



DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING
DEGREE PROGRAMME IN INFORMATION ENGINEERING

MIDDLEWARE FOR SENSOR DATA PROCESSING

Author _____
Teemu Leppänen

Supervisor _____
Jukka Riekki

Accepted _____ / _____ 2009

Grade _____

Leppänen T. (2009) Middleware for sensor data processing. University of Oulu, Department of Electrical and Information Engineering, Oulu, Finland. Master's Thesis, 57 p., 1 appendix.

ABSTRACT

In traffic today, major issues are road safety, fuel consumption and human driving errors. Ecological traffic requires diminished congestion and emission as well as use of public transport. Intelligent traffic systems solve these issues with advancement of information and communication technologies. In future traffic, actors are context-aware and collect and share information seamlessly. Sensor networks offer means for traffic data collection, as sensors can be placed to vehicles, roads, roadsides and the surrounding infrastructure.

In this thesis, a platform for traffic based sensor data network is developed. Open source sensor network middleware and external software libraries are compared against each other. The Global Sensor Network (GSN) is selected because of amount of ready-made features and runtime dynamic changeability. In tests, GSN was found scalable and processing time per 1MB data packet to be less than 1ms. Components for data synchronization, feature extraction, classification and data formatting are developed on top of the middleware allowing free deployment of sensors, data processing algorithms and client applications to the system by using public interfaces. A separate shared memory component was considered for communication between components but experiments indicated the built-in database systems to be equally fast and these plans were discarded. An example application is developed, where middleware receives accelerometer and GPS location data from a mobile phone which is installed in a vehicle. The application presents a mapped visualization of the route with potholes located on the road. Overall, the developed system is found to work adequately for processing of the traffic data.

Key words: intelligent traffic system, sensor network, sensor data processing, middleware.

Leppänen T. (2009) *Palvelinohjelmisto anturitiedon käsittelyyn*. Oulun yliopisto, Sähkö- ja tietotekniikan osasto. Diplomityö, 57 s., 1 liite.

TIIVISTELMÄ

Nykypäivän liikenteessä merkittävimpiä ongelmia ovat turvallisuus, polttoaineen kulutus ja ajovirheet. Ympäristöystävällinen liikenne edellyttää liikennemuutosten ja päästöjen vähentämistä sekä julkisen liikenteen hyödyntämistä. Älykkäät liikennejärjestelmät ratkaisevat näitä ongelmia tietojen ja viestintäteknologian kehityksen myötä. Tulevaisuuden liikenteessä toimijat ovat kontekstintietoisia ja keräävät ja jakavat liikennetietoa saumattomasti toisilleen. Anturiverkot tarjoavat keinoja liikennetiedon keruuseen, koska antureita voidaan sijoittaa liikennevälineisiin, tiehen, teiden varsille sekä muuhun infrastruktuuriin.

Tässä työssä toteutetaan ohjelmistoalusta liikennepohjaisen anturitiedon käsittelyä varten. Avoimen lähdekoodin ohjelmistoalustoja ja ulkoisia ohjelmistokirjastoja verrataan toisiinsa ja Global Sensor Network (GSN) valitaan alustaksi valmiiden ominaisuuksien määrän sekä ajonaikaisen dynaamisen laajennettavuuden takia. Testeissä GSN todettiin helposti laajennettavaksi. Yhden megatavun kokoisen tietorakenteen käsittelyajaksi mitattiin alle 1ms. Ohjelmistoalustan päälle kehitetään komponentteja anturitiedon synkronointiin, piirteiden etsintään, luokitteluun ja tiedon formaatin muokkaamiseen. Komponenttien tulee mahdollistaa anturien, tiedonkäsittelyalgoritmien ja asiakassovellusten vapaa liittäminen tiedonkäsittelyketjuun avoimia rajapintoja noudattaen. Erillistä jaetun muistin komponenttia harkittiin, mutta testeissä sisäänrakennetut tietokantajärjestelmät osoittautuivat yhtä nopeiksi ja tästä luovuttiin. Työssä kehitetään esimerkkisovellus, joka vastaanottaa autoon asennetun matkapuhelimen tuottamaa kiihtyvyydestietoa sekä GPS-paikannustietoa. Sovellus esittää karttapohjalla kuljetun reitin ja siitä tunnistetut kuopat. Kaiken kaikkiaan toteutetun järjestelmän voidaan todeta soveltuvan hyvin liikennetiedon anturitiedon käsittelyyn.

Avainsanat: älykkäät liikennejärjestelmät, anturiverkko, anturitiedon käsittely, ohjelmistoalusta.

TABLE OF CONTENTS

ABSTRACT	
TIIVISTELMÄ	
TABLE OF CONTENTS	
PREFACE	
TERMS AND ABBREVIATIONS	
1. INTRODUCTION	8
2. SENSOR NETWORKS	11
2.1. Introduction and concepts	11
2.2. Middleware for mobile sensor network	13
2.3. Existing sensor network platforms	14
2.3.1. Global Sensor Network	15
2.3.2. IrisNet	16
2.3.3. Other sensor network platforms	17
2.4. Extensions to platforms	18
2.4.1. Multithreading libraries	18
2.4.2. Distributed shared memory	19
2.5. Comparison of existing systems	20
2.5.1. Selecting sensor network platform	21
2.5.2. Selecting shared memory implementation	23
2.6. Feasibility of selected systems	24
2.6.1. Global Sensor Network	24
2.6.2. Object Spaces	25
3. DESIGN	26
3.1. General design issues	26
3.2. General requirements	26
3.3. Operations for sensor data	27
3.4. Data types and structures	29
3.5. Interfaces to external services, sensors and applications	30
3.6. Server architecture	31
4. IMPLEMENTATION	33
4.1. Server system and environment	33
4.2. Prototype application	34
4.3. Global Sensor Network modules	36
4.3.1. Wrappers	36
4.3.2. Virtual sensors	36
4.3.3. Configuration	39
5. TESTING AND RESULTS	40
5.1. Testing environment and cases	40
5.2. Performance tests	40
5.3. Scalability tests	41
5.4. Tests with real system data	41
5.5. Criteria for successful solution	43
6. DISCUSSION	45
6.1. Analysis of system	45
6.2. Analysis of server implementation	46

6.3.	Analysis of tests.....	48
6.4.	Future work	50
7.	CONCLUSION	52
8.	REFERENCES.....	53
9.	APPENDICES.....	57

PREFACE

This work was funded by the National Technology Agency of Finland as a part of the Cooperative Traffic research program of the Strategic Centre for Science, Technology and Innovation in the Field of ICT. The author would like to thank all the personnel in Sensor Data Fusion and Applications project.

Furthermore, I want to thank Prof. Jukka Riekkı for excellent guidance through the development and writing of this thesis. Prof. Janne Haverinen deserves thanks for the second reviewing of thesis. I want to thank M.Sc. Mikko Perttunen for the conversations during everyday work at the lab and giving advice on this thesis. I wish to express my gratitude to my friend, Dr.Tech. Susanna Pirttikangas, for introducing me this subject for thesis work. My friend M.Sc. Marko Pyhähuhta also deserves thanks for the language revision.

Last but not least, I would like to thank my family for support and encouragement to wrap up my studies during these years. Finally!

Oulu, October 26, 2009

Teemu Leppänen

TERMS AND ABBREVIATIONS

3G	Third Generation. A group of standards for mobile telecommunication.
GPRS	General Packet Radio Service. A mobile data service.
GPS	Global Positioning System. Navigation system based on satellites.
GPX	GPS Exchange Format. A GPS data description.
GSN	Global Sensor Network. Implementation for sensor network middleware.
HSQldb	Hyper Structured Query Language Database. A database system.
HTTP	Hypertext Transfer Protocol. Protocol for retrieving hypertext documents.
I/O	Input/Output. Interfaces between information processing system and outside world.
JDBC	Java Database Connectivity. Java library for accessing databases.
JVM	Java Virtual Machine. Software that executes programs like a physical machine.
PHP	Hypertext Preprocessor. Scripting language for Internet.
RFID	Radio Frequency Identification. Objects used for identification purposes.
SDFA	Sensor Data Fusion and Applications. A research project in Finnish Cooperative Traffic research program.
SQL	Structured Query Language. Database computer language.
UDP	User Datagram Protocol. Transmission protocol of data packets in Internet.
URL	Uniform Resource Locator. Specifies location and availability of resources in domain.
USB	Universal Serial Bus. Communication technology between computer and peripheral device.
WiMAX	Worldwide Interoperability for Microwave Access. A telecommunication technology.
WLAN	Wireless Local Area Network. Wirelessly connected computer network.
XML	Extensible Markup Language. Set of rules for description of data.
XML-RPC	A remote procedure call network protocol using XML to encode its calls.

1. INTRODUCTION

Traffic today has several major issues requiring advancement. Road safety and fuel consumption are the most important issues reported by drivers. Human driving errors contribute to road safety: handling of the vehicle, observation and anticipation errors, poor positioning and even falling asleep during driving. Ecological and sustainable transportation requires less fuel consumption, less congestion, less emission and promotion of various modes of transportation. Road infrastructure improvement requires traffic safety, vehicle route prediction, fast transportation, traffic flow management and efficient planning of transportation. [1]

Intelligent transportation systems are aimed at improving these issues by applying advances in information technology, communications and sensor technology to solve problems on traffic system level. Cooperative traffic is a combination of ubiquitous services to exchange information between intelligent vehicles, road, and transport infrastructure. An intelligent infrastructure matches vehicles with external systems, communication links and back-office systems. It offers context sensing methods and context-aware information to all transport entities; for example weather and road conditions, traffic disorders, emergency and breakdown services, route planning, road maintenance and travel information. Users can ask for personalized information and it can be separated to different groups such as private, commercial and professional uses. Wireless communications are a natural way of communication within an intelligent transportation system: vehicle-to-vehicle communications can use dedicated short-range communications standards and protocols, personal mobile radio or Wi-Fi/WiMAX local broadband network. In addition to the above-mentioned characteristics, vehicle-to-infrastructure communications include for example mobile communications such as GPRS or 3G. Typical infrastructure sensors are video cameras, posts or traffic signs, inductive loops detecting passing vehicles and RFID tags embedded on the road or buildings. A sketch of an intelligent transport system can be seen in Figure 1. [1]

Intelligent vehicles have extensions to present-day vehicles: there is an in-vehicle embedded system for data-mining from a variety of in-vehicle sensors; the vehicle is capable of communicating with internal and external systems and there is a human-machine-interface for driver assistance, based on collected information and making driving not only more comfortable and efficient, but also less risky and more productive in travel time. Driver assistance topics consist of aiding in navigation, foresighted driving and enhancing vision, object and hazard detection, driver behavior analysis, collision avoidance, adaptive cruise and speed control systems and travel information. In-vehicle sensors can be electrical, optical or mechanical, for example video cameras, laser scanners, GPS receivers, accelerometers, gyros or weather sensors. Sensors collect information of road and weather conditions, vehicle location and relative position to other traffic, driver behavior and driving style, and of special conditions like night time, rain or ice. An important topic is aggregation and fusion of information from multiple sensors combined with information from other sources. In-vehicle communication systems can be based on nomadic devices, for example mobile phones with Bluetooth, or Wi-Fi/WiMAX local broadband or digital video broadcasting. [1]

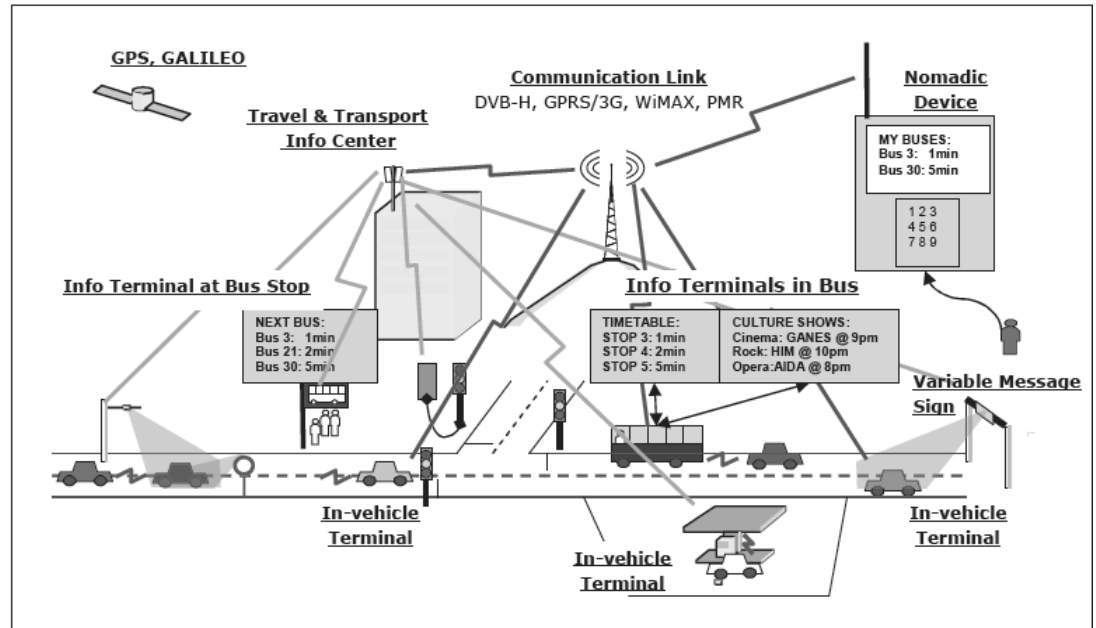


Figure 1. Sketch of an intelligent transport system.

Sensor data fusion and applications (SDFA) project is part of a Finnish cooperative traffic information and communications technologies research program. The research program is focused on monitoring road and weather conditions, for example perception of friction, and driver behavior analysis. Also communication methods for information distribution among vehicles, roadside infrastructure and other entities in traffic are developed. The SDFA project is focused on developing solutions for collecting and integrating sensor data in the context of cooperative traffic. The project's tasks are to analyze requirements and produce a state-of-the-art review while building scenarios, collecting in-vehicle and roadside infrastructure sensor data, developing methods for sensor data fusion and a toolbox of generic solutions for client applications, developing specifications and architecture for in-vehicle data collection platform, and building prototypes and demonstrations. One project participant will prepare an instrumented vehicle to be used as a data collection platform. The first-year goal of the project was to create a prototype system for collecting data sets of road conditions and for recognizing events and patterns in the data. Data sets were collected driving around the cities of Oulu and Tampere using an accelerometer sensor combined with mobile phone GPS location data. Sensor data filtering, preprocessing, data synchronization and fusion will be studied in the project. Road anomalies like potholes were selected as the first object to be recognized from the data. A separate client application was developed for visualizing the route and the located anomalies in a web browser map. [2]

In this master's thesis, a middleware for sensor data processing in the context of cooperative traffic is designed, implemented and tested. The middleware was developed for a mobile application but is also suitable as general sensor data processing middleware. Existing sensor network platforms and relevant extension software libraries are studied, tested and compared against each other. After a platform is selected, it is customized and developed to act as a middleware for fulfilling SDFA project's requirements. The middleware works as an offline and online data processing platform both for previously collected sensor data sets and for

streamed real-time sensor data. The processed data is combined with geographical data and an interface to a visualization application is developed so that the results can be visualized in a web browser. The middleware implementation is then tested and analyzed against the requirements, and conclusions for future work are drawn.

The rest of this thesis is organized as follows. In chapter 2, introductions to sensor networks and sensor network middleware architectures are given. Then existing implementations and software extensions are presented and compared. Based on the comparisons, a platform is selected to be used in the SDFA project. In chapter 3, design issues and requirements for sensor network middleware are described in general and in the context of the SDFA project. Then a prototype server design is presented. The system implementation based on this design is presented in chapter 4. In chapter 5, testing of the selected middleware platform and the prototype system as well as the test results are presented. In chapter 6, the prototype system and the tests are analyzed, and some directions for future work are given. Conclusions for the work are drawn in chapter 7.

