor network nodes.

We are working actively in collaboration with WP6 to address these issues.



Approach. This task involves:

- Identifying existing or creating from scratch software that can provide the network management functionality required by SNEE;
- Designing and implementing changes to the SNEE optimiser which result from (limitations of) the network management functionality. For example, certain routing and when-scheduling made by the SNEE optimiser may be delegated to the network management functionality if appropriate;
- In liaison with WP6, incorporating this functionality into the SNEE optimiser;
- Evaluating the above design and implementation;
- Writing up the results obtained.

Dependencies. This task has the following dependencies:

- None.

6.9 Summary

This chapter has described the work to be carried out by WP2 for the remainder of the project, in order to meet the requirements which arise from other work packages in the SemSorGrid4Env project. Some of these tasks will be done in collaboration with other work packages. For example, implementing the network management functionality will involve collaboration with WP6, and generalising the SNEE query processor to support the event streams and query evaluation over robust networks will involve collaboration with WP4. Note that the functional decomposition of tasks proposed in this chapter is around functional components within the WP2 architecture, which is to an extent orthogonal to the phase-release model described in in the technical annex. These tasks will be scheduled to ensure timely production of the forthcoming deliverables.



Bibliography

- [AAB⁺05a] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryzkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *Proceedings of 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Asilomar, CA, USA, January 2005.
- [AAB⁺05b] Mario Antonioletti, Malcolm P. Atkinson, Robert M. Baxter, Andrew Borley, Neil P. Chue Hong, Brian Collins, Neil Hardman, Alastair C. Hume, Alan Knox, Mike Jackson, Amrey Krause, Simon Laws, James Magowan, Norman W. Paton, Dave Pearson, Tom Sugden, Paul Watson, and Martin Westhead. The design and implementation of grid database services in OGSA-DAI. *Concurrency - Practice and Experience*, 17(2-4):357–376, February/April 2005.
- [AAK⁺06] M. Antonioletti, Malcolm Atkinson, A. Krause, S. Laws, S. Malaika, Norman W. Paton, D. Pearson, and G. Riccardi. Web services data access and integration - the core (WS-DAI) specification, version 1.0. Recommendation GFD.74, Open Grid Forum, 5 September 2006. <http://www.ogf.org/documents/GFD.74.pdf>.
- [ABB⁺03] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom. Stream: The stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.
- [ABB⁺09] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. In Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors, *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2009.
- [ABD⁺89] Malcolm P. Atkinson, François Bancilhon, David J. DeWitt, Klaus R. Dittrich, David Maier, and Stanley B. Zdonik. The object-oriented database system manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89)*, pages 223–240, Kyoto, Japan, December 1989. Elsevier.
- [ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal — The International Journal on Very Large Data Bases*, 15(2):121–142, June 2006.
- [ACQ⁺03] Daniel J. Abadi, Donald Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, November 2003.
- [ACK⁺06] Mario Antonioletti, Brian Collins, Amy Krause, Simon Laws, James Magowan, Susan Malaika, and Norman W. Paton. Web services data access and integration - the relational realisation (WS-DAIR) specification, version 1.0. Recommendation GFD.76, Open Grid Forum, 20 July 2006. <http://www.ogf.org/documents/GFD.76.pdf>.
- [ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer, 2004.
- [Agg07] Charu C. Aggarwal, editor. *Data Streams: Models and Algorithms*. Advances in Database Systems. Springer, 2007.
- [AGGR99] R. Agrawal, J.E. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications, December 14 1999. US Patent 6,003,029.



