IST-004400

Vertical System Functions

EW Identifier: EW-T313-UCAM-001-06
Date: 06/09/2005

Author(s) and company: Marcelo Pias and George Coulouris (University of Cambridge); Pedro Jose Marron and Daniel Minder (University of Stuttgart); Nirvana Meratnia, Maria Lijding and Paul Havinga (University of Twente); Sebnem Baydere and Erdal Cayirci (Yeditepe University); Chiara Petrioli (University of Rome La Sapienza)

Work package/task: WP 3 / T 3.1.3
Document status: final
Confidentiality: public
Keywords: Vertical system functions, context and location management, sensor data consistency, communication, distributed storage and search, data aggregation, resource management and security, privacy and trust

Abstract: This study identifies the roles and effects of *vertical system functions (VFs)* in the context of cooperating objects. It provides an up to date discussion of proposed VFs and points out avenues for future research.

Table of Contents

1 Executive Summary

This study (WP3.1.3) is the third in a series of four other studies intended to thoroughly survey the research literature and identify the relevant state of the art in the context of distributed cooperating objects (COs) and wireless sensor networks (WSNs). In particular, it discusses the roles and effects of *vertical system functions (VFs)*, which provide the required system functionality to address the needs of the CO applications.

The objective of the first study (WP3.1.1) is to identify applications and application scenarios and their requirements. The second study (WP3.1.2) aims at studying the state of the art of paradigms and algorithms used to address the system characteristics and requirements of the CO application domains. The fourth study (WP 3.1.4) describes the set of programming models, paradigms and system architectures that facilitate the integration of the enabling technologies identified in this document (as well as in the study WP3.1.2) to the application requirements discussed in the study WP3.1.1.

The most important results of these studies will be used as an input to the preparation of a research roadmap (WP3.3). This task has the goal of identifying relevant open issues to the future development of cooperating objects.

2 Introduction

In the scope of the studies, a CO is defined as a collection of sensors, actuators, controllers or other COs that communicate with each other to achieve, more or less autonomously, a common goal. This recursive definition accounts for the cases where groups of cooperating objects can be regarded as a single CO. Such arrangements enable them to combine hardware and software components to support advanced sensor applications. Thus, organising such components into a framework that can cope with the inherent complexity of the overall CO system will be an important exercise for developers.

The list below of key points in the design space of a CO is based on the requirements set for the wireless sensor platforms developed independently at UCLA and UC Berkeley [37, 48]:

- *Small physical size:* reducing physical size has always been one of the key design issues. For instance, some applications will need more powerful CO units than others. Hence, COs are likely to be heterogeneous devices in terms of processing, communication and sensing capabilities. Such a diversity poses the challenge to find the right balance between the physical device size and the minimal set of required hardware subsystems to be implemented in a CO.
- *Low power consumption:* energy constrains processing, lifetime and interconnect capability of the basic CO device. The system should make efficient use of the resources striving to minimise the overall power consumption. As a result, this will increase the CO active time without battery recharging, which is an issue for a set of applications such as large-scale forest fire monitoring.
- *Concurrency-intensive operation:* data will be frequently gathered from local sensors or received from other COs, then they are processed through filtering/aggregation, and sent to

other COs through the network. These tasks should be carried out simultaneously in order to achieve the target sensing and actuation goals at the highest performance. Therefore, strict resource sharing and task scheduling are key design issues.

- *Diversity in design and usage:* networked sensor devices will tend to be application-domain specific providing only the necessary hardware support. For instance, sensors and actuators built for medical health-care applications exhibit more complexity when compared to simpler sensors used in environmental monitoring. Therefore, the hardware and software framework of a CO should facilitate trade-offs among component reuse, cost, and efficiency.

- *Robust operation:* CO devices will be numerous and deployed over a large environment. The individual devices should be carefully designed having reliability as one of the key properties. Although device failures can be overcome with distributed fail-over techniques, this approach should be avoided whenever it is possible because of the communication costs incurred. COs need autonomous management abilities to self-test, self-calibrate and self-repair as recently advocated in [61, 56].

- *Security, privacy and trust:* each CO should have sufficient security mechanisms in place to prevent and counter-measure unauthorised access, denial-of-service attacks, and unintentional damage of the information locally stored. These mechanisms need to be considered in the design phase in order to make them pervasive throughout the system.

- *Compatibility:* the cost to develop software components dominates the cost of the overall system. It is important to be able to reuse code developed for other systems.

- *Flexibility:* the CO system will evolve over time both in terms of hardware and software. Support for this growth can be introduced in the system through hardware programmability and reconfiguration by using programmable processors and FPGA-based platforms. Also, software modularity introduces flexibility and should be prioritised during the system design.