2. SENSOR NETWORKS

In this chapter, an introduction to sensor networks is given, research topics in the field are described, and selected middleware architectures for mobile sensor applications are presented and compared. Then, selected existing sensor network platforms are presented and compared with each other. Publicly available well-known software extension libraries for the use with sensor network platforms are also presented and compared. From the comparisons, a basis for a sensor network platform for the SDFa project is presented.

2.1. Introduction and concepts

A sensor is a device that observes physical phenomenon and converts it to a signal that can be utilized by observers. Typical sensor types are acoustic, chemical, electric, optical, thermal, inertial, pressure, flow, ionizing radiation, position, automotive and environmental [3, 4]. Sensor nodes are connected to one or more physical sensors and have a data processing unit with memory, a transceiver unit, a power unit and a locating unit [3, 5]. Technically the nodes can be characterized in several ways which also affect their design. Low power consumption and small physical size are important factors in hardware design [3, 5]. Often sensor node hardware components are cheap publicly-available off-the-shelf components. Since nodes are largely unattended, reliability and robust operation are important factors; therefore the hardware and software must be able to cope with minor device failures [3, 5]. The capabilities of hardware components are limited, and thus the software must make efficient use of memory and processor cycles by utilizing a high level of concurrency [3, 5]. The diversity of applications and hardware configurations require modularity from software components [5, 6]. TinyOS is an example operating system for sensor nodes taking advantage of these guidelines [5]. It consists of a task scheduler and components. Components declare their interfaces and this allows the system to create a hierarchy of components, where higher level components issue commands to the lower level components which in turn respond by signaling events. A component consists of four parts: command handlers, event handlers and tasks which operate in the context of the frame. The frame is a statistically allocated fixed-size memory chunk, the memory consumption of which is known at the compile time. This avoids dynamic memory allocation overhead and variable locations are static, which means that execution time is saved. Tasks are atomic and run to completion, so only a single stack is needed, which is important in a concurrent system if memory is limited. The task scheduler is a power-aware first-in-first-out scheduler, which puts the processor to sleep if the queue is empty.

The sensor network consists of nodes which collaborate to form a sensing network [6, 7]. There are numerous nodes densely deployed close to the phenomenon with or without their number or position being predetermined, or the nodes can be mobile; thus the topology of the network is variable and can change often [3, 8, 9]. The network can have controlling “base stations” or be self-organizing without guidance [6-9]. Self-organization helps to cope with failures in the network or in individual nodes as failure in a single node should not affect the performance of the overall network [3]. There is usually a sink node in the network, to which all the sensor data is routed and which is connected to end users [3]. In the nodes, data aggregation and

data fusing are used to reduce the amount of data in transfer [3]. This can also provide a richer view of the phenomena if multimodal sensors are used. An example of sensor networks is Smart dust technology [10]. It is a network consisting of integrated, autonomous and freely deployed sensor nodes which are minimized in size and costs. The nodes are millimeter-scale in size, use optical transmission, have a data processing unit, and use solar cells as power source.

Queries for data in a network have two types: data-centric, where a query is sent to a region in the network, and address-centric, where the query is sent to an individual node [6]. Data-centric queries should be used: giving an individual address to each node is costly and limitations in the memory and computing power of the nodes does not allow depending on an individual node. For data dissemination, there are also two scenarios: a sink node can receive one result per query or continuous updates of results per query [6]. Every node in the network should act as a router for data, which helps the data find the shortest route [6]. There are several routing protocols for sensor networks [3]: 1) a graph of minimum energy path can be constructed for the network, 2) flooding can be used to broadcast data to all the nodes regardless of whether they have already received it or not, 3) gossiping sends data to a randomly selected neighbor node despite of the time it takes for packets to reach the intended recipient, 4) data description is broadcasted and recipients answer if they are interested in that data type, 5) a tree is created from each node representing possible transmission routes to a neighboring node with power consumption taken into consideration, 6) clusters of nodes are formed where a randomly selected node in turn acts as a transmitter of data to the sink node, 7) nodes respond to the request of interest by forming a gradient field in the network through which the data is sent.

To produce geographically meaningful data, based on GPS, for example, localization techniques are needed with sensor networks. Routing of data also relies on location information. Because of the limited power available, nodes communicate with local neighbors. In the network there are beacon or anchor nodes which know their location a priori, and at least three beacon nodes are required to produce a two-dimensional coordinate system. Beacon placement has a major effect on accuracy especially when avoiding terrain irregularities and environmental or indoors obstacles. One problem with placement is that nodes in border areas in a network have fewer neighbors which influence accuracy in algorithms. Algorithms are also affected by node density. Computationally algorithms can be divided into four categories: 1) centralized algorithms, where an intelligent base station is deployed in the network where calculations are made and then transported back to nodes, 2) distributed algorithms, where nodes independently calculate the distance to few beacons and use this information to determine their location and create approximation of network, 3) another type of distributed algorithms add additional adjusting steps to the calculations based on error metric, and 4) coordinate system stitching where the network is divided into sub-regions each calculating optimal local network maps which are then merged. A radio transceiver in sensor nodes can be used for localization; there are algorithms that calculate signal strength or hop count of radio links between nodes, or the time difference of the arrival of signals and the angle of arriving signals can be measured. However, noise and physical obstacles complicate the calculations, and additional hardware, such as a speaker or a microphone, may be required. [8]

Power conservation, as for example battery life, is an important issue in sensor networks [3, 8]. Hardware components such as sensors, central processing unit,

peripheral devices and radio transceivers can be turned off when not in use and started again by watchdogs or events [3]. However it must be determined if turning on/off the components consumes more power than is otherwise saved. Routing and transmission protocols must be power aware, thus minimizing collision and duplication of data packets in the network [3].

Wireless network, broadband, radio frequency transmission, Bluetooth, optical or infrared communications are commonly used as transmission media in sensor networks [3, 5, 10]. Communication system capabilities are limited by the background noise, distortion and attenuation of signals when distances increase [9]. Optical transmission requires clear line-of-sight and links should be directional [10]. Frequency diversity and deployment of nodes can allow multiple transmission routes [9]. Reliability of higher frequencies or bit rate can be increased by lowering the distance between nodes, yet usage consumes more power [9, 10].

2.2. Middleware for mobile sensor network

Middleware's tasks are to provide system services and programming abstractions for diversity of applications, interfaces to heterogeneous sensor devices, runtime environment for the application, and efficient utilization of system resources [11-13]. Mobile sensor networks are one part of wireless sensor networks. In wireless sensor networks, data must be somehow collected from sensor nodes, and mobile data collectors offer several advantages over centralized networks [11]. Nodes will store data locally, can move closer to sink and provide data only when requested, which all can save energy. Coverage of a network can be extended freely which helps in case of data collection problems with, for example, a partially connected network; also if a sensor node fails, mobile data collectors can be used instead. There is no need for centralized infrastructure in a network which helps save costs. And finally users can query and collect data anytime from any part of the network. Mobile data collectors, however, have communication issues when transferring data to the sinks: communication channel bandwidth is limited, and routing is influenced by destination address, quality of service parameters and content of the data [11]. Routing protocols are divided into proactive and reactive protocols: proactive protocols always maintain route information and broadcast data when a link becomes available, and reactive protocols require flooding of the network to locate the sink whenever a node needs to send data [11].

In context with cooperative traffic, several characteristics for middleware can be found. A loosely-coupled architecture follows from the diversity of applications. Geo-spatial location of sensors can be established. Isolated data sources, heterogeneous communication systems and different application requirements set challenges for these systems. Within the same application data type, quality and quantity is variable. The lifespan of data is from milliseconds to years, and the source of data can be in-vehicle systems or intrusive or non-intrusive infrastructure sensors. [1, 14]

Design issues with wireless and mobile sensor middleware are: 1) the architecture should be data-centric, 2) application knowledge is needed to tailor the services offered by middleware, 3) localized algorithms should be used in sensor nodes whenever possible, 4) the middleware should have lightweight computation and communication requirements as nodes have limited capabilities, 5) quality of service can fluctuate in applications, as it is likely that all performance requirements of

different applications cannot be met simultaneously [12]. Several requirements can also be pointed out [13]. Asynchronous operation and component-based architecture should be used because of communication outages and the number of different communication technologies. Sensor application functionality should be dynamically changeable to adapt to changed circumstances. The middleware should have means for self-configuration and self-maintenance to achieve scalability and react to dynamic changes and failures in the network. The underlying operating system or I/O functionality should be hidden from applications and runtime priorities should be given to answer different needs of applications.

TIME architecture is a middleware for distribution and processing of traffic information. The system consists of sensor nodes, gateway nodes and relay nodes. Sensor nodes are location-aware, equipped with traffic sensors, and they have limited processing and storage capabilities. Sensor nodes can be, for example, infra-red camera nodes and, in the future, mobile phones. Sensor nodes filter data based on temporal and spatial correlations and compress it before sending to gateway nodes. Gateway nodes collect the data from sensor nodes and deliver it to the middleware for processing. Relay nodes ensure connectivity of sensor nodes and gateway nodes in sparsely deployed environments. The middleware works in an event-based manner and data flow is peer-to-peer between components. A component has two separate parts, business logic for data processing and wrapper to manage component communication via endpoints. Endpoints also have security roles for protecting privacy of data. [15]

CarTel developed at Massachusetts Institute of Technology is a middleware for mobile sensor nodes. Each node in the system is a mobile computer connected to several sensors. CarTel uses its own network stack where data is not buffered before sending, it just informs the nodes when a connection is available and then the nodes send data “at the last moment”. The main component in CarTel is the portal which handles control and configuration of the system and is a sink for all sensor data. CarTel is based on queries made by applications which cause nodes to generate stream responses of data. The data can be heterogeneous and is normalized in a sensor and then preprocessed to provide summaries before actual data values in streams. The data is then populated into PostgreSQL relational database in the portal, and applications issue queries to retrieve data from the database. Queries are snapshot queries meaning that the applications do not receive a continuous stream of data but run on whatever data is currently available. The portal includes a visualization library where data is operated as traces of, for example, a driven route. Applications are web-based and hosted by the portal framework, which also handles user authentications. [16]

2.3. Existing sensor network platforms

Implementations of generic ready-made sensor middleware were considered for the SDFA project as a sensor network server platform. A search was conducted in literature and the Internet for finding suitable middleware. The most suitable four architectures are presented here.

2.3.1. *Global Sensor Network*

Global Sensor Network (GSN) is a sensor network middleware developed in Ecole Polytechnique Fédérale de Lausanne. Lightweight Java implementation, usage of a subset of modules and minimal configuration needs make GSN deployable to all kinds of portable devices and standard personal computers. Communication in GSN is handled in peer-to-peer manner using standard Internet protocols. GSN provides access control mechanism for the whole system, or on the level of an individual sensor. GSN can dynamically adapt to run-time changes in a network, allowing new sensors to be added or client applications using sensor data. [17]

When launching the system, it looks for preconfigured physical sensors in the system and initializes them with corresponding wrapper Java class instances offering software interfaces to the sensors by itself. GSN keeps track of resource usage and shares hardware wrappers among virtual sensors. There is also a fault recovery system in GSN: unused sensors and wrappers are withdrawn from the system to release used resources, this also happens if a failure is detected in the sensor. GSN has a bundled web interface for monitoring and using the server. Physical sensors can be launched, used and monitored using the web interface anywhere in the network. It also provides a Java interface for implementing a personal control web interface for the user's physical sensors. XML documents are used to configure sensors, which can be added to the system in run-time just by adding a given configuration file. Remote sensor data streams are retrieved using given network addresses. The configuration file allows applying parameters to the sensor data for limiting system resource usage: limiting data stream rate, use of a windowing mechanism, setting boundaries to lifetime of data, data buffer size, and the number of virtual sensors that can access data simultaneously. GSN uses built-in database interfaces for HSQLDB and MySQL databases using JDBC. The HSQLDB database can also reside in memory for faster processing. The memory database is automatically saved to disk when the system is shut down and retrieved to the memory when the system is started. [17]

Figure 2 shows the GSN architecture. The figure is adopted from Book of GSN by GSN Team [17]. It can be split into three parts: data acquisition, data processing and data publishing. Data acquisition is achieved by hardware drivers known as wrappers in GSN. Each wrapper has a location given in GPS coordinates. GSN installation package has several ready-to-use wrappers, including HTTP, UDP, RFID, serial port, MICA motes, TinyOS and USB Webcams. Remote wrappers can be used to obtain data from other GSN server instances in the Internet. [17]

Data processing in GSN is event-driven: new output data stream element production is triggered by arrival of input data elements. GSN uses virtual sensors as data processing units, and each virtual sensor corresponds to the wrapper of a physical sensor or the data stream of another virtual sensor. They can receive several data streams either locally or from a remote system, and they will produce one output stream. Streams can be named to distinguish them from one another, and stream elements can have any number of any sizes of parameters. There are two interfaces for data processing: firstly a wrapper interface when retrieving data from physical sensor and secondly a virtual sensor interface which is used to write application specific data processing code. Data elements in the stream from wrapper to virtual sensor can also be filtered to produce actual data stream, which can also be shared among several virtual sensors. All operations of standard SQL syntax can be applied

to sensor data when filtering. Internally in GSN, a core engine hosts the virtual sensor containers which manage every aspect in the lifecycle of sensors. [17, 18]

Data publishing happens in output dispatcher module, and can consist of control messages to external devices or data output to visualization in the GSN web interface or in another client application. [17]

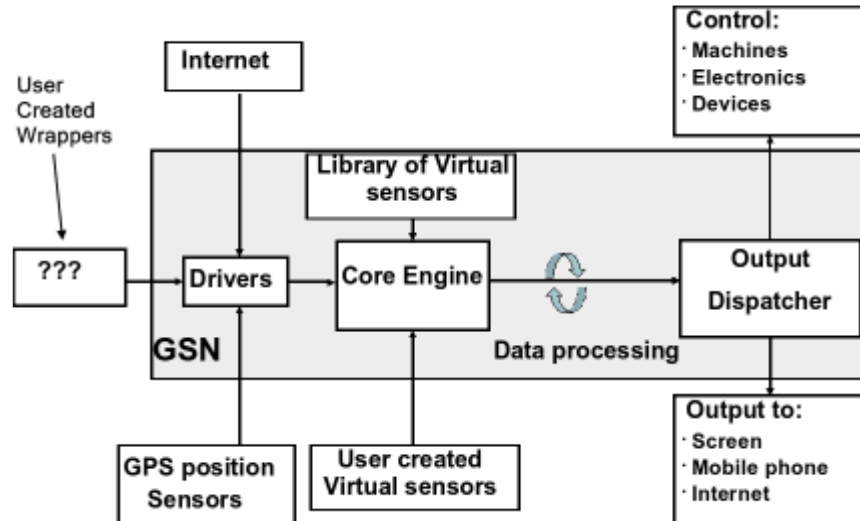


Figure 2. The Global Sensor Network architecture.