- [AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272, Dallas, TX, USA, May 2000. ACM.
- [AHK⁺06] Mario Antonioletti, Shannon Hastings, Amy Krause, Stephen Langella, Steven Lynden, Simon Laws, Susan Malaika, and Norman W. Paton. Web services data access and integration – the XML realization (WS-DAIX) specification, version 1.0. Recommendation GFD.75, Open Grid Forum, 2 August 2006. <http://www.ogf.org/documents/GFD.75.pdf>.
- [AJA04] Nabil R. Adam, Vandana Pursnani Janeja, and Vijayalakshmi Atluri. Neighborhood based detection of anomalies in high dimensional spatio-temporal sensor datasets. In *SAC*, pages 576–583, 2004.
- [Alp04] E. Alpaydin. *Introduction to machine learning*. The MIT Press, 2004.
- [Ban06] Tim Banks. Web services resource framework (WSRF) – primer. Standard Specification 1.2, OASIS, 23 May 2006.
- [BBC⁺07] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML path language (XPath) 2.0. Recommendation, W3C, January 2007.
- [BBC⁺09] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, pages 1061–1062, Madrid, Spain, April 2009. ACM.
- [BBD⁺02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002)*, pages 1–16, Madison, WI, USA, June 2002. ACM.
- [BC07] Christian Bizer and Richard Cyganiak. D2RQ – lessons learned. In *W3C Workshop on RDF Access to Relational Databases*, Cambridge, MA, USA), October 2007. <http://www.w3.org/2007/03/RdfRDB/>.
- [BCF⁺07] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language. Recommendation, W3C, January 2007.
- [BDM02] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002.
- [BGFP08] Christian Y. A. Brenninkmeijer, Ixent Galpin, Alvaro A. A. Fernandes, and Norman W. Paton. A semantics for a query language over sensors, streams and relations. In *BNCOD*, pages 87–99, 2008.
- [BGFP09] Christian Y. A. Brenninkmeijer, Ixent Galpin, Alvaro A. A. Fernandes, and Norman W. Paton. Validated cost models for sensor network queries. In *DMSN*, 2009. To Appear.
- [BGJ08] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming SPARQL – Extending SPARQL to process data streams. In *Proceedings of 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *LNCS*, pages 448–462, Tenerife, Spain, June 2008. Springer.
- [BGS00] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Querying the Physical World. *IEEE Personal Communications*, 7:10–15, October 2000.
- [BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *SIGMOD Conference*, pages 93–104, 2000.



- [BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *Proceedings of First International Semantic Web Conference (ISWC)*, volume 2342 of *LNCS*, pages 54–68, Sardinia, Italy, June 2002. Springer.
- [BL94] V. Barnett and T. Lewis. *Outliers in statistical data*. Wiley New York, 1994.
- [BS03] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.
- [BSG⁺06] Joel W. Branch, Boleslaw K. Szymanski, Chris Giannella, Ran Wolff, and Hillol Kargupta. In-network outlier detection in wireless sensor networks. In *ICDCS*, page 51, 2006.
- [BW01] S. Babu and J. Widom. Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120, 2001.
- [CBB⁺00] R. G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez, editors. *The Object Data Management Standard: ODMG 3.0*. Morgan Kaufmann, 2000.
- [CC09] Oscar Corcho and Jean-Paul Calbimonte. Design of the SemSorGrid4Env ontology-based data integration model. Deliverable D4.1, SemSorGrid4Env, August 2009.
- [CG06] Graham Cormode and Minos N. Garofalakis. Streaming in a connected world (tutorial). In *Proceedings of the 32nd International Conference on Very Large Data Bases*, page 1266, Seoul, Korea, September 2006. ACM.
- [CGL⁺09] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: The DL-Lite approach. In *Proceedings of 5th International Reasoning Web Summer School (RW 2009)*, LNCS. Springer, August 2009. To appear.
- [CGN05] Andrew W. Cooke, Alasdair J. G. Gray, and Werner Nutt. Stream integration techniques for grid monitoring. *Journal on Data Semantics*, 2:136–175, 2005.
- [CGRS04] K. Chakrabarti, M.N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets, July 6 2004. US Patent 6,760,724.
- [CHSR09] Mike Clark, Craig Hutton, Jason Sadler, and Samantha Roe. Wp 7.1 flood user requirements specification. Deliverable D7.1, SemSorGrid4Env, March 2009.
- [CHYC96] M.S. Chen, J. Hun, P.S. Yu, and W.R. Ctr. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and data Engineering*, 1996.
- [CL03] J.H. Chang and W.S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 487–492. ACM New York, NY, USA, 2003.
- [CLKB04] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)*, pages 449–460, Boston, MA, USA, 2004. IEEE Computer Society.
- [CMZ07] G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *Intl. Conf. on Data Engineering*, 2007.
- [Cod70] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM - CACM ACM*, 13(6):377–387, June 1970.
- [DF03] Murat Demirbas and Hakan Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Peer-to-Peer Computing*, pages 32–39, 2003.