The question that arises is whether any available real-time operating system (RTOS) suits this list of requirements. Some researchers believed that traditional RTOS was unsuitable and therefore they designed alternatives such as the micro-threaded operating system TinyOS [48]. Whether this new OS is entirely appropriate for a CO system remains to be investigated. As advanced sensor applications emerge, it is likely that the emphasis will be put on the design of more complex micro-electromechanical (MEMS) sensors as already suggested in [129]. To support this sensor development, a CO system will be built from powerful resources that need more implemented functionality and efficient interactions than the ones currently offered by TinyOS. Readers are referred to the study WP3.1.4 for a comprehensive discussion on CO systems and architectures.

The rest of the document is organised as follows: Section 3 defines a *vertical system function* in the context of cooperating objects. Section 4 briefly discusses the characteristics and requirements of the CO applications studied in WP3.1.1 (Applications and Application Scenarios). Different types of VFs to address these applications needs are then discussed including context and location management, data consistency, communication and security, just to name a few. Section 5 gives a summary of the VFs discussed in the document and concludes with some final remarks.

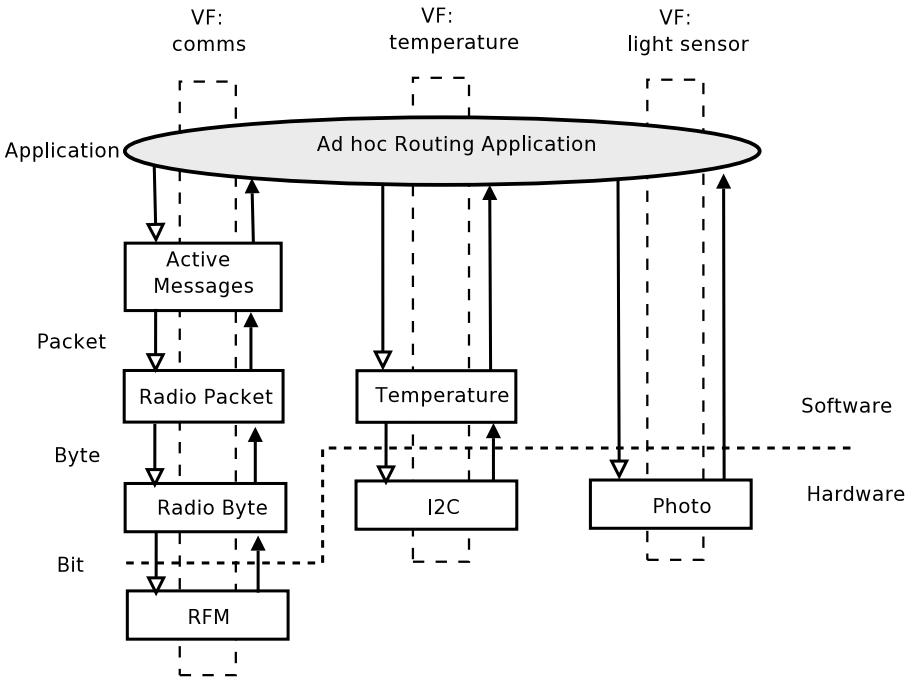


Figure 1: Example of Vertical Functions: temperature and light sensing application

3 Vertical System Function (VF)

The current operating systems proposed for WSNs and COs cannot offer all the required functionality to the applications. Thus, a vertical system function is defined in this study as the *functionality that addresses the needs of applications in specific domains and in some cases a VF also offers minimal essential functionality that is missing from available RTOS.*

Figure 1 introduces a simplified architectural view of an application-example originally discussed in [48]. The goal is to monitor the temperature and light conditions of an area and periodically transmit their measurements to a central base station. In this example, there are three VFs that applications may 'invoke': the communication subsystem (core system function), the light and temperature sensors (application specific functions). Each VF is represented in the diagram by a vertical stack of components. We envisage that *standardised APIs* will create mechanisms for linking applications to vertical functions.

A component, labelled box in the diagram, is defined as a self-contained unit of code that encapsulates its implementation and interacts with its environment by means of well-defined interfaces. Such an interaction can be achieved through a *software wiring* process that interconnections the output ports of a component to the input ports of another one. The composition of components can be represented as a system configuration graph, where components are vertices and their interconnects form the graph edges [48, 24, 60].

The essence of component interactions is twofold. The *control* interaction handles the requests for data to lower-level components (top-down). In contrast, the *data* interaction deals with the requested

data by creating a bottom-up data flow path between components. This is shown in the figure as vertical lines with arrows.