2.3.2. *IrisNet*

IrisNet (Internet-scale Resource-Intensive Sensor Network Services) is a research project at Intel Research, the goal of which is to design an architecture for Internet wide sensor network. IrisNet is made up of Internet-connected personal computers which act as hardware platforms for sensors, as local storage units for collected data and as source for sensor data feeds to respond user queries. IrisNet shares these sensors in a distributed manner to maximize usage. Figure 3 shows an example architecture. [19]

Physical sensors are accessed using a sensing agent generic interface. Sensing agents collect raw data from one or several sensors and store them in buffers, where services can have access to the data. Services access data using senselets, which are code modules for data processing, and can be uploaded dynamically to sensing agents. Each senselet is separated from others by executing the code in its own process, allowing the sensing agent to limit resource usage and provide fault tolerance. Senselets can share sensor feeds for limited distributed computing. Sensing agents can also use arbitrary privacy filters to limit access to raw data by dividing between trusted and untrusted senselets, the latter having access only to filtered data. After processing, a senselet sends the processed data to a nearby organizing agent which is controlling the relevant database part. [19]

Organizing agents collect the data for a single sensing service for a particular query to local caches. A sensing service can consist of any number of several types of sensors. Caches are used to improve the efficiency for repeated queries. XPATH 1.0 language is used for queries. Data is logically organized into an XML document in geographical hierarchy and with self-describing tags. Physically data is fragmented to a number of nodes in IrisNet, but it is seen as a single entity to which

queries are made in high-level languages supporting arithmetic and other operators. IrisNet uses an adaptive algorithm for partitioning database into organizing agents and for replicating database fragments between primary and secondary replicas. Primary replicas have a consistent copy of the data while the copy for the secondary ones is only weakly consistent. [19]

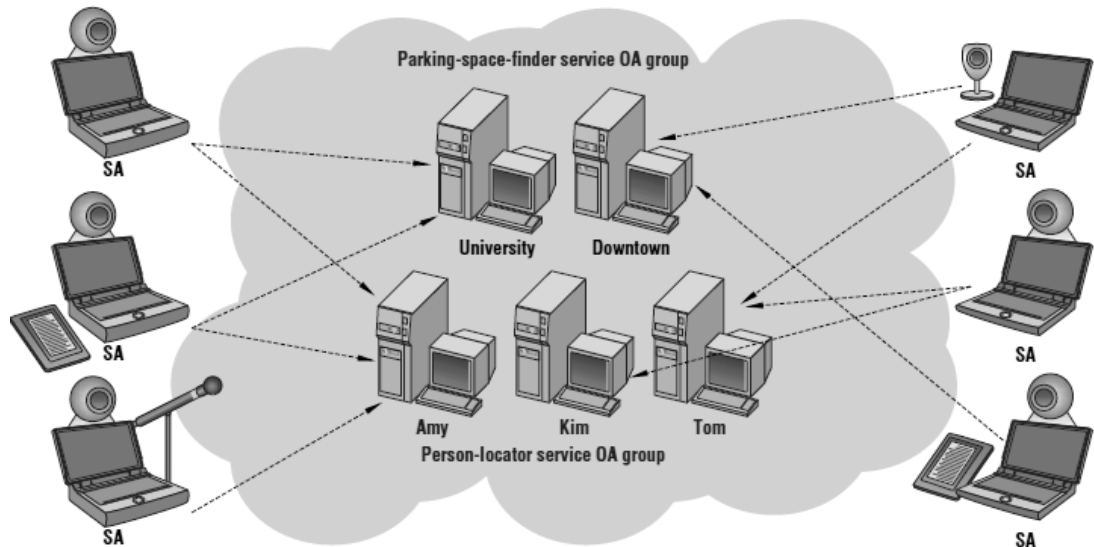


Figure 3. Example architecture of the IrisNet system.

2.3.3. Other sensor network platforms

Hourglass is an Internet-based middleware for connecting sensors, services and applications. The Hourglass system consists of dedicated machines that provide service registration and lookup service for discovery and routing of data streams from sensors to applications. Data flows through abstractions called circuits from producers to consumers which the circuit manager creates in response to requests from applications. Circuits also form links between nodes. Circuits are defined in XML-based language. Data routing is based on circuit identifiers making routing decisions trivial and resulting in low latency in each node. The system detects disconnected nodes using heart-beat messages, and allows remaining parts of the circuit still to function. Nodes in the circuit are realized as services. Service types are data producers, consumers or operators. Hourglass offers generic services for buffering, filtering and interface to persistent data storage, and also application specific services can be added. Service producers can consist of several nodes in a network and act as a unit. A service has a “topic” which is stored in a registry in the lookup service and is used for locating a data producer that realizes this type of a request. [20]

High Fan-In (HiFi) is a distributed architecture for sensor networks. It is aimed for large-scale networks using high-rate data producing nodes. HiFi is based on TelegraphCQ dataflow engine and query processor with TinyDB database system. Operators produce, process and stream data and are connected through “fjords”. Operators can be wrappers, file readers, sensor proxies, filters and routing operators. Fjords can also mix streams. TelegraphCQ uses adaptive algorithm and SQL-based

language to optimize routing in the system. Data queries are in turn optimized using TinySQL engine. There is no XML-integration in HiFi. [21]

2.4. Extensions to platforms

Additionally, service extension software libraries were considered on top of the sensor network server. Multithreading and shared memory libraries could offer significant performance improvements to the system. Several libraries were found by search in the Internet and literature, the most well-known of which are compared in this chapter.

2.4.1. Multithreading libraries

In distributed systems, multithreading techniques are essential, allowing processing of data and communication to coexist. Processes can be divided to threads for finer task granularity in processing units, thus achieving parallelism and better overall performance. Parallelism can be transparent as code migration is possible where code fragments can be distributed in a network to free processing units. Local communication between threads is done by using shared data. Threads can allow blocking system calls without blocking the entire system. Threads also hide communication latencies as the system can initiate communication and switch from one context to another thread while waiting. [22]

Adaptive Communication Environment (ACE) is a platform independent open source framework for concurrent software systems, written in C++. Implementation exists also for TinyOS sensor networks. The framework offers implementations for multithreading, synchronization of threads, dynamic linking of distributed services at run-time, shared memory management, and inter-process communication. ACE also provides The Ace Object Request Broker, which is a Common Object Requesting Broker Architecture standard based middleware for communication between objects. With the middleware, the framework can be extended to a sensor network system. ACE gives basically the same functionality as existing complete sensor network platforms, nonetheless it is missing interfaces to sensors. [23]

Boost is a collection of open source libraries extending C++. The package consists of 80 different libraries, from general purpose to operating system abstractions. Boost was considered for this project because it offers libraries for multithreading, streaming, inter-process communication and shared memory usage. Compared to selected sensor network platforms, there are mechanisms for inter-process communication and shared memory usage. However, using Boost in the project requires much more implementation work, as the whole system framework and interfaces to sensors need to be implemented. [24]

Threading Building Blocks from Intel is a library that allows scalable parallelism and multithreading with C++ [25]. It is specifically optimized for Intel or emulating multi-core processors. Threads are seen as tasks and program code is divided into tasks; then a task manager and scheduler optimizes the program execution automatically in a data-parallel manner. Netscape Portable Runtime is another platform independent package offering multi-threading, thread synchronization and network I/O [26]. Both of these packages lack distributed services and shared

memory management routines – thus they require substantial implementation work in order to be considered for the SDFA project.

2.4.2. Distributed shared memory

Distributed shared memory means a large shared memory to which each node in a network has access in addition to local memory in nodes [22]. There are both hardware and software based implementations. Distributed shared memory model conserves memory by offering a single view of common data instead of local replication of data in each node [22]. It also offers fast means of communication between processes as notifications circulate through a shared memory. Scalability is easily achieved in distributed shared memory, since clients can easily connect to the memory in run-time without interfering with other nodes. The shared memory can be organized in a page-based manner, where each memory block has a fixed size, or in an object-based manner, where objects can have variable sizes [22]. In an object-based system, the objects contain changes to themselves, so memory consistency is guaranteed at all times and the objects in memory are not replaced. Instead each object is placed into memory as a new instance [27]. A lease can be set for objects, and when expiring, objects are removed from memory by the system [28].

Space-based architecture is a software architecture where programs communicate through a distributed shared memory called “space”, which is shared among distributed processes [28]. Processes communicate asynchronously and concurrently by generating objects, known as tuples, and adding them to the space where other processes can withdraw tuples from [27]. All tuples in the space are accessible to all processes and one tuple can be shared by several processes, implementing master/worker pattern where a master dispatches tuples to the workers for work [28]. The system achieves linear scalability by adding new workers. Processes have no knowledge of one another, regarding where the tuple originates or which process will use it [27]. Tuples have persistent independent existence in space regardless of the lifecycle of the generating process, until explicitly removed from space [27]. Tuples consist of type name and parameters, which can be executable code or data values. There are three operations defined for tuple spaces [27]. $Out(N, P)$, where N is type name and P represents parameters, will insert the tuple into space. $In(N, P)$ will withdraw a matching tuple from space. Matching is based on a given type name and parameter list [27]. If none exists, $in(N, P)$ suspends until there is at least one available. $Read(N, P)$ will take a copy of the tuple from space but not remove it; it will remain there for other processes to use. Another paradigm based on tuple spaces is Object Spaces [29, 30]. Objects are created from classes as in object-oriented programming. Objects need to be registered into object directory when they are put into the space. Lookup service based on an object’s properties can then be used to find objects in space. Objects cannot be used while they are in space but they must be removed from space beforehand, and then deposited back to space. This provides mutual exclusion, only one process at a time can use an object’s methods.

Fly Object Space is an implementation of object spaces paradigm. It runs as a software service on a host computer or in a local network. It offers standard interface with $write(P, t1)$, $read(P, t2)$, $take(P, t2)$ and $snapshot(P)$ operations for manipulating tuples, where P is a template object of used tuple type, $t1$ is the time that an object will live in the space and $t2$ is the time for how long these operations will block the process. $Snapshot(P)$ operation makes a copy of the object template to be used,

which can internally improve the performance of the service. There is also an interface for writing several tuples at once to the space and reading from the space. Templates of tuple class have attributes for type name and parameters. Additionally, the system offers an interface for automatically receiving notifications of insertion and removal of tuples of a given template from space. [31]

TSpaces is another implementation of object spaces paradigm. It consists of two components, a tuple space component and a database component. The tuple space component offers a standard interface with Write(), Read(), Take() operations with additional operations WaitToRead() and WaitToTake(), which are blocking operations. Several tuples can be received using ConsumingScan() operation, and tuples can also be nested. Large data objects can be represented as URLs, and thus there is no need for consuming data transfers. TSpaces has an interface for defining new operations. For security reasons, users need to log into the space before using it. The database component offers a management layer to save and retrieve data from a relational database system, where TSpace operations are used in a transactional context. [32]

2.5. Comparison of existing systems

Existing systems with different feature sets were considered for comparison. Selected sensor network platforms were GSN and IrisNet, which were compared against the multithreading library ACE. General architecture, real-time performance, concurrency control, and distributed service management were compared with other minor features. The main feature and architectural differences are given in Table 1.

Table 1. Main differences in compared platforms and software library

	GSN	Irisnet	ACE
General architecture	Database-centric distributed system	Database-centric distributed system	Single server or distributed system
Concurrent processing	Multithreading	Processes	Multithreading/processes
Data acquisition methods	Events, polling or streaming	Streaming	Not implemented
Reported sample rates	75KB/s in 25ms without data processing	200hz for sensor, response time of 1s for 1-30 simultaneous queries	Not applicable
Ready-made input interfaces	XML-RPC, serial port, USB web camera, HTTP, UDP	Shared memory, web camera	Not implemented
Ready-made output interfaces	XML-RPC, HTTP, email	Shared memory	Not implemented
Offered distributed services	Adapts to runtime changes in network	Adapts to runtime changes in network	Not implemented, ACE ORB extension library required
Built-in database support	JDBC, MySQL, HSQLDB	XIndexe	Not implemented

2.5.1. *Selecting sensor network platform*

GSN's architecture is a distributed database-centric system, which means that data is stored in a database instead of memory, and data flows through the database between wrappers and sensors. An example server architecture is shown in figure 4. GSN is platform independent. GSN allows addition of pluggable sensors in runtime. GSN is implemented in Java and can thus be used in a multi-platform environment. Concurrency in GSN is realized by threading: all wrappers and sensors in system run in their own threads. Data acquisition methods available for GSN are file input, data streaming, polling for data or event-based acquisition. The system has ready-made input and output interfaces which include XML-RPC, serial port, HTTP, UDP, USB web camera for input and HTTP, email and XML-RPC for output. Any Java code can be used with GSN and Java has mechanisms for running external applications or services within the JVM. GSN has built-in database support for MySQL, HSQLDB and JDBC interface. The given data throughput rate for GSN is 75KB in 25ms without any processing. [33]

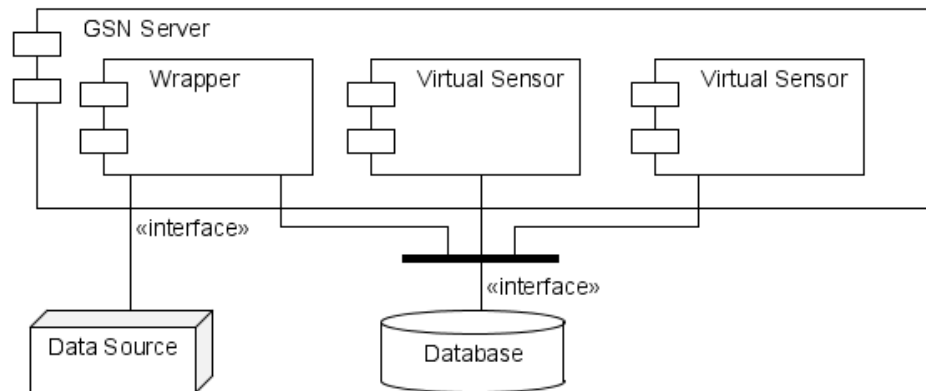


Figure 4. Example of server architecture.

IrisNet's architecture is also centered around a distributed database. IrisNet supports only the Microsoft Windows platform and is implemented in C/C++ languages. Concurrency is implemented by using processes as processing unit. External code can be run as executables or libraries. IrisNet adapts to changes, as adding new sensors, in network in runtime. IrisNet has a built-in distributed shared memory for faster data processing and streaming, but data can reside in local caches in nodes in the sensor network. For this, there is also a distributed database query processing mechanism, which gives a one second response time for one to 30 simultaneous queries of measurement data. A supported database system is XIndex, which is a native XML database. IrisNet's ready-made input interfaces are shared memory and USB web camera, with an output interface to shared memory. A major issue with IrisNet is that it is no longer under development. [19]

ACE offers distributed services with ORB in a single server environment. ORB allows dynamic linking of services at runtime. It is implemented in C++ and platforms include Microsoft Windows, Linux, posix and some real-time operating systems. Concurrency can be implemented with both processes and threads. Implemented input interfaces are sockets and shared memory; output interfaces are

HTTP and shared memory. There are no specific data acquisition methods or ready-made database support. [23]

Clearly GSN offers the most ready-made features and options of the compared sensor network platforms. Compared to IrisNet, there are more ready-made input and output interfaces which are usable in the SDFa project. GSN is platform independent as IrisNet is developed for Microsoft Windows. Also systems with more than one database are supported and GSN is still under continuous development. In GSN, it is possible to configure where data is stored, but in IrisNet data is stored in the network in local nodes. Irisnet, however, has built-in XML support which could be useful when considering development of internal data formats for the system, and code modules in IrisNet can be loaded dynamically to sensor nodes. Sensor data rate in the vehicle is around 500KB per second if streaming from video camera is used, and GSN should be able to handle this easily. This data rate estimate was calculated from data sets collected earlier in the project. With ACE there is a lot more implementation work to be done. Hourglass and HiFi are more focused on robust fault tolerant data flow, routing and connectivity issues, and there are no publicly available implementations to use. Boost, Threading Building Blocks and Netscape Portable Runtime all require implementation work in order to provide the same functionality as ACE, and GSN itself already provides multithreading.

When compared to design issues and requirements given in chapter 2.2, GSN has a data-centric component-based architecture, localized algorithms are used within wrappers, and priorities can be configured for each component. The GSN middleware is generic but can be modified to fulfill service requirements of different applications, while applications and sensors can connect to the system in runtime so a different functionality can be introduced. GSN communication interfaces are generic and asynchronous, they can be tailored as lightweight as possible, and a fault tolerance mechanism is implemented. TIME [15] architecture is data-centric, component-based, and tailored for collecting and processing traffic data. Localized algorithms are used in sensor nodes, but they are not lightweight, as for example compression algorithm is used. Runtime priorities cannot be given for nodes. TIME can adapt to communication failures in the network but otherwise there is no quality of service control. Application functionality cannot be modified or different types of sensors cannot be added to the system. CarTel [16] also has a data-centric component-based architecture and is tailored for mobile sensor nodes. Localized algorithms can be introduced in nodes, but there is no mechanism for fault tolerance. Communication is asynchronous and uses personal network stacks with buffering, but quality of service cannot be guaranteed. Self-maintenance is implemented for communication in nodes. Application behavior or different types of sensors cannot be added or changed in runtime. A major issue is that only parts of the CarTel implementation source code are released today.