- [DH05] AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.
- [DLK08] A.J. Dou, S. Lin, and V. Kalogeraki. Real-time querying of historical data in flash-equipped sensor devices. In *RTSS*, 2008.
- [EKX95] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. A database interface for clustering in large spatial databases. In *KDD*, pages 94–99, 1995.
- [ES07] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, first edition, 2007. <http://www.springer.com/978-3-540-49611-3>.
- [eXi] eXist: Open source native XML database. <http://exist.sourceforge.net/> accessed 15 July 2009.
- [Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, Information and Computer Science, University of California, Irvine, California, USA, 2000.
- [FMSS07] Mary F. Fernández, Philippe Michiels, Jérôme Siméon, and Michael Stark. XQuery streaming à la carte. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 256–265, Istanbul, Turkey, April 2007. IEEE Computer Society.
- [FW04] David C. Fallside and Priscilla Walmsley (Eds). XML schema part 0: Primer second edition. Recommendation, W3C, 28 October 2004. <http://www.w3.org/TR/xmlschema-0/>.
- [GBJ⁺09] Ixent Galpin, Christian Y. A. Brenninkmeijer, Farhana Jabeen, Alvaro A. A. Fernandes, and Norman W. Paton. Comprehensive optimization of sensor network queries. In *SSDBM*, pages 339–360, 2009.
- [GGF⁺09] Alasdair J. G. Gray, Ixent Galpin, Alvaro A. A. Fernandes, Kevin Page, Jason Sadler, Manolis Koubarakis, Kostis Kyzirakos, Oscar Corcho, Raúl Garcia, José Mora, Víctor Manuel Díaz, and Israel Liébana. SemSorGrid4Env architecture – phase I. Deliverable D1.3v1, SemSorGrid4Env, August 2009.
- [GGFP09] Alasdair J. G. Gray, Ixent Galpin, Alvaro A. A. Fernandes, and Norman W. Paton. Web services data access and integration - the data stream realisation (WS-DAI-Streaming) specification. Technical report, University of Manchester, August 2009.
- [GGP08] Miguel Esteban Gutiérrez and Asunción Gómez-Pérez. Web services data access and integration – the RDF(S) realization (WS-DAI-RDF(S)) ontology specification. Working Draft Version 0.8, Open Grid Forum, 22 February 2008.
- [GGR09] Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data Stream Management*. Data-Centric Systems and Applications. Springer, February 2009.
- [GHM06] Steve Graham, David Hull, and Bryan Murray (Eds). Web services base notification 1.3 (WS-BaseNotification). Standard, OASIS, 1 October 2006. <http://docs.oasis-open.org/wsn/>.
- [GK01] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 58–66. ACM New York, NY, USA, 2001.
- [GLvB⁺03] David Gay, Philip Levis, J. Robert von Behren, Matt Welsh, Eric A. Brewer, and David E. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI*, pages 1–11, 2003.
- [GM98] P.B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. *ACM SIGMOD Record*, 27(2):331–342, 1998.



- [GM04] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.
- [GMWU99] Hector Garcia-Molina, Jennifer Widom, and Jeffrey D. Ullman. *Database System Implementation*. Prentice-Hall Incorporated, Upper Saddle River, NJ, USA, 1999.
- [GNW07] Alasdair J. G. Gray, Werner Nutt, and M. Howard Williams. Answering queries over incomplete data stream histories. *International Journal of Web Information Systems*, 3(1/2):41–60, 2007.
- [GÖ03] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, June 2003.
- [GPO06] Amol Ghoting, Srinivasan Parthasarathy, and Matthew Eric Otey. Fast mining of distance-based outliers in high dimensional datasets. In *SDM*, 2006.
- [Gra90] Goetz Graefe. Encapsulation of parallelism in the volcano query processing system. In *SIGMOD Conference*, pages 102–111, 1990.
- [Gra07] Alasdair J. G. Gray. *Integrating Distributed Data Streams*. PhD thesis, Heriot-Watt University, Edinburgh, UK, November 2007.
- [GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD Conference*, pages 73–84, 1998.
- [GT06] Steve Graham and Jem Treadwell. Web services resource properties 1.2 (WS-ResourceProperties). Standard, OASIS, 1 April 2006.
- [Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, December 2001.
- [Haw80] D. Hawkins. *Identification of Outliers*. Chapman and Hall, 1980.
- [HK98] A. Hinneburg and D.A. Keim. An efficient approach to clustering in large multimedia databases with noise. *Knowledge Discovery and Data Mining*, 5865, 1998.
- [HK06] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.
- [HRO06] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In *Proceedings of 32nd International Conference on Very Large Data Bases (VLDB)*, pages 9–16, Seoul, Korea, September 2006. ACM.
- [HS08] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. *Computational Geometry: Theory and Applications*, 39(3):191–208, 2008.
- [HSW⁺00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *ASPLOS*, pages 93–104, 2000.
- [HV05] M. Halkidi and M. Vazirgiannis. *Data Mining and Knowledge Discovery in Databases and Web (in Greek)*. Tipothito, 2005.
- [Ioa03] Y. Ioannidis. The history of histograms (abridged). In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 19–30. VLDB Endowment, 2003.
- [JC04] Ankur Jain and Edward Y. Chang. Adaptive sampling for sensor networks. In *DMSN*, pages 10–16, 2004.
- [JK98] A. Joshi and R. Krishnapuram. Robust fuzzy clustering methods to support web mining. In *Proc. Workshop in Data Mining and knowledge Discovery, SIGMOD*, pages 15–1, 1998.