The component wiring process will facilitate the synthesis of individual components into a larger-scale system. Components may be grouped together to form more concise modules [24, 81]. The CO system may be arranged in several *levels of abstraction* from the most abstract (closest to the application) down to the most concrete (closest to the hardware devices). Abstraction allows better software structuring for clarity and reuse. Although layers are natural consequences of arranging components in the system, they are not a design requirement.

The implementation of a VF will follow a layered approach when the system architecture is designed in such a way. Although layered architectures facilitate key design properties such as flexibility and abstraction levels, they require attention to conformance and can severely impact performance in systems that lack hardware resources, for instance, low-power wireless sensor platforms.

The question to address in this case is how we map a general layered CO system architecture to a resource-constrained hardware platform without sacrificing overall performance. Ideally we seek to minimise the overhead imposed on the system by the various components and levels of abstraction that are on the way before the hardware components can be accessed. It is important to note that critical realtime applications such as control of industrial plants may not tolerate high system response time.

To achieve an energy-efficient design, the traditional strict modularisation or layering is not appropriate. In wireless sensor networks, for instance, monolithic design of communication software is used to reach the required energy-efficiency needs. However such a design choice makes system development and management very difficult.

An approach that can be used to improve performance in systems low in resources is to implement the VF using *cross-layer* interactions where the software components do not necessarily interact with components immediately above or below in the abstraction level [75]. For instance, most abstract components (closest to the application) may bypass other components and interact directly with the most concrete components (closest to the hardware). The cross-layer design explored in [40, 28] have shown promising results in reducing power consumption.

The architectural framework introduced in Figure 1 refers to a standalone cooperating object. We believe that in practice there will be collections of COs in constant interaction to accomplish a pre-assigned goal. In some application scenarios, VFs will be implemented through a chain of software components that may or may not be within the operational boundaries of a single CO but rather distributed in the network. For example, a VF that is responsible for collecting temperature readings of rooms in an office building needs distributed coordination among COs located in each room in order to implement the intended functionality.

4 Types of Vertical System Functions

The set of characteristics exhibited in CO applications are more diverse than the ones found in applications of traditional wireless and wired networks. Critical factors impact the architectural and protocol design of such applications. These factors also introduce some strict constraints.

The study WP3.1.1 (Applications and Application Scenarios) examined this set of characteristics and requirements. Below we briefly review the relevant ones and discuss the most suitable VFs

to address them. The reader should refer to the document WP3.1.1 for a comprehensive study on selected cooperating objects applications.

Network topology: in a CO application, nodes may communicate directly provided they are geographically close to each other. Such a communication can be established in a *single hop* network topology. When nodes are located far from each other, they need to rely on third nodes to forward their data packets requiring, therefore, a *multi-hop* sensor network. VFs which may offer direct support to this characteristic are *communication* (Section 4.3) and *distributed storage and data search* (Section 4.5).

Scalability: the number of COs that may support an application can vary depending on the environment where it is deployed and on its task. This property is rather important in outdoor applications and it is often the design issue of techniques for *distributed storage and search*. As the system scales up, consistency of the data gathered from multiple sources should be addressed in an efficient manner. *Data consistency* functionality is extensively discussed in Section 4.2.

Fault tolerance: it is highly possible that some COs may fail during the operation of the network for various reasons including battery discharge and harsh environmental operation conditions. The *data consistency* VF tackles some of the issues associated with node failures. In addition, the *communication* VF should provide the data communication resilience required by the applications. Fault tolerance is also closely related to the *security* VF (Section 4.4) since node failures may be caused by attackers.

Localisation: there exist several CO applications for target tracking and physical event detection including intrusion and forest fire that require node and/or target localisation. GPS may be the natural choice for computing a node's location. These devices, however, do not work in indoor areas and are still of high cost for low-power sensor nodes. The *context and location management* VF (Section 4.1) surveys the most recent research advances in this area.

Time synchronisation: applications need to establish a common sense of time among the cooperating objects participating in their sensing and actuation goals. Such a functionality can be offered through a *time synchronisation* VF (Section 4.8).

Security: the CO system may be threatened by unauthorised users trying to access the network. Also, there are security risks on the physical layer of the network. For example, jamming signal may corrupt the radio communication between the entities in the mission-critical networks. In a CO, security is pervasive and must be integrated into every system component to achieve a secure system. Thus, the security functionality is likely to be offered at different levels and not exclusively by the *security, privacy and trust* VF.