When comparing GSN to TIME and Car Tel, the architectures are very similar but only in the GSN application is it possible to change functionality in runtime as well as add sensors and applications to the system, also in runtime. Only GSN has a fault tolerance system for sensor nodes. Possible communication problems are better handled in TIME and CarTel, but these are out of the scope of this work and studied in other cooperative traffic program projects. The source code for the implementations of TIME and CarTel is not released completely for public use and it is not clear if the TIME implementation can be modified for different applications at all. Therefore, GSN was selected as the middleware for the SDFa project.

2.5.2. *Selecting shared memory implementation*

Server implementation with GSN may require shared memory system in parallel with a database for real-time processing of sensor data if the included database system is not fast enough. Tuple space or object spaces were considered a suitable shared memory model because of the distributed nature of the paradigm, linear scalability, similarity of the search capabilities of the database systems, and implementation simplicity.

Two implementations were considered on top of the GSN platform, Fly Object Spaces and TSpaces. Both of these implementations offer the same services with essentially no differences. An example server architecture with tuple space implementation is shown in Figure 5. Data flow sequence diagram with tuple space is shown in Figure 6. Implementation is set up as an additional layer on top of GSN and can be used for data distribution instead of GSN's streaming interface. A data source wrapper would insert data packets to shared memory using `write()` operation. A virtual sensor would fetch data packets from memory by using `read()` operation and process the data and then insert the modified data back to shared memory by using `write()` operation. Finally a virtual sensor connected to the database would remove the packets out of memory by `take()` operation and store them to the database permanently. Notifications are sent internally in the shared memory system to notify virtual sensors when there is data available in the memory, and also when data is no longer needed and can thus be removed. After each step in processing, the type of packets would change for forthcoming virtual sensors to define suitable data for them.

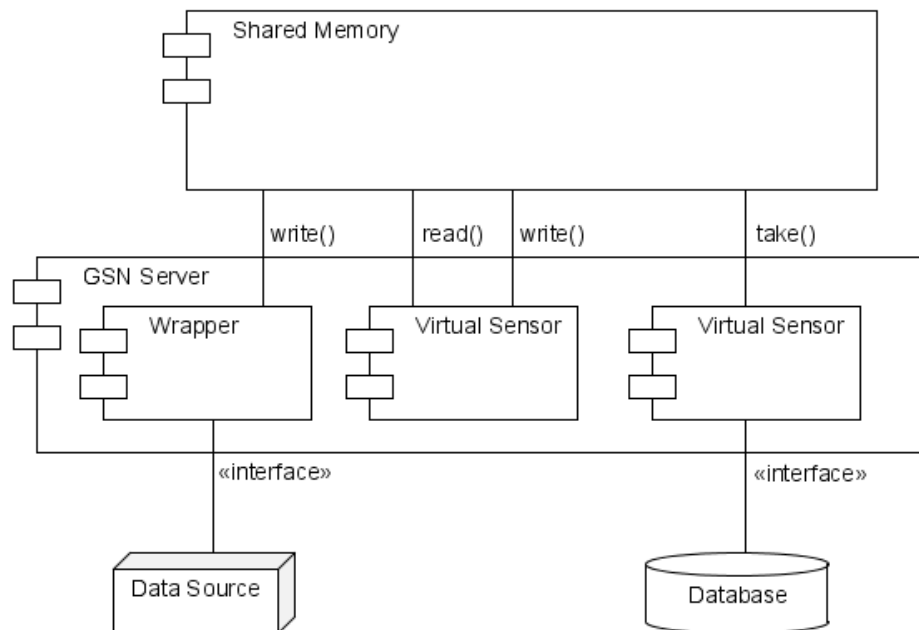


Figure 5. Example of server architecture with tuple space.

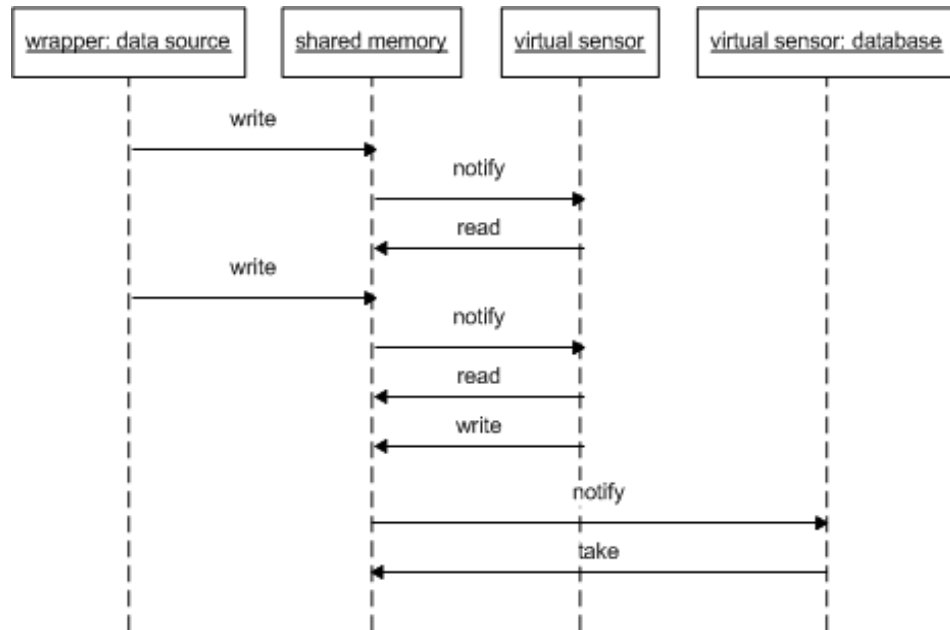


Figure 6. Data flow sequence diagram in server with tuple space.

2.6. Feasibility of selected systems

Tests were conducted in order to test the overall feasibility of the selected systems for the SDA project. Because both object spaces implementations offer identical services, testing was needed to find the faster implementation. Table 2 shows the test cases.

The testing platform was a normal desktop personal computer with Windows XP Professional Service Pack 3 as the operating system. The central processing unit was Intel Core 2,40 GHz processor with 2GB of memory. Java version 1.6 was used with GSN version 0.95. MySQL version 14.14 with distribution 5.1.31 was used as the database system. HSQLDB version was 1.8. Fly Object Space version was 1.1, and TSpaces version was 2.1.2 with service pack 3. Data packets were created before streaming so that memory allocation time would not affect the results. After sending each packet, there was a 20ms sleep period to allow other threads to execute in the server before sending next packet. This creates overhead time in the measurements, but as this will be a consistent delay in every test from now on, it can be ignored when interpreting the results. Without this delay, the GSN's internal packet delivery would crash.

Table 2. Feasibility test cases

Test case 1: One GSN instance running. One wrapper streams 10,000 data packets of varying size through database to one virtual sensor.
Test case 2: Object spaces implementations running on top of GSN instance. One wrapper streams 10,000 data packets of varying size through shared memory to one virtual sensor.

2.6.1. Global Sensor Network

In test case 1, the overall feasibility of GSN implementation with built-in database systems was tested. Figure 4 shows the GSN architecture, where the wrapper was

used as data source and sent 10,000 consecutive packets of sizes 1KB, 10KB and 50KB through database to first virtual sensor without any processing of the data. Time between sending and receiving each data packet was measured. The tested database systems were MySQL, HSQLDB in file and HSQLDB in memory. There was a limitation of data packet size in GSN with MySQL as the maximum packet size was 65KB. JVM memory size was set to 128MB as in default GSN system configuration.

As results in Table 3 show, HSQLDB in the file is the fastest choice but there is no real difference whether the database resides in file or in memory. With packet size of 50KB/s data throughput rate is 1,7MB/s, compared to the SDFA project's example data throughput rate 500KB/s. Even with the slowest tested throughput, MySQL with 50KB data packets, throughput is more than required as it is 625KB/s. Therefore GSN is capable of receiving data from sensors fast enough in the context of the SDFA project as wrappers are considered interfaces to sensor nodes or physical sensors and data packet sizes can be customized optimally for each sensor data type.

Table 3. Feasibility test results for GSN

Packet size	MySQL	HSQLDB in file	HSQLDB in mem
1KB	68ms	31ms	31ms
10KB	66ms	31ms	32ms
50KB	80ms	30ms	32ms

2.6.2. Object Spaces

Test case 2 is similar to test case 1, except that data does not flow through the database but both shared memory implementations at a time. Figure 5 shows the architecture, where data source wrapper sends data packets to shared memory using write()-operation and then the first virtual sensor retrieves packets from shared memory using read()-operation. Time between sending and receiving each data packet was measured. The tested packet sizes were 1KB, 10KB, 50KB and 1MB. There is no set packet size limitation for any of the implementations, so even larger packet sizes are possible. Both systems require an external server running in the same platform or a local network.

Test results are shown in Table 4. Fly Object Spaces was considered the best option for object spaces implementation because of better data throughput rate with larger packet sizes than TSpaces. Overall, as test cases 1 and 2 show, object spaces implementation on top of GSN offered no significant performance improvement compared to HSQLDB in the test platform. There is no data packet size limitation in either of those tested. HSQLDB uses JDBC interface and is built-in with GSN, thus there is no need for additional software layer implementing distributed shared memory.

Table 4. Feasibility test results for Object Spaces

Packet size	Fly Object Spaces	TSpaces
1KB	31ms	31ms
10KB	30ms	31ms
50KB	30ms	34ms
1MB	31ms	100ms

3. DESIGN

First in this chapter, generic distributed server design issues are given. Then requirements and description for the sensor data processing server in the SDFa project context are given, and finally prototype server design and specifications are presented.

3.1. General design issues

Servers can be architecturally divided into iterative servers which themselves handle the requests and provide a response if needed, and concurrent servers which start a new thread or process for each request. This thread or process will handle the response itself. Servers can be stateless or stateful. Stateless servers do not keep any state information of clients and forget requests when they are processed. Stateful servers keep information of clients in tables, for example in (client, file) pairs or history information of a client's behavior. Recovering from faults is a problem with stateful servers, as they need to resume the entire state of a server just before the crash. One design issue is how clients communicate with the server: globally defined endpoints could be associated with services or there could be a superserver in the network to handle all the endpoints and assign incoming requests to new processes for processing the request further. For interrupting server requests, there are several possibilities: cancel the client's connection to the server or establish a separate control endpoint where control commands can be sent. Object server is a type of server used with distributed systems. The server only provides the means to invoke local objects and act as a place where objects live. Objects can be added or removed from the system freely. For invoking an object there are two mechanisms: there can be one similar interface for invoking all objects or the objects can be transient, invoked in the server only when needed. Object adapter in the server can be used for activation of objects which could have different activation policies. [22]

3.2. General requirements

The purpose of the prototype server is to allow rapid development and testing of external code and algorithms in a ready-made sensor network environment. Main components in the server are input interfaces to retrieve data from sensors, data processing modules, and output interfaces to client applications and to the database to store data permanently. The system should be scalable in runtime and allow sensors, data processing modules and applications to connect dynamically to serve different needs that occur during data collection and processing. The software should allow real-time concurrent processing of data coming from several different sensors, also combining it with previously collected data stored in a database, for example. This architecture can also have distributed system services, for example distributed shared memory, with several server platforms each connected to several sensors. A general system description is given in Table 5.

Table 5. General system description

Platform independent	There could be a number of server instances each running on its own platform and operating system
Real-time performance	Concurrent processing of several data streams coming from different sensors, rapid propagation of data elements between components using distributed shared memory
Data storage	Database server should be running within the system as permanent data storage
Dynamic allocation of components	Sensors and applications should be able to connect to server freely in runtime
Running external code	Server should be able to integrate data processing code written for separate platforms in different languages
Scalability	New additional sensors should be able to connect to system without bringing down the system performance
Mobile phone as data source	Implement data acquisition methods from mobile phone to server in on-line mode
Generic data output interface	Implement interface for external client applications to request processed data

3.3. Operations for sensor data

In the sensor data fusion domain, information processing can be divided into three levels: Sensing devices level, sensor fusion level, and decision level [34]. In the sensing devices level are the physical sensors and sensing technologies. Sensor fusion level consists of operations for detecting features in the data. In the decision level are services, applications and information management. Integration of multiple data sources are often required to provide useful information for the decision level. In the SDFa project, the sensor data fusion model has three levels of processing: sensor refinement as data fusion, feature fusion, and decision fusion [34]. These levels can be seen in Figure 7. Data fusion level consists of preprocessing and fusion of raw data coming from vehicle sensors and other sensor nodes such as mobile phone. The resulting data is normalized and noise in the data is filtered. Synchronization based on time is also performed on this level. On the feature fusion level, characteristic variables of the phenomenon are extracted from the data and fused. The variables are application specific. Reliability of the estimation is also assessed on this level. In the SDFa project's first phase, data produced by accelerometer sensor is synchronized and processed with GPS location data. On the decision fusion level, features are fused to recognize an object or produce estimation of situation. Objects that are detected and identified in the SDFa project are road anomalies and environmental conditions, events or physical obstacles on the road. Collected features are stored in a database and it should be possible to make queries online or offline based on event type, location, route, road conditions and time. Feature extraction is done using external code modules and these algorithms are application specific and will be studied later in the project. There is also a need to deliver the collected query results as decisions to users, which is done in the decision

fusion level. In the SDFa project's first phase, there will be a visualization client application for located road anomalies.

From these operations, the following use cases can be discovered. Use case "Extract features" has a generic template for running external code or algorithms in the data processing server on the feature or decision fusion levels. At the data fusion level it may be needed for preprocessing and filtering the raw data. Data fusion and synchronization methods should be available for every processing level and unit. This functionality is implemented in GSN's virtual sensors or in algorithms in external sensor nodes. At all levels, a database interface should be available for "Store data" use case. "Collect data" use case should have the means for retrieving data from sensors and old data from persistent database storage. This is implemented in GSN's wrappers. Client applications should be able to specify their own characteristic variables for features and decisions in "Provide client data" use case, implemented with virtual sensors. Use cases are shown in Figure 8.

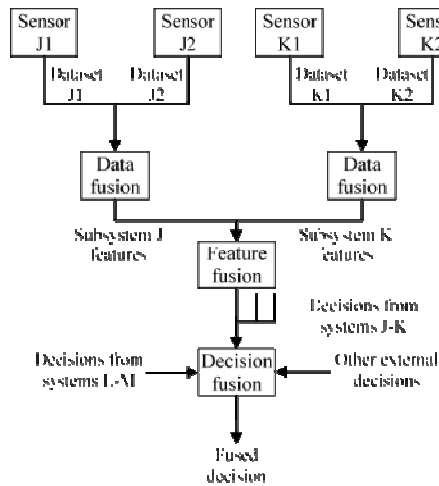


Figure 7. Three levels of data fusion model.

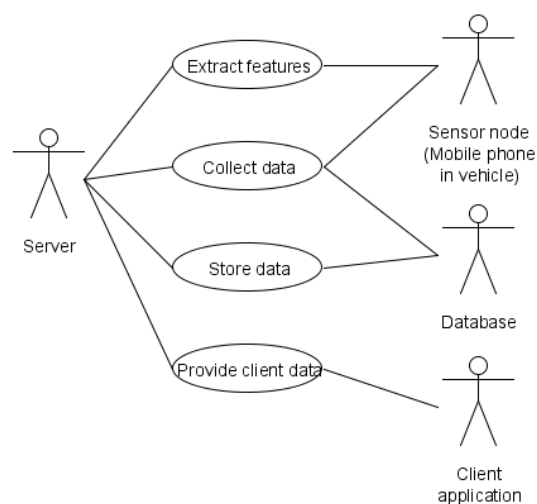


Figure 8. Use cases for sensor data operations.