- [JMF99] AK Jain, MN Murty, and PJ Flynn. Data clustering: a review. *ACM computing surveys*, 31(3), 1999.
- [JMR05] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Sampling algorithms in a stream operator. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 1–12. ACM New York, NY, USA, 2005.
- [KCC⁺03] Sailesh Krishnamurthy, Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Samuel R. Madden, Vijayshankar Raman, Fred Reiss, and Mehul A. Shah. TelegraphCQ: An architectural status report. *IEEE Data Engineering Bulletin*, 26(1):11–18, March 2003.
- [KKK09] Kostis Kyzirakos, Manolis Koubarakis, and Zoi Kaoudi. Data models and languages for registries in SensorGrid4Env. Deliverable D3.1 Version 1.0, SemSorGrid4Env, August 2009.
- [KMB05] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. *Lecture Notes in Computer Science*, 3633:364, 2005.
- [KN98] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.
- [Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, December 2000.
- [KR90] L. Kaufman and P.J. Rousseeuw. Finding groups in data: an introduction to cluster analysis. *New York*, 1990.
- [KW05] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley, June 2005.
- [LAGD08] Song Lin, Benjamin Arai, Dimitrios Gunopulos, and Gautam Das. Region sampling: Continuous adaptive sampling on sensor networks. In *ICDE*, pages 794–803, 2008.
- [LBW07] Andreas Langegger, Martin Blöchl, and Wolfram Wöß. Sharing data on the grid using ontologies and distributed SPARQL queries. In *Proceedings of 18th International Workshop on Database and Expert Systems Applications (DEXA 2007)*, pages 450–454, Regensburg, Germany, September 2007. IEEE Computer Society.
- [LDI09] Israel Liébana, Víctor Manuel Díaz, and Agustín Izquierdo. Fire risk monitoring and warning in Castilla y León - requirements specification. Deliverable D6.1, SemSorGrid4Env, March 2009.
- [Len02] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002)*, pages 233–246, Madison, WI, USA, June 2002. ACM.
- [LLWC03] Philip Levis, Nelson Lee, Matt Welsh, and David E. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys*, pages 126–137, 2003.
- [Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
- [McB02] Brian McBride. Jena: A semantic web toolkit. *IEEE Internet Computing*, 6(6):55–59, November/December 2002.
- [MFHH05] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.
- [MM04] F. Manola and E. Miller (eds). RDF primer. Recommendation, W3C, 10 February 2004. <http://www.w3.org/TR/rdf-primer/>.



- [MPC07] Jun-Ki Min, Myung-Jae Park, and Chin-Wan Chung. XTREAM: An efficient multi-query evaluation on streaming XML data. *Information Sciences*, 177(17):3519 – 3538, September 2007.
- [MRL99] G.S. Manku, S. Rajagopalan, and B.G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *ACM SIGMOD Record*, 28(2):251–262, 1999.
- [MS03] A. D. Marbini and L. E. Sacks. Adaptive sampling mechanisms in sensor networks. In *London Communications Symposium*, 2003.
- [Naj08] Jamil M. Naja. Multiple executing queries in sensor networks. Msc, School of Computer Science, University of Manchester, UK, 2008.
- [NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.
- [Noy04] Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, December 2004.
- [OGS09] OGSA-DAI open grid services architecture database access and integration services project, 2005-2009. <http://www.ogsadai.org.uk/> accessed 15 June 2009.
- [OR95] F. Olken and D. Rotem. Random sampling from databases: A survey. *Statistics and Computing*, 5(1):25–42, 1995.
- [Pe08] E. Prud’hommeaux and A. Seaborne (eds). SPARQL query language for RDF. Recommendation, W3C, 15 January 2008. <http://www.w3.org/TR/rdf-sparql-query/>.
- [PGI99] V. Poosala, V. Ganti, and Y.E. Ioannidis. Approximate query answering using histograms. *IEEE Data Engineering Bulletin*, 1999.
- [PKGF03] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, pages 315–, 2003.
- [PRMS09] Kevin R. Page, David C. De Roure, Kirk Martinez, and Jason Sadler. Specification of high-level application programming interfaces. Deliverable D5.1, SemSorGrid4Env, February 2009.
- [QL08] Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In *Proceedings of 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *LNCIS*, pages 524–538, Tenerife, Spain, June 2008. Springer.
- [Qui86] JR Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- [Qui93] J.R. Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [RCGP06] Jesús Barrasa Rodríguez, Óscar Corcho, and Asunción Gómez-Pérez. A semantic portal for fund finding in the eu: Semantic upgrade, integration and publication of heterogeneous legacy data. In *Proceedings of 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, part 3*, volume 4253 of *LNCIS*, pages 588–597, Bournemouth, UK, October 2006. Springer.
- [RKV95] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *SIGMOD Conference*, pages 71–79, 1995.
- [RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD Conference*, pages 427–438, 2000.
- [RSZ05] Cauligi S. Raghavendra, Krishna M. Sivalingam, and Taleb Znati, editors. *Wireless Sensor Networks*. Springer, 2005.