3.4. Data types and structures

In this work, consideration was given to uniform internal data presentation. Sensor data sources can have a different nature, data representation schema, scale, rate and priority. Thus there should be generic means for data collecting, processing, retrieval and storing. Processed data elements should be made available for all processing units consuming a given data type. With distributed sensor networks, master/worker pattern is a scalable solution for propagating data elements within the system. The database system should be used to store data permanently, which affects the selected data format. Typical queries for the data could be such as “locate and show all pot-holes in the given area”.

Sensor data types in the first phase of the SDFa project are: numerical GPS location data, accelerometer data, extracted feature vectors, and type of anomaly in location. Location and time are two important features associated with the collected sensor data, and queries for data should be made based on both or either of those. Sampling rate and duration of the data sets are variable. Also, any kind of metadata could be potentially associated with any data. Three different data structures are derived from these data values and the data fusion model for the SDFa project. First there is synchronized data, where accelerometer data is synchronized with location GPS coordinates. Secondly, there are feature vectors extracted from the synchronized data and, thirdly, classified data from the feature vector, where the type of location of the feature vector is classified. Table 6 lists these structures. Data flow combined with operations for data in the system is given in Figure 9.

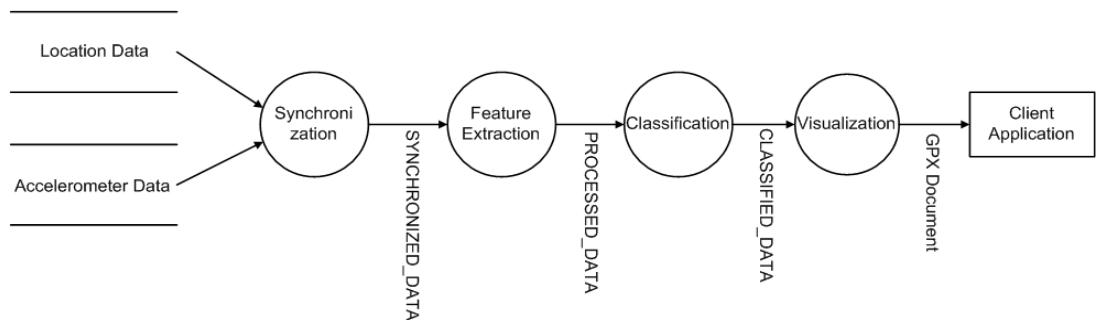


Figure 9. Data flow with operations in system.

Generic output data format and interface was also one requirement for the server which is needed for client applications to request processed data. XML is one solution for data presentation format, as it is a markup language designed for structured information storing and handling in documents [35]. XML documents consist of freely specified elements and attributes. Therefore an open format based on XML allows as generic a usage of the processed data as possible. GPS Exchange Format (GPX) was selected as data presentation format for visualization [36]. It is an open standard data format for representation of GPS data in the Internet. It gives a common set of tags for describing data and yet it is simple and light-weight. Basic elements in GPX documents are: waypoints, routes, tracks, associated metadata and private attributes. Elements allow usage of all kinds of user-defined private attributes and values, which will serve different visualization needs in the future. Several geographical map applications on the Internet support GPX, for example OpenStreetMap and even some mobile phone applications [37]. There are several

ready-made libraries in Java to create and use XML documents. XPath or XQuery could be used to process data in documents [38, 39]. XStream is usable for transforming and serialization of data from any format to XML and back [40]. Database systems like XIndices also support queries directly to stored XML documents [41].

Table 6. Data stream structures

SYNCHRONIZED_DATA	
TIME	Timestamp
LATITUDE	Latitude coordinate
LONGITUDE	Longitude coordinate
ALTITUDE	Altitude coordinate
ACCELERATION_X	Acceleration in X axis
ACCELERATION_Y	Acceleration in Y axis
ACCELERATION_Z	Acceleration in Z axis
ID	Client ID

PROCESSED_DATA	
LATITUDE	Latitude coordinate
LONGITUDE	Longitude coordinate
ALTITUDE	Altitude coordinate
FEATURES	Feature vector
ID	Client ID

CLASSIFIED_DATA	
LATITUDE	Latitude coordinate
LONGITUDE	Longitude coordinate
ALTITUDE	Altitude coordinate
TYPE	Type of location
ID	Client ID

3.5. Interfaces to external services, sensors and applications

One requirement for the system was for it to be able to run external code modules or algorithms as services within the server. Code can be run as external executables, code libraries, linked object libraries or on a separate platform in the network. Another need is to develop interfaces to access data from sensors not within the GSN platform, for example the vehicle's own internal network or data gathering platform, and to send the processed data to external client applications.

MATLAB was considered one platform where to run external algorithms. There are a number of ways to send data and awake processing from the server: 1) data can be written in files in readable form, 2) function objects in MATLAB runtime can be directly called from the server, 3) socket communication can be used to stream data to runtime or executable, 4) code modules as executables can be launched from the operating system or server using command-line parameters [42]. Reading data directly from database is also possible. Because MATLAB can be run anywhere in

the network, socket communication to executable was considered to be the best option because it allows simple and fast streaming of data to any node in the network. Data elements could be sent as comma-separated values string for simplicity.

In an on-line server, a defined interface is needed to receive data from external sensor nodes such as mobile phones and send the data to client applications. It was decided with the project participants that mobile phones will use HTTP protocol with socket communication on top of GPRS or WLAN to deliver the collected data files to a web server running in the server platform. An external visualization application based on JavaScript was developed by other project participants in the SDFa project. Communication interfaces were cooperatively designed. HTTP protocol on top of socket communication was selected as the communication method between the server and the visualization application. GPX documents are used as output data presentation format.

3.6. Server architecture

General server architecture is drawn in this chapter from the given requirements and design issues. GSN implementation can be seen as a stateless object server: it only invokes objects based on requests from virtual sensors, which can be considered objects adapters, and objects are alive only when requested. It also uses global endpoints for communication with clients.

Platform independency in the server can be achieved using common programming languages and since GSN is implemented using Java, this is a natural choice. Any Java class libraries can be used within the GSN server. A general server architecture for the prototype system is given in Figure 10. Wrappers are considered data producing units in the system and are connected to a physical sensor, sensor nodes or external data sources. Virtual sensors are both the data producing and processing units in the system as processing results are sent further in the system. In data processing, algorithm chaining is required because of the need for phased processing of data elements. This is achieved by configuration of data streams between virtual sensors which will then automatically recognize available input data. If data processing units require external components or specific servers in the system, their interfaces should be implemented in corresponding wrappers or virtual sensors as templates.

As for the prototype server's input interfaces from local files or mobile phone, file and socket I/O are sufficient as interfaces to data sources. The only data output interface needed is for the visualization client application. Thus, two input interface wrappers to retrieve collected data sets and one virtual sensor to output processed data to client applications are needed for the I/O interfaces from the server to global endpoints. For the external executable's input, one outputting virtual sensor template and one inputting wrapper template is required for using MATLAB executables with the prototype server. A virtual sensor would send data elements to the executable and wrapper receive them from the executable. The template allows free use of external executables as long as the interface to and from the GSN server is not changed. Additionally, one virtual sensor is needed for retrieving old data from the database for data fusion and processing, but this is not required in the first year tasks of the SDFa project. Polling of incoming data in wrappers and in virtual sensors connected

to external systems is a suitable data acquisition method, because each case is run as a separate thread and is thus non-blocking.

For streaming data elements within the system, there are GSN's built-in database interfaces. Wrappers and virtual sensors stream data elements through the database where data is stored internally, and internal mechanisms take care of the data delivery. Master/worker pattern can be achieved by configuration of virtual sensors, as there is no limitation for the number of stream recipients and for the number of streams that a virtual sensor can receive. GSN has built-in database interfaces to MySQL and HSQLDB using JDBC so there is no need to implement separate interfaces.

GSN has a built-in mechanism for adding wrappers and virtual sensors to the system, and thus requirements for dynamic allocation of components and scalability are fulfilled. Several instances of the GSN server can run in the network, and interconnection of virtual sensors in different systems can be achieved just by configuration.

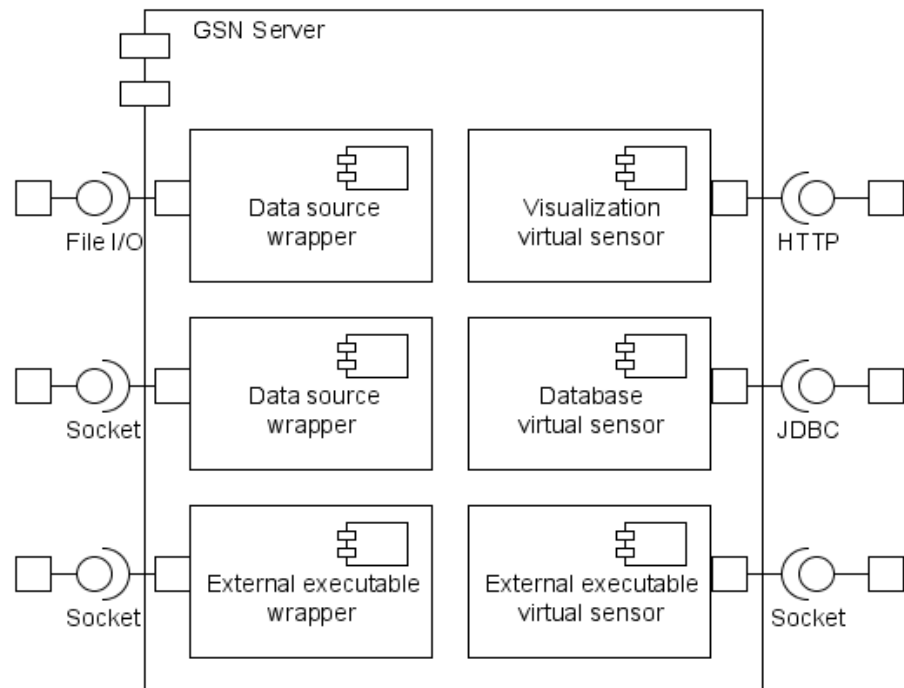


Figure 10. Prototype server architecture.

4. IMPLEMENTATION

In this chapter, the implementation of the data processing server and data collecting mobile client are presented. The data processing server is based on GSN sensor network platform. GSN offers a complete functionality to run a template of the sensor network server but some additional configuration and implementation work is needed to fulfill SDFA project requirements. First, a description of the operating environment and the whole system is presented, and then detailed descriptions of the functionality and configuration of GSN server components are given.

4.1. Server system and environment

GSN server and additional modules are run in a test-bench in Windows XP Professional Service Pack 3 operating system on a dedicated VMware virtualization platform in a public network for receiving on-line data from mobile phone data collecting clients in vehicles. The platform has 1GB of physical memory and processor speed is 2,83GHz. Figure 11 shows the deployment diagram of the on-line system. The server workstation has an Apache web server which receives collected data files from the mobile phone and also runs scripts needed for visualization client applications. The GSN server works as a middleware for the sensor data processing tasks using wrappers, virtual sensors and classification software library. MATLAB executable runs algorithms used for feature extraction. These components are described in detail in following sections. The collected sensor data is stored in HSQLDB database in the server workstation. The off-line system is similar, except data is read from files located on the server workstation. A visualization application called OpenStreetMap can be used in the web browser on a separate workstation from anywhere in the network.

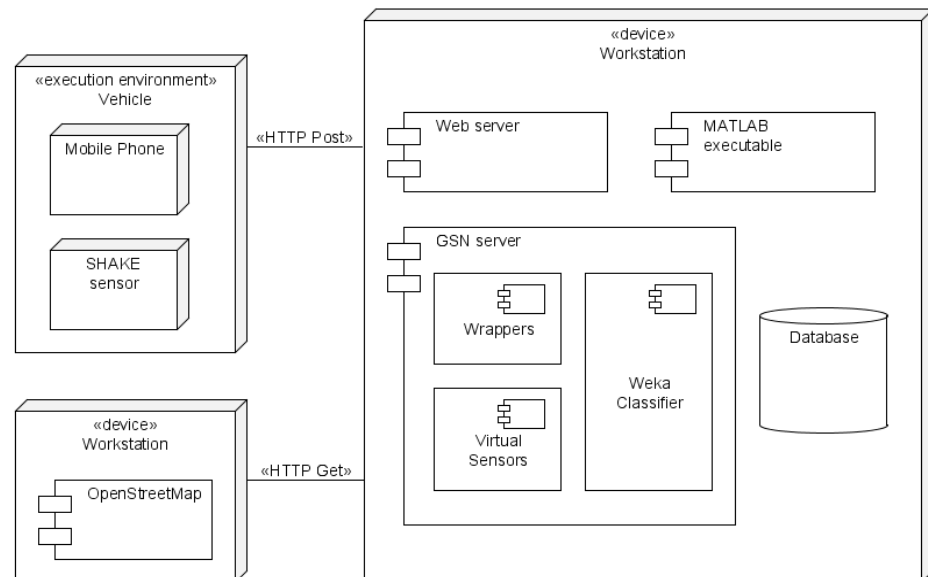


Figure 11. Deployment diagram of on-line system.

The mobile data collection client was based on code written in Python by research partners in the SDFA project. It was written for S60 software platform for Nokia mobile phones. The mobile phone model used was Nokia N95. The code was modified to comply with the prototype server's interfaces and data structures. The software collects GPS location data from the phone's built-in GPS receiver and accelerometer data from a built-in sensor in the mobile phone. Also Bluetooth connected SHAKE SK6 accelerometer sensor can be used [43]. The GPS receiver gives coordinate data maximum once in a second and the accelerometer sensor outputs data in the frequency of 100Hz. GPS data consists of latitude, longitude and altitude values and accelerometer data includes values for each x, y and z coordinates. Collected data in the mobile phone is stored locally as two files, one containing GPS data with timestamps and the other containing accelerometer data with timestamps. In the prototype, no preprocessing or filtering of data is done in the mobile phone's software, except when the needed built-in accelerometer sensor data is converted to the same scale as SHAKE SK6 sensor data. Data files are sent to the workstation's web server using HTTP POST request on top of WLAN or GRPS communication in configurable regular intervals, for example every 5 seconds.

4.2. Prototype application

Based on requirements and specifications drawn in chapter 3 and previous sections in this chapter, a prototype GSN server for online and offline processing of the collected data sets was implemented. A component diagram of the prototype application with component external interfaces is given in Figure 12.

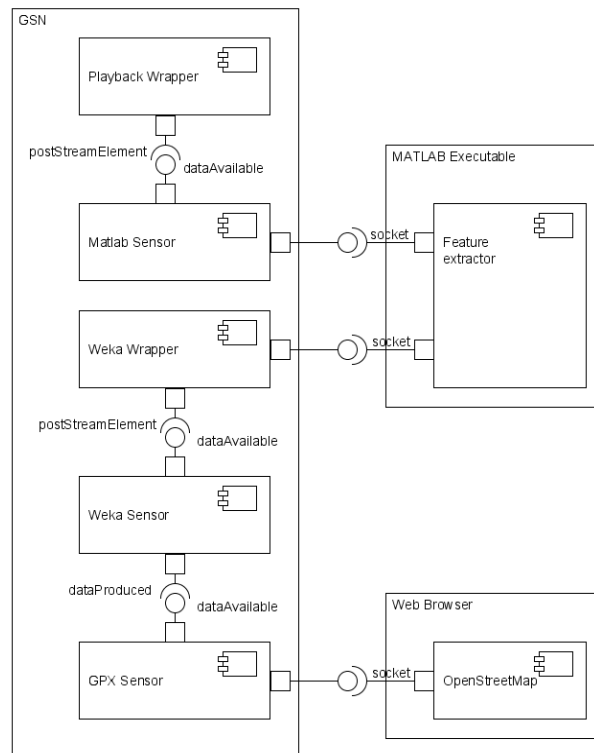


Figure 12. Component diagram of prototype application with component interfaces.

In the workstation web server, there is a PHP script running and forwarding the received data files using sockets to the GSN server for online processing. This script is modified to comply with GSN server interfaces from a version developed by other project participants for their own use. Client id is given in association with the data files during file uploading and this id is later used by the visualization application for requesting data of a given client from the server. Off-line processing and visualization in system is done in similar manner, except data is read from files on the server platform and not transferred from the mobile phone.

In the GSN server, Playback wrapper reads the data in files, synchronizes it and then streams it forward to the MATLAB virtual sensor. The sensor connects to a running instance of MATLAB code executable to enter data buffers. This executable is run in the system as separate independent process. In the executable, the received data is processed in algorithms developed by other project participants. After processing, the executable sends feature vector to Weka wrapper in server, and an acknowledge message back to the MATLAB virtual sensor to announce readiness for a new data buffer. Weka wrapper receives the feature vectors and streams them to GPX virtual sensor to be transformed to a specified GPX document format. GPX sensor then waits for client application's requests for data.

A Javascript object is run in the web server to respond to visualization client application requests for data. In the object, requests are passed to the GPX sensor in server which in return sends a GPX document back to the object where it is visualized. The output from the visualization application is given in Figure 13. Each feature vector collected in the route is shown on map, "Road" as transparent red diamond sign and "Pothole" as warning sign. Current GPS position is given in the top left corner and current position as a non-transparent red diamond sign. The Javascript code in the web server was developed by other project participants.



Figure 13. Example of visualization client application's output.

4.3. Global Sensor Network modules

GSN is implemented using Java and it offers templates of parent classes and interfaces for implementing the user's own virtual sensors and wrappers. For each wrapper or virtual sensor, there is a Java class instance to be run from GSN main module when data becomes available or is requested. Templates contain methods for receiving, processing and sending data streams. Wrappers and virtual sensors' output methods stream data elements onward if required, since there is no requirement for a sensor or wrapper to output anything. This will not break the GSN architecture if data at some point does not reside in the database from where it can be retrieved by other virtual sensors. Input and output data stream formats are specified in configuration files and implementation Java classes, to determine which sensors can receive data from which wrappers or other sensors. This, in general, implements a master/worker pattern. The implemented modules and their interfaces can be seen in Figure 10 in chapter 3.6 and are described later in this section.

4.3.1. Wrappers

From SDFA project requirements and components described in previous section, two kinds of wrappers were implemented. Playback wrapper reads coordinate data and accelerometer data from separate files on the server workstation and synchronizes them to one data stream for output. Location-based synchronization is done by using the read GPS coordinates for accelerometer data until their timestamp overruns the coordinate timestamp, and a new boundary timestamp is read from GPS coordinate data. A stream element consists of timestamp, latitude, longitude, altitude, acceleration x-axis data, acceleration y-axis data, acceleration z-axis data and client id. This wrapper outputs the stream SYNCHRONIZED_DATA. Another playback wrapper, inherited Java class from Playback wrapper, was implemented for receiving data file names as HTTP POST request through socket in on-line from a PHP script running in the web server. File names point to files received from a mobile client and saved by PHP script to the server platform. Otherwise the functionality is similar.

Weka wrapper starts a socket server, waits for socket connection from a MATLAB external executable and then receives feature vectors one by one from the executable or from any other data source if applicable. Weka is a machine learning and data mining software library which offers Java implementations for several machine learning algorithms [44]. The data format is a comma-separated values string that includes middle point coordinates of the received buffer, generated feature vector and client id. A feature vector is then streamed to the Weka virtual sensor. A stream element consists of latitude, longitude, altitude, feature vector and client id as comma-separated string. This wrapper outputs the stream PROCESSED_DATA.

4.3.2. Virtual sensors

For the prototype system, three implementations of virtual sensors were written: sensor for sending data to MATLAB executable for feature extraction, sensor for sending feature vectors to Weka classifier and sensor for outputting GPX data to client applications. For the sake of simplicity, the virtual sensor's internal

implementation details such as threading, messaging, data buffering and data handling are left out from the activity diagrams presented in this section.

MATLAB virtual sensor's task is to receive coordinate and accelerometer data stream from playback wrapper, and send the data to MATLAB executable. An activity diagram of this virtual sensor is shown in Figure 14. A socket server is started for sending data values. Received values from the stream SYNCHRONIZED_DATA are sent as comma-separated values string in buffers of arbitrary size when a specified time limit, for example 500 milliseconds, calculated from the timestamp of the data has expired. In the executable, feature extraction is done by external MATLAB algorithms, which are developed in the SDFa project separately by other participants. Then the socket server waits for an acknowledging message from the executable before sending the next data values. This is needed because data processing in the executable can be time consuming and otherwise communication would suffer from a data overflow condition. Also, MATLAB executables can only be used as a single-threaded process; thus it is not possible to implement threading where one thread is used for communication and for buffering data values in the receiving side and another thread for data processing.

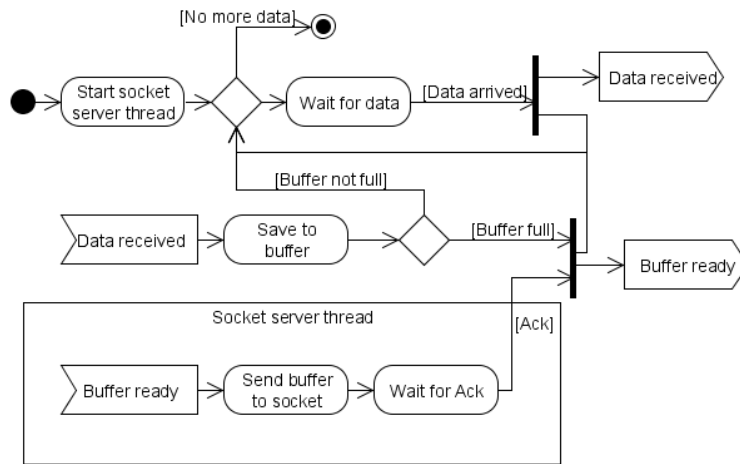


Figure 14. Activity diagram of MATLAB virtual sensor.

Weka virtual sensor executes feature classification for the feature vector received in the stream PROCESSED_DATA from Weka wrapper. An activity diagram of this virtual sensor is shown in Figure 15. The feature vector consists of GPS coordinates of the current location and feature parameters in predefined format. Classification is based on a model created from training datasets, and code for the classification was developed by other project participants. Also a virtual sensor removes duplicates from the data, which are calculated from GPS coordinates after classification. If a classified data element is close enough and the same type as a previously classified element, it is removed from the stream. This reduces the amount of data streamed from the sensor. Proximity threshold of GPS coordinates can be configured in meters. A model's file name and parameter format can be given in a virtual sensor configuration file, thus allowing any model to be freely used with the GSN server. Output of the classification algorithm is the given coordinate data and type of the waypoint, either "Road" or "Pothole". This virtual sensor outputs the stream CLASSIFIED_DATA.

GPX virtual sensor starts a socket server which waits for client applications to connect using HTTP GET requests, performs buffering of received data from the stream CLASSIFIED_DATA, and then sends a GPX document constructed from data to the client application. An activity diagram of this virtual sensor is shown in Figure 16. Signal “HTTP GET received” is external to the virtual sensor and indicates that a request was received from client application. For each data requesting client application, there is a need for a separate GPX sensor running in the system. Client id for the GPX data is given in the individual configuration files and the sensor filters out other data elements from the stream based on the id. A small special GPX schema was created cooperatively by project participants to be used as output data presentation format. The GPX sensor receives data elements consisting of coordinates and label which is transformed into a GPX waypoint Java object. Waypoints are then buffered into a GPX document Java object which is then serialized using XStream library to an XML presentation. An XML Schema of GPX format is given in appendix 1. A GPX document contains a single gpx element. Its first three child elements form the header: GPX version, file creator and metadata. The next element is a mandatory waypoint giving current GPS location at the time of creation, for example giving the location of the vehicle. Also, optional waypoints can be used to describe other events at the current location. For waypoints, two predefined types are given, road or pothole. These types were chosen based on SDFA project requirement for finding road anomalies such as potholes. Waypoints can include optional metadata consisting of an URL or picture etc. After this, optional route elements can describe the travelled road. A route consists of name, description of route, optional metadata and waypoints. This is used if client applications request GPX data at intervals; the travelled route is then sent as a single GPX document.

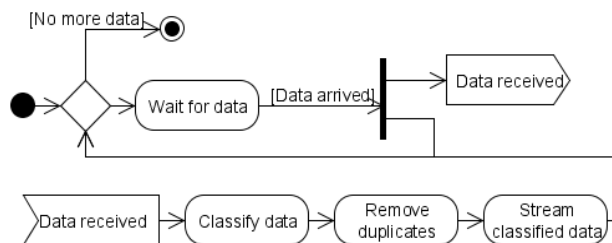


Figure 15. Activity diagram of Weka virtual sensor.

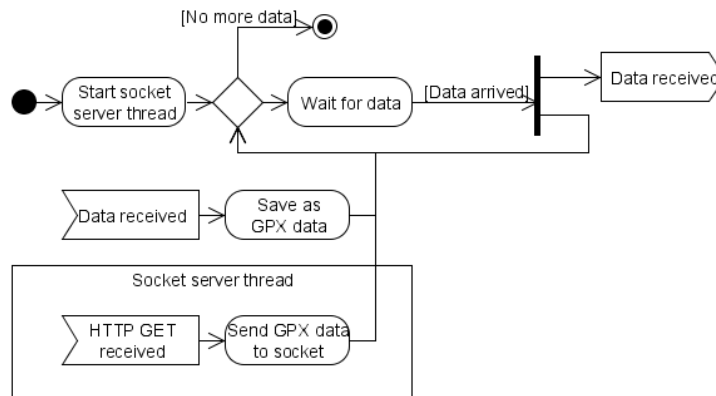


Figure 16. Activity diagram of GPX virtual sensor.

4.3.3. Configuration

GSN configuration is based on several configuration files in XML language. Physical environment of the server platform and Java virtual machine is described in build scripts. There is a general configuration file for the GSN server where GSN web client and the database system being used can be configured. Each wrapper is described in one properties file where the name of the corresponding Java class implementation is given. This class will be instantiated when the server detects a request to produce data from the wrapper. Virtual sensors each have one separate configuration file where initialization parameters, input and output data structures, names of input data streams, database SQL queries and data filtering based on SQL sentences can be given. The corresponding data source wrapper name and its parameters are given in the file also, or a special wrapper called local can be used to obtain data from another virtual sensor. This implements chaining between virtual sensors.

In the prototype application, three different data streams were identified. Data structures of these streams are described in chapter 3.4. There are two MATLAB virtual sensor configurations used in the server, one is for on-line and the other for off-line processing. They differ by which data source wrapper they use, one was using the socket server waiting for on-line data and another read data files directly from the server platform. In on-line configuration, an input socket port for incoming files needs to be given. In both configurations, the stream SYNCHRONIZED_DATA is defined as data source, and data buffer size and MATLAB executable input socket port are given. In Weka virtual sensor configuration, the input stream is PROCESSED_DATA and classification model name and its parameters are given. The parameters are probability threshold for classification result and detection threshold to distinguish GPS coordinates from each other. Weka wrapper is defined as data source and an output socket port of MATLAB executable is given for it. In GPX virtual sensor configuration, CLASSIFIED_DATA is defined as data source and a socket port for client application requests is given.

5. TESTING AND RESULTS

In this chapter, the performance and scalability tests of the GSN system implementation are presented. Then the implemented prototype application for the SDFFA project is evaluated. Performance and potential latencies in data processing components are tested. System implementation performance tests were conducted in a laboratory environment with randomly generated test data. Scalability of the system was tested by changing the number of sensors within one GSN platform. The prototype application was field tested with collected real data with off-line processing.

5.1. Testing environment and cases

Test cases are shown in Table 7. For test cases, the testing platform was the same as used in the tests in chapter 2.6. It was a normal desktop personal computer using Windows XP Professional Service Pack 3 as the operating system. Central processing unit was Intel Core 2,40GHz processor with 2GB of memory. Java version 1.6 was used with GSN version 0.95. MySQL version 14.14 with distribution 5.1.31 was used as the database system. HSQLDB version was 1.8.

As in chapter 2.6, there was a 20ms sleep period in the thread after sending each packet to allow other threads to execute in the server before sending the next packet. The created overhead can be ignored because it is consistent and similar in every test case. Data packets were created before streaming so that memory allocation time would not affect the results.

Table 7. Test cases

Test case 3: One GSN instance running and 50 wrappers simultaneously stream 10,000 data packets through database systems each to its own virtual sensor without any processing.
Test case 4: One GSN instance with different number of wrappers each streaming 10,000 data packets to their own virtual sensors without any processing.
Test case 5: Real system data consisting of GPS location and accelerometer data collected with mobile phone is streamed through prototype system.

5.2. Performance tests

In test case 3, there is one GSN instance running and 50 wrappers each simultaneously stream 10,000 data packets through database systems each to their own virtual sensors without any processing. This corresponds to 50 sensors or sensor nodes connected to a single server. Database systems were HSQLDB in memory and MySQL database for comparison. The tested packet sizes were 1KB, 10KB and 50KB. JVM memory size was set to both 128MB and 1GB.

Test results are shown in Table 8. With HSQLDB in memory, the delivery time of a 1MB packet is only twice the delivery time of a 1KB packet, and with MySQL a 50KB packet takes more than 20 times the amount of time of a 1KB packet. Delivery time per packet was less than one millisecond with HSQLDB, which is significantly faster than the results with MySQL. JVM memory size has no significant effect on the results with HSQLDB. The total execution time of some tests was more than 8 hours, streaming 500GB of data, meaning this can be considered a good enough

stress test. Therefore GSN with HSQLDB in memory handles large amounts of data with a number of data sources well.

Table 8. Performance test results

a) JVM memory size 128MB

Packet size	HSQLDB in mem	MySQL
1KB	0,5ms	2,8ms
10KB	0,5ms	30ms
50KB	0,7ms	65ms
1MB	0,9ms	-

b) JVM memory size 1GB

Packet size	HSQLDB in mem	MySQL
1KB	0,4ms	1,2ms
10KB	0,6ms	31ms
50KB	0,7ms	65ms
1MB	0,9ms	-

5.3. Scalability tests

In test case 4, One GSN instance with HSQLDB in memory was tested with 1, 30 and 50 wrappers each streaming 10,000 data packets through HSQLDB in memory to their own virtual sensors without any processing. Data packet sizes 1KB, 10KB, 50KB and 1MB were used. JVM memory size was set to 1GB. This test case uses combined results from test case 3.

The internal streaming time was measured, that is the time to send a packet from a wrapper to a virtual sensor. The results are shown in Table 9. This test case uses combined test results from test case 3. It seems that with the smaller packet sizes the number of wrappers does not matter, as the whole database transaction takes about the same time, which is about 5 minutes 10 seconds. In that sense the GSN server is scalable and performance does not vary depending of the number of data sources used. With larger packet sizes there are differences, however, and more variation can be seen in the results especially with 1MB packet size. Another result is that JVM memory size does not seem to affect performance; this can be seen when comparing results from test case 1 using 128MB of memory with one wrapper's results in this case.