- [RWP04] Dongmei Ren, Baoying Wang, and William Perrizo. Rdf: A density-based outlier detection method using vertical data representation. In *ICDM*, pages 503–506, 2004.
- [Sco92] D. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley & Sons, 1992.
- [SFJ03] Marko Smiljanić, Ling Feng, and Willem Jonker. *Intelligent Search on XML Data: Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *LNCS*, chapter Web-Based Distributed XML Query Processing, pages 207–216. Springer, 2003.
- [SKG05] Amir Soheili, Vana Kalogeraki, and Dimitrios Gunopulos. Spatial queries in sensor networks. In *GIS*, pages 61–70, 2005.
- [SLMJ07] Bo Sheng, Qun Li, Weizhen Mao, and Wen Jin. Outlier detection in sensor networks. In *MobiHoc*, pages 219–228, 2007.
- [SMWG09] J.J. Sharplesa, R.H.D. McRaeb, R.O. Webera, and A.M. Gill. A simple index for assessing fire danger rating. *Environmental Modelling & Software*, 24(6):764–774, June 2009.
- [SMZ07] Kazem Sohraby, Daniel Minoli, and Taieb F. Znati. *Wireless sensor networks: Technology, protocols, and applications*. Wiley, 2007.
- [SPP+06] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006.
- [SQL92] Information technology – database language SQL. Standard ISO/IEC 9075:1992, ISO, 1992.
- [SS04] S. Staab and R. Studer, editors. *Handbook on Ontologies*. Springer, 2004.
- [Sun09] Sun Microsystems. Jdbc overview, 1994-2009. <http://java.sun.com/products/jdbc/overview.html> accessed 27 May 2009.
- [TcZ+03] Nesime Tatbul, Ugur Çetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.
- [TGNO92] Douglas B. Terry, David Goldberg, David A. Nichols, and Brian M. Oki. Continuous queries over append-only databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 321–330, San Diego, CA, USA, June 1992. ACM.
- [TK03] S. Theodoridis and K. Koutroumbas. *Pattern recognition*. Academic press, 2003.
- [Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.
- [WMN04] R. Willett, A. Martin, and R. Nowak. Backcasting: adaptive sampling for sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 124–133. ACM New York, NY, USA, 2004.
- [WVV+01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information — a survey of existing approaches. In *Workshop on Ontologies and Information Sharing, IJCAI 2001*, 2001.
- [Xin07] Apache Xindice, 2007. <http://xml.apache.org/xindice/> accessed 15 July 2009.
- [YG02] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.
- [YG03] Yong Yao and Johannes Gehrke. Query processing in sensor networks. In *CIDR*, 2003.



- [YG08] Jie Yin and Mohamed Medhat Gaber. Clustering distributed time series in sensor networks. In *ICDM*, pages 678–687. IEEE Computer Society, 2008.
- [ZC06] Yongzhen Zhuang and Lei Chen. In-network outlier cleaning for data collection in sensor networks. In *Proceedings of the First International VLDB Workshop on Clean Databases (CleanDB 2006)*, Seoul, Korea, 2006.
- [ZCY⁺07] Aoying Zhou, Feng Cao, Ying Yan, Chaofeng Sha, and Xiaofeng He. Distributed data stream clustering: A fast EM-based approach. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 736–745, Istanbul, Turkey, 2007.
- [ZLW07] Qi Zhang, Jinze Liu, and Wei Wang. Incremental subspace clustering over multiple data streams. In *ICDM*, pages 727–732, 2007.
- [ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD Conference*, pages 103–114, 1996.
- [ZWL07] Yongzhen Zhuang, Lei Chen 0002, Xiaoyang Sean Wang, and Jie Lian. A weighted moving average-based approach for cleaning sensor data. In *ICDCS*, page 38, 2007.