Table 9. Scalability test results

Packet size	1 wrapper	30 wrappers	50 wrappers
1KB	31ms	1,0ms	0,4ms
10KB	31ms	1,1ms	0,6ms
50KB	30ms	1,3ms	0,7ms
1MB	39ms	3,9ms	0,9ms

5.4. Tests with real system data

In Test case 5, real system data consisting of GPS location and accelerometer data collected with mobile phone is streamed through the prototype system in off-line mode. On-line tests were not possible because when the implementation was finished

the vehicles used for data collection were no longer available. The vehicle was equipped with a video camera, Nokia N95 Mobile phone and SHAKE SK6 sensor. Figure 17 shows the video camera snapshot of the test system mounted in the car's instrument panel. There is no other use, in the prototype system, for the video camera data than labeling training data sets for the classification model used with Weka. Test data were collected by driving a vehicle around the city of Oulu, Finland. Example excerpts of collected data from files are shown in Table 10.



Figure 17. Test system mounted on the instrument panel in a vehicle.

Table 10. Excerpts of collected data from files

Example of collected acceleration data:			
% Acc_X(g)	Acc_Y(g)	Acc_Z(g)	Timestamp
0.163	0.965	-0.458	1241004549657
0.18	0.965	-0.458	1241004549657
0.114	1.113	-0.278	1241004549658
0.032	1.064	-0.376	1241004549659
0.13	1.08	-0.343	1241004549660
0.098	1.015	-0.343	1241004549660
-0.245	0.736	-0.13	1241004549718
-0.212	0.687	-0.18	1241004549719
-0.163	0.556	-0.147	1241004549720
-0.245	0.785	-0.114	1241004549720

Example of collected GPS location data:			
% Latitude_deg	Longitude_deg	Timestamp	
65.010585028096	25.470446405763003	1241006283068	
65.010647221794	25.470219340092	1241006284074	
65.010690472398	25.470071902471002	1241006285072	
65.010766244774	25.469800412730002	1241006287075	
65.010827097368	25.469528503894	1241006288075	
65.010870934705	25.469345359379002	1241006289076	
65.010912844205	25.469192976437	1241006290072	
65.0110138461	25.468869854192	1241006292076	
65.011073525228	25.468694588663002	1241006293070	

The number of data packets in each data processing component, total execution time per component and processing time per data buffers and packets are calculated. Results are shown in Table 11. There was 27% overhead caused by detailed logging in the results, which is subtracted when calculating the results. With MATLAB virtual sensor, data buffer size was set to 500ms of measurement data per buffer. In Weka virtual sensor, classification of the first packet took 240ms because the sensor and Weka Java objects were launched when the first data arrived. Results are given with the first packet and without it in parenthesis. For the GPX sensor, data throughput rate and execution time are dependent on the client application's data polling frequency which can be configured; thus it is not calculated here. In the server, JVM memory size was set to 1GB. Test data consists of two files, one for the GPS location data and the other for the accelerometer data. Data packet size created from the data is less than 100B, but an additional 100B of data from the feature vector is added to the data packet size after processing in MATLAB. Based on the results, processing in the MATLAB executable should be optimized further. One millisecond per packet is adequate result for Weka classification.

Table 11. Results of testing with real system data

a) Overhead caused by logging

Total execution time with logging	16869s
Total execution time without logging	13282s
Overhead caused by logging	27%

b) Time per data packet in components

	MATLAB	Weka	GPX
Number of data packets	97506	2075	882
Total execution time in component	12670s	2953ms (2713ms)	
Overhead of logging reduced	9976s	2325ms (2085ms)	
Time per data buffer	4,80s		
Time per data packet	102ms	1,12ms (1,00ms)	

5.5. Criteria for successful solution

Criteria for successful prototype application can be drawn from the requirements and then compared against testing results. All test cases must be passed and additional findings must not contradict requirements. Criteria are listed in Table 12.

Feasibility of the application was tested as test case 1 in chapter 2.6 and test case 5 in chapter 5.4. Expected incoming data rate was met and the application is capable of receiving on-line data without extensive latencies. System performance with large data packets and a large number of sources was tested in chapter 5.2 and considered adequate as time per data packet was less than one millisecond. Scalability was

tested in chapter 5.3 and it was found that with smaller packet sizes the system performance is not affected by the number of sensors streaming data in the server. Latencies of data processing were tested in chapter 5.4, and it was found that using external MATLAB executables is very slow compared to using Java class libraries.

Overall it can be said that GSN and the prototype system has met the given criteria and can thus be considered a feasible platform for sensor data processing in the context of the SDFA project. A problem was encountered with large latencies when using the external executable, but, as it is a prototype system under development used as a test-bench, this is currently tolerable.

Table 12. Criteria for successful prototype application

Application is generally feasible for intended purpose.	Tested in test case 1 and 5.
Overall system performance is adequate for large amounts of data and number of data sources.	Tested in test case 3.
Application is scalable. Data sources can be added to system with minimal effect to performance.	Tested in test case 4.
Latencies caused by internal data processing are tolerable.	Tested in test case 5.
Latencies caused by data processing in external systems are tolerable.	Tested in test case 5.

6. DISCUSSION

In this thesis, a prototype system for sensor network middleware was developed in the context of cooperative traffic. This middleware can be used as in-vehicle or remote data processing server. Mobile clients can utilize the system by sending sensor data on-line for real-time processing. The key idea in the design was to allow easy testing and development of different data processing algorithms with collected sensor data. Output of the data processing can be used to visualize the extracted features such as road conditions or anomalies in the road.

6.1. Analysis of system

In the S DFA project, requirements for the prototype system were quite straightforward but operating environment and overall system description with component details were not available at the time of implementation, as they are developed during the project. That is why a generic implementation of sensor network middleware is presented in this work. Comparison to other sensor network middleware featuring mobile clients, by name TIME and CarTel, is given in chapter 2.5. Communication and data routing issues are focused in those systems, and in GSN database optimization was not considered at all, but this was also a major feature in the compared systems. However, as complete implementation is not freely available for those systems, they cannot be considered for this project. In addition, several other sensor network platforms' architectural differences were compared against GSN. GSN offered most ready-made features and still has on-going development. IrisNet has support for XIndice database which could be useful for the S DFA project in future, but development for IrisNet has been stopped. It would have been desirable to test the performances of at least IrisNet and CarTel platforms for comparison with GSN.

Required real-time throughput criterion cannot yet be given for the in-vehicle system, as the type and number of sensors or client applications connected to the server is not known yet. There was an estimation calculated from the example data sets collected previously in the project, which was 500KB per second if there is one video camera sensor streaming data to the server with one accelerometer sensor and mobile phone built-in GPS receiver. Without the video camera, data throughput rate drops to less than 1KB/s. It was tested and found that, even with the slowest tested configuration, GSN was able to handle the required incoming throughput rate. As the number of sensors or client applications increase, the number of threads running simultaneously in the server increases, which will eventually affect the system performance. It might even be feasible to introduce more than one server platform running in the system to distribute the system load. An additional shared memory layer on top of GSN and between platforms could also quicken the transfer of data from server platform to another. This remains to be tested when a more detailed system architecture is available. Using distributed shared memory instead of a built-in database system in a single server was tested and it was found that there was no significant improvement with the database residing in memory.

One of the tasks of the S DFA project not implemented in this work is preprocessing or filtering of sensor data in data collection client software. Only scaling of accelerometer data is done, as it is possible to use external or built-in

accelerometer sensors which have different scales. Preprocessing or filtering could be done in the sensor node itself, for example in a mobile phone, or in the GSN wrapper connected to the sensor. This will also affect the data throughput rate of an on-line system as phases of processing are done outside the server platform. Mobile phone battery life could be affected by increased computation needs, but in the in-vehicle prototype system this should not be an issue, since the vehicle's power system could be used. Use of localized algorithms in the mobile phones is one issue which will be studied later in the S DFA project. This may also even lead to implementing virtual sensors or wrappers in mobile data collection software. A synchronization method is needed for more precise locating of anomalies extracted from the data. GPS gives an overall accuracy of ± 5 meters and the speed of the vehicle could also affect this. Preprocessing, filtering and synchronization methods are studied by research partners in the S DFA project.

With mobile phone data collecting clients, communication problems were sometimes experienced when running tests, especially when using WLAN to connect to the server as the coverage of WLAN base stations is coarse. GPRS worked better. In the prototype system, data collecting clients will stream data to the server at regular intervals, for example every five seconds. Studying these communication methods is out of the scope of this work, but this is being studied in other cooperative traffic projects. Also, finding a GPS satellite signal sometimes took a frustratingly long time. Relay nodes introduced in TIME architecture may be needed for data routing in the future, if distance to the server from the mobile clients grows. In the server platform, a PHP script running in the web server developed for forwarding data files to server from mobile phones worked without any problems. This component could be removed in the future when direct connections to the server are to be considered. A visualization client application was implemented with Javascript. It suffered from timing delays and a data overload condition was possible, as it has no data buffering capabilities and socket communication is blocked while the script is processing data. For example Java language implementation could be considered for this work. Also inaccuracies of GPS coordinate data can clearly be seen on the map created with OpenStreetMap, which may affect the interpretation of results.

6.2. Analysis of server implementation

Almost all the system requirements given in chapter 3.2 were met in the prototype system. The system is platform independent and scalable, has permanent data storage, and allows dynamic allocation of components in runtime. There is also a generic output interface and mobile phones were used as data sources when testing. External code was run in the system with MATLAB external executable and Weka classification. Concerning real-time performance, optimizations are needed with an external executable in the prototype system. In chapter 3.5, a number of ways to connect to MATLAB algorithms were presented but all are essentially the same when considering performance.

In chapter 3.3, operations for sensor data in the context of the S DFA project were given. The prototype system conforms to these three defined levels. Data fusion can be implemented in sensor nodes, wrappers or virtual sensors. Feature fusion level operations can be accomplished by developing application specific virtual sensors and by configuring data streams in the system. Decision fusion level operations can also be implemented in virtual sensors which can fuse data coming freely from other

systems in the vehicle or traffic. This data is outputted in generic data format and can be visualized for example in a web browser. Weka library was used for classification and was found very simple to use; basically the only inputs needed for classification were the training model used and the received data packets. MATLAB runtime engine is single-threaded, which is a significant disadvantage as it cannot have several threads used for communication and data processing simultaneously. As receiving several distinct data streams simultaneously was being blocked by this, it set limits to data fusion capabilities, which may introduce extra work to the prototype system in the future.

Using socket communication for communicating between server and external executables was straightforward and it allows external modules to reside anywhere in the network. However, in future systems, this may require security measures being taken into consideration. Working with data coming from mobile phones on-line, a wrapper running socket server was implemented based on file reading wrapper, which worked without problems. HTTP protocol is widely used, so the interface to a mobile phone can be considered generic, allowing other systems and applications to use it freely. However, a number of simultaneous data producing clients will affect overall system performance since a complete "route" with all the wrappers and sensors is required for each client, and starting of these components in GSN is handled internally. This will probably slow the system performance down and the effect of these issues to the quality of service for applications remains yet unknown. When outputting the data from GPX virtual sensor to client applications, some delays were experienced depending on the client side software, as scripts in the web server receiving the data did not have buffering and multithreading mechanisms and GPX sensor had to wait for all the previous data to be processed. As HTTP protocol was used here as well, the outputting of GPX data format was made as generic as possible.

There was no internal format developed in this work for collecting the sensor data. An internal data format could be based on XML as there are several ready-made software libraries to be used with it, which simplifies the needed implementation work. Using GPX format for the presentation of road anomalies and traveled route and XStream library for serialization was found simple and straightforward. As only numerical data was used in the prototype system, it was easily handled and streamed in the system. With external interfaces, data values were streamed as comma-separated strings, which was adequate as the data packets were quite small in size, about 200B. This simple way of presenting data in the system was found good enough for a prototype. Also it helps understand the prototype algorithms and code if data is in human readable form. The amount of data streaming in the system was reduced first in MATLAB virtual sensor, where for each buffer consisting even of hundreds of data packets sent to feature extraction only one data packet was received for classification. The amount of data was reduced further in classification where location based filtering was done after classification to remove duplicate data packets. This is helpful in the feature fusion level as data rate will not be very high. Unfortunately this has no effect with the latencies with external executables since every packet still has to be sent to feature extraction. The amount of data streamed in the system was reduced to one hundredth in two phases of processing.

Application knowledge was not used in this work for tailoring the services offered by the middleware, although this could be considered in the future. Efficient sharing of system resources in GSN was not tested in this work, since multiple data

collecting clients and different client applications were not available at this phase of the SDFFA project. Also GSN's internal fault tolerance mechanism was not tested in this work. HSQLDB database system should be used instead of MySQL, because it offers significant performance improvement especially with larger data packet sizes. There was a limitation of data packet size with MySQL as the maximum packet size was 65KB for all GSN data types; this may severely limit its use. GSN does not have methods for retrieving old previously collected data from database, so as this will probably be required in the future, this needs to be implemented as a wrapper. Also implementing this type of a wrapper is one consideration if there is more than one server or database platform in the system. Using other database systems with GSN would require database design and query optimization, which may require substantial implementation effort. However, as GSN is an open source project, all code can be modified if needed. The Java class interfaces for wrapper and virtual sensor were easy to use. Any Java software library can be easily integrated with the wrapper and virtual sensor classes, as is demonstrated in the Weka virtual sensor. Virtual sensor configuration files offered a good way for modifying virtual sensor behavior through free use of user parameters, meaning also that virtual sensor template configurations are easy to implement and use. Important parameters for sensors such as geo-spatial location, internal data buffer size and runtime priorities can be established in configuration. In the prototype system, there was no need to adjust the data buffer size parameter, since data flow was quite straightforward and no problems were experienced. Routing of data in GSN is handled by configuration of the virtual sensor's data inputs and outputs, which may not be the fastest solution but is easy to use. In the future, this may need to be changed. Fusing of data coming from separate data sources in the virtual sensor was not tested in this work, but this can be achieved simply by adding source streams to the configuration file. This can affect performance at the data fusion level but, on the higher levels, data rates should not be very high as demonstrated by the prototype application. Data stream configuration in runtime should have been easier: now changes to both the configuration file and code are required if data items in the stream are modified.

One consideration should be given to integration of previous and current work with sensor networks done in the Computer Engineering laboratory to the prototype system. For example in the UbiCity project there is on-going development of intelligent sensors, mobile devices and web interfaces [45]. With a number of ready-made hardware and software template wrappers coming with the GSN installation and the easiness of implementation of application specific wrappers this should not require substantial work.

Overall, GSN's architecture design and implementation are loosely-coupled, component-based and very flexible, allowing complete sensor networks to be set up and implemented with minimal effort. One purpose of the system is to be a test-bench for testing data processing algorithms, for which this implementation is feasible despite of currently introduced latencies.

6.3. Analysis of tests

There were feasibility, performance and scalability tests as well a field test with real data performed for the prototype system. The idea of the feasibility tests was to make sure that the system fitted with the given data throughput rate, which was calculated from sensor data sets collected earlier in the project. Also superiority of the two

selected object spaces implementations was determined. Results were very similar with smaller packet sizes but Fly was considerably faster with larger data packets. Even with the slowest tested combination, GSN had a faster throughput than required. However, the packet size largely affects the data throughput in tests as the time per packet is about the same with shared memory and database. One issue with all testing was that there was a need for a sleeping period in data streaming threads before sending the next data packets. This created an overhead time to measurements, but it was ignored when interpreting results because it was consistently present in every test. Without this delay, the GSN system crashed. Some overhead in tests was experienced by internal server mechanism to invoke Java objects only when they are requested. However, this was not significant in other tests as in the test with real data with the Weka classification library, where this overhead was measured to be 240ms. Also in tests with real data there was quite a large overhead caused by detailed logging of components to find out latencies in the server as logs were written to a file, but it was subtracted from the results.

Performance tests can be considered adequate since the largest amount of data tested was 500GB and testing lasted for more than 8 hours in some cases. This test tested only the streaming of packets from one wrapper to one virtual sensor, and there was no data processing included. The idea is that this represents the first step of a sensor node streaming data to the system from a physical sensor. So this data can be saved to the database and processed later, if current processing of other data consumes too many system resources. GSN was found to be able to handle a large amount of data without major difficulties. As expected, a database residing in memory was much faster than a database system on hard disk. However, it should be noted that streaming of a data packet size of 1MB takes only about double the time to handle a packet size of 1KB. Increasing the memory available for the JVM did not affect the results. These two matters could mean that system performance cannot be increased with software improvements if GSN system limits are met.

In scalability tests, it was measured whether the number of data streams, such as one per sensor, will affect the server performance. From the results it seems that, with the same packet size the number of wrappers does not really matter as the whole database transaction takes about the same time. This might be caused by JDBC or HSQLDB internal implementation. This was also noted in the feasibility tests. In that sense, the server is scalable and performance does not vary depending on the number of data sources used. Further tests could be conducted to find absolute limits, however it seems unlikely that more than 30 data sources per server platform are required in the SDFA context. Another scalability issue to be tested is the usage of several platforms simultaneously streaming data to each other. It would also be useful to know if using distributed shared memory in that case would improve performance.

Real data tests were conducted by first collecting GPS location and accelerometer sensor data with an in-vehicle system when driving around the city of Oulu. On-line testing of this was not possible since vehicles were not available for test drives when implementation of the system was finished; thus data processing tests are conducted in off-line mode. From the results it can be seen that using external MATLAB executables for data processing can create big delays in the system. This is, of course, dependent on the quality of the MATLAB code itself, and thus further tests are needed with different MATLAB code or algorithms. With multiple clients collecting and visualizing data in the server, using this manner for data processing

may not be feasible at all. Therefore it would be more feasible to use Java class libraries or MATLAB builder for Java which provides MATLAB functions as Java classes [46]. One scenario to reduce the latencies would be to use object spaces shared memory with master/worker pattern to deliver the data buffers for processing to a number of server or MATLAB instances running in the network, if the usage of Java language or optimization methods do not reduce them enough. With the GPX sensor, the data throughput rate is variable based on configuration parameter of polling frequency, for example 100ms used in early testing of the visualization client application. During this time about 60 data packets as 12KB of data is streamed to the sensor, an amount which the sensor can handle easily. The data packet size of 200 bytes can be considered very small and data was only numerical, but if for example video camera data is appended to packets it can cause further delays in the processing. The amount of data streamed in the server reduces radically when the processing advances, which is useful for the data fusion model in the SDFa project as the amount of data does not form bottlenecks in the higher levels of the model. When collecting the data sets, it should be noted that they were collected using different vehicles, so even if data was collected from the same location, it may not be reusable or the results may not compare effectively. It seems that the roads in the municipality of Oulu may be in too good a condition for effective measurement of road anomalies as for example bumps where quite hard to find.

6.4. Future work

The middleware and prototype system developed in this thesis will be put to use in the SDFa project's next phase. Goals for the next phase are not yet decided but some items can already be found. The system should manage several simultaneous mobile clients sending collected data to server and visualize it in real-time. This will require faster processing of data elements, which needs to be implemented into the server; the MATLAB implementation currently used is unacceptably slow for real-time applications. Possible solutions for this problem were found, using Java language or improving the performance of MATLAB code. This issue needs to be studied separately perhaps with a real scenario derived from the tasks of the project. Interfaces to the instrumented vehicle's sensors or roadside infrastructure sensors should be implemented to receive multimodal sensor data which will also require implementation of synchronization and fusing algorithms in the server. Another interface is needed for retrieval of old data stored in the database, so that later in the project queries for data can be made online or offline based on event type, location, route, road conditions, time or any other features. Old data could also be fused with real-time data from on-going measurement. These features open interesting scenarios for further study. Collected data sets could be merged or compared for example for during different times of year or different weather conditions for road condition analysis, traffic flow analysis and even real-time driver behavior analysis. As GSN allows easy integration of any kind of system, interfaces to existing sensor networks systems developed in the Department of Electrical Engineering could also be considered.

Development of internal data format and data representation is one thing which should be considered. XML offers a good way to structure data and types into distinct representation when multimodal sensor data is introduced to the system. This

task goes in parallel with studying and testing of different database systems for the system, especially XIndex as it is optimized for XML documents.

Security in sensor networks has not been studied in this thesis, but when accessing an instrumented vehicle's sensors and when collecting data visualizing multiple clients this is an important issue. Also with the possible integration of roadside infrastructure data and sensors, security and privacy are big concern. The problem is that security measures are largely application specific, and this needs to be studied further [7].

7. CONCLUSION

Intelligent transport systems are aimed at using advances in information, communication and sensor technologies to improve traffic. Major problems faced today are road safety, fuel consumption and emissions from traffic. Sensor networks offer a suitable platform for observing phenomena and events in traffic, as all types of sensors can be freely deployed into vehicles and to road infrastructure. Location- and time-based data gives valuable information regarding for example road conditions or traffic flow.

This thesis presented a prototype system for sensor data processing and fusion. Several ready-made sensor network middleware implementations were evaluated and the most feasible one, the Global Sensor Network, was selected as a platform for the system. GSN is a platform independent open source project and has most ready-made features and interfaces to sensors. Its functionality is changeable in runtime and external data processing algorithms can be run in the system, which are important requirements in the SDFa project. Data gathering, synchronization, feature extraction, classification and data format transformation components were implemented. These components allow freely all kinds of data collection clients to connect to the system and developed generic output interface does not limit the usage of results. A distributed shared memory component was tested on top the middleware to determine if it should be used instead of built-in database systems in the prototype for increasing performance, but no improvement was discovered.

In the SDFa project, mobile phones with GPS receivers and accelerometer sensors are used to collect and send data to the server where application specific features are extracted from the data. The generated results can then be visualized for example in a web browser. The first result in the SDFa project was selected to be recognition of road anomalies such as potholes based on accelerometer sensor data. Feasibility, performance and scalability of the system for real-time data processing was tested, and it was found that, overall, the implemented middleware is feasible for this kind of application. Several issues were also found for future development of the system. Security and privacy issues in the prototype system were not considered in this work. Fusing old measurement data with the data collected even in real-time opens interesting scenarios for further study.

8. REFERENCES

- [1] Laitinen J. & Hakala H. (2008) Strategic Reseach Agenda. Cooperative Traffic ICT, TIVIT Ltd, 29 p.
- [2] Riekki J., Markkula J. & Perttunen M. (2008) Sensor Data Fusion and Applications Project Plan. Cooperative Traffic ICT, TIVIT Ltd, 26 p.
- [3] Akyildiz I., Su W., Sankarasubramaniam Y. & Cayirci E. (2002) A survey on sensor networks. *IEEE Communications Magazine*, Vol. 40, 8, pp. 102-114.
- [4] List of Sensors. (Read 31.08.2009) URL: http://en.wikipedia.org/wiki/List_of_sensors.
- [5] Hill J., Szewczyk R., Woo A., Hollar S., Culler D. & Pister K. (2000) System architecture directions for networked sensors. *ACM SIGOPS Operating Systems Review*, Vol. 34, 5, pp. 93-104.
- [6] Tubaishat M. & Madria S. (2003) Sensor networks: an overview. *IEEE Potentials*, Vol. 22, 2, pp. 20-23.
- [7] Blomqvist E. (2003) Security in sensor networks. Master's Thesis. Helsinki University of Technology, Control Engineering Laboratory, Espoo, Finland, 90 p.
- [8] Bachrach J. & Taylor C. (2005) Localization in Sensor Networks. In: Stojmenović I. (edited) *Handbook of Sensor Networks: Algorithms and Architectures*. John Wiley & Sons, New York, NY., pp. 277-311.
- [9] Pottie G. (1998) Wireless sensor networks. In: *IEEE Information Theory Workshop*, June 22-26, Killarney, Ireland, pp. 139-140.
- [10] Kahn J., Katz R. & Pister K. (1999) Next century challenges: mobile networking for 'Smart Dust'. In: *MobiCom '99*, August 15-20, Seattle, Washington, Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, pp. 271-278, New York, NY.
- [11] Dyo V. (2005) Middleware design for integration of sensor network and mobile devices. In: *2nd International Doctoral Symposium on Middleware*, Grenoble, France, Proceedings of the 2nd international doctoral symposium on Middleware, pp. 1-5, New York, NY.
- [12] Yu Y., Krishnamachari B. & Prasanna V. (2003) Issues in Designing Middleware for Wireless Sensor Networks. *IEEE Network*, Vol. 18, pp. 15-21.
- [13] Modukuri K. & Hariri S. (2008) Autonomous Middleware Framework for Sensor Networks. *Journal of Pervasive Computing and Communications*, Vol. 1, 4, pp. 337-345.

- [14] Manasseh C. & Sengupta R. (2008) Middleware for Cooperative Vehicle-Infrastructure Systems. California PATH program, Research Report, University of California, Berkeley, CA., 23 p.
- [15] Bacon J., Beresford A., Evans D., Ingram D., Trigoni N., Guitton A. & Skordylis A. (2008) TIME: An Open Platform for Capturing, Processing and Delivering Transport-Related Data. In: 5th IEEE Consumer Communications and Networking Conference, January 10-12, Las Vegas, NV, Proceedings of IEEE Consumer Communications and Networking Conference, pp. 687-691.
- [16] Hull B., Bychkovsky V., Zhang Y., Chen K., Goraczko M., Miu A., Shih E., Balakrishnan H. & Madden S. (2006) CarTel: a distributed mobile sensor computing system. In: The 4th ACM Conference on Embedded Networked Sensor Systems, October 31 - November 03, Boulder, CO, Proceedings of the 4th international conference on Embedded networked sensor systems, pp. 125-138, New York, NY.
- [17] GSN Team. (2008) Book of GSN. Ecole Polytechnique Federale de Lausanne, User Manual, Lausanne, Switzerland, 57 p.
- [18] Aberer K., Hauswirth M. & Salehi A. (2006) The Global Sensor Networks: middleware for efficient and flexible deployment and interconnection of sensor networks. Ecole Polytechnique Federale de Lausanne, Technical Report, Lausanne, Switzerland, 26 p.
- [19] Gibbons P., Karp B., Ke Y., Nath S. & Seshan S. (2003) IrisNet: An Architecture for a Worldwide Sensor Web. IEEE Pervasive Computing, Vol. 2, 4, pp. 22-33.
- [20] Shneidman J., Pietzuch P., Ledlie J., Roussopoulos M., Seltzer M. & Welsh M. (2004) Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Harvard University, Technical Report TR-21-04, Cambridge, MA., 12 p.
- [21] Franklin M. (2005) HiFi: Network-centric query processing in the physical world. In: 21st International Conference on Data Engineering, April 5-8, Tokyo, Japan, Invited talk, 43 p.
- [22] Tanenbaum A. & van Steen M. (2002) Distributed systems: principles and paradigms. Prentice Hall, Upper Saddle River, NJ., 803 p.
- [23] Schmidt D. (Read 05.05.2009) The ADAPTIVE Communication Environment. URL: <http://www.cs.wustl.edu/~schmidt/ACE.html>.
- [24] Rivera R., Dawes B. & Abrahams D. (Read 05.05.2009) Boost C++ Libraries. URL: <http://www.boost.org/>.
- [25] Intel Corporation Ltd. (Read 05.05.2009) Intel Threading Building Blocks 2.2 for Open Source. URL: <http://www.threadingbuildingblocks.org/>.

- [26] Chang W., Bolyard N. & Mielczarek T. (Read 05.05.2009) Netscape Portable Runtime. URL: <http://www.mozilla.org/projects/nspr/>.
- [27] Gelernter D. (1985) Generative communication in Linda. ACM Transactions on Programming Languages and Systems, Vol. 7, 1, pp. 80-112.
- [28] Angerer B. (Read 05.05.2009) Space-Based Programming. URL: http://onjava.com/pub/a/onjava/2003/03/19/java_spaces.html.
- [29] Polze A. (1993) The Object Space Approach: Decoupled Communication in C++. In: Ege R., Singh M. & Meyer B. (Editors) TOOLS 1993: 11th International Conference on Technology of Object-Oriented Languages and Systems, pp. 195-204, Prentice Hall, Upper Saddle River, NJ.
- [30] Object Spaces. (Read 09.10.2009) URL: http://en.wikipedia.org/wiki/Tuple_space#Object_Spaces.
- [31] Zink Digital Ltd. (Read 06.05.2009) Fly Object Space. URL: <http://www.flyobjectspace.com/>.
- [32] McLaughry S. & Wycko P. (1999) T Spaces: The Next Wave. In: 32nd Annual Hawaii International Conference on System Sciences, January 5-8, Maui, HI, Vol. 8, pp. 8037, Washington, DC.
- [33] Aberer K., Hauswirth M. & Salehi A. (2006) A middleware for fast and flexible sensor network deployment. In: 32nd international conference on Very large databases, September 12-15, Seoul, Korea, Proceedings of the 32nd international conference on Very large databases, pp. 1199-1202.
- [34] Mazhelis O. (2009) Sensor Data Fusion and Applications. State of the Art Report, Cooperative Traffic ICT, TIVIT Ltd, 36 p.
- [35] Walsh N. (Read 09.10.2009) A Technical Introduction to XML. URL: <http://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN58>.
- [36] Foster D. (Read 06.05.2009) GPX: the GPS Exchange Format. URL: <http://www.topografix.com/gpx.asp>.
- [37] Coast S. (Read 05.05.2009) OpenStreetMap. URL: <http://www.openstreetmap.org/>.
- [38] The World Wide Web Consortium. (Read 9.10.2009) XML Path Language (XPath) 2.0. URL: <http://www.w3.org/TR/xpath20/>.
- [39] The World Wide Web Consortium. (Read 9.10.2009) XQuery 1.0: An XML Query Language. URL: <http://www.w3.org/TR/xquery/>.
- [40] Walnes J. (Read 09.10.2009) XStream. URL: <http://xstream.codehaus.org/>.

- [41] The Apache Software Foundation. (Read 9.10.2009) Apache XIndice. URL: <http://xml.apache.org/xindice/>.
- [42] The MathWorks Inc. (Read 06.05.2009) MATLAB - The Language of Technical Computing. URL: <http://www.mathworks.com/products/matlab/>.
- [43] SAMH Engineering Services (2007) SHAKE Sensing Hardware Accessory for Kinaesthetic Expression Model SK6. Dublin, Ireland.
- [44] Witten I. & Frank E. (2005) Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco, CA., 525 p.
- [45] Ojala T. (Read 31.08.2009) UbiCity. URL: <http://www.mediateam oulu.fi/projects/ubicity/>.
- [46] The MathWorks Inc. (Read 09.10.2009) MATLAB Builder JA. URL: <http://www.mathworks.com/products/javabuilder/>.

9. APPENDICES

Appendix 1. XML schema of GPX format.

```

<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="latitude">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="-90.0" />
      <xs:maxInclusive value="90.0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="longitude">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="-180.0" />
      <xs:maxInclusive value="180.0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="wptType">
    <xs:all>
      <xs:element name="lat" type="latitude" />
      <xs:element name="lon" type="longitude" />
      <xs:element name="ele" type="xs:integer" />
      <xs:element name="type" type="xs:string" />
      <xs:element minOccurs="0" name="desc" type="xs:string" />
      <xs:element minOccurs="0" name="link" type="xs:string" />
    </xs:all>
  </xs:complexType>
  <xs:complexType name="rteType">
    <xs:all>
      <xs:element name="name" type="xs:string" />
      <xs:element name="desc" type="xs:string" />
      <xs:element minOccurs="0" name="extensions" />
      <xs:element name="rtept">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="wpt" type="wptType" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="gpxType">
    <xs:sequence>
      <xs:element name="version" type="xs:decimal" />
      <xs:element name="creator" type="xs:string" />
      <xs:element minOccurs="0" name="metadata" type="xs:string" />
      <xs:element maxOccurs="unbounded" name="wpt" type="wptType" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="rte" type="rteType" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="gpx" type="gpxType" />
</xs:schema>

```