Data traffic characteristics: The amount of data travelling inside the network determines the traffic characteristics of an application. In a particular application, the data transferred among nodes may be limited to a few bytes for simple measurements whereas heavy video-audio traffic may be conveyed in another application scenario. At least three VFs can provide the adequate support for different types of traffic. The *communication* and *distributed storage and data*

search for high-level dissemination of sensor data offer mechanisms for transferring the data of interest in the network. In addition, the *data aggregation* (Section 4.6) provides an energy-efficient optimisation tool for various types of data traffic.

Networking infrastructure: CO networks can be *infrastructured* or *infrastructureless (ad hoc)*. Even in some applications, the data can be collected by some mobile nodes when passing by the source nodes. This is an important characteristic that determines the type of system approach used (with or without supporting infrastructure) in the majority of VFs surveyed in this document including the techniques for *context and location management* VF and MAC layer/routing protocols in the *communication* VF.

Mobility: in some applications, all physical components of the system may be static whereas in others, the architecture may contain mobile nodes. Applications which can benefit from autonomous robots for actuation may require special assistance for mobility. Adequate support for medium and high mobility in multi-hop networks is still an open issue that should be addressed in the implementation of future VFs such as *context and location management*, *communication* and many others.

Node heterogeneity: the majority of CO applications include nodes that have distinct hardware and software technical specifications. In a precision agriculture application, for instance, there may exist various types of sensors such as biological and chemical. Energy may be constrained in some of the nodes. Thus, the *data search VF mechanism* needs to be energy efficient. Also, data will originate from different sensor nodes so that adequate schemes for ensuring consistency of heterogeneous sensor data must be used. This can be offered to the CO application through the *data consistency* and *aggregation* VFs.

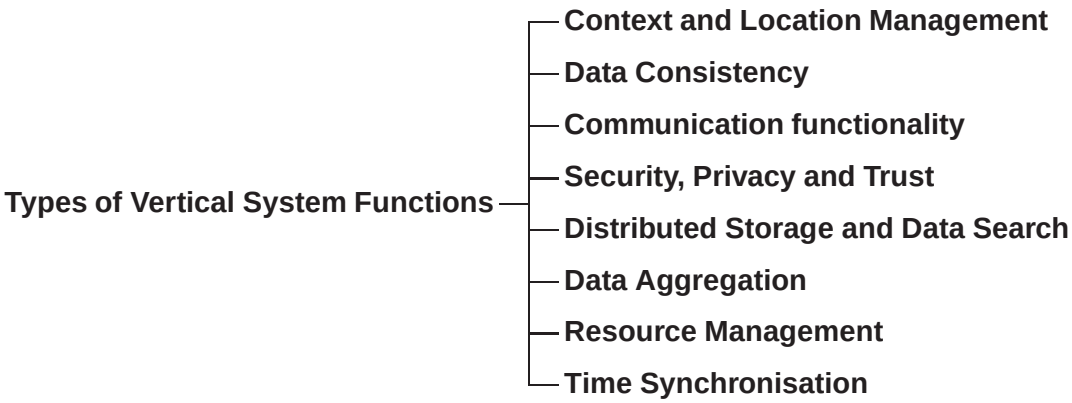
Power Awareness: power consumption is one of the performance metrics and limiting factors almost in any CO application. Systems require prolonged network lifetime. Thus, efficient power consumption strategies must be developed. Power-aware communication protocols are supported in the *communication* VF. Also, as more complex sensors are designed - for instance in healthcare applications - there is a growing need for tighter control on nodes' resources to save energy. The *resource management* VF (Section 4.7) deals with the power consumption issue.

Real-time: the system delay requirements are very stringent in real-time applications. The broad meaning of delay in this context comprises the system data processing and network delay. For instance, in a industrial automation scenario actuation signals are required in real-time. VFs are capable of offering the required functionality to applications through cross-layer system approaches which can significantly reduce the overall system delay. Thus, resource monitoring and system adaptation achieved with cross-layer component-based interactions are important schemes that should be made available to the applications. The *resource management* VF can offer the necessary functionality to real-time applications.

Reliability: end-to-end reliability guarantees that the transmitted data is properly received by the receiving-end. In some applications end-to-end reliability may be a dominating performance metric; whereas it may not be important for others. In security and surveillance applications in

particular, guaranteed end-to-end delivery is of high importance. Such non-functional requirement can be fulfilled through the coordination and interaction of various VFs including the *data consistency*, *communication* and *security, privacy and trust*.

We will follow the order below to discuss the relevant vertical system functions in the sections that follow:



4.1 VF: Context and Location Management

Distributed Cooperating Object systems are designed to measure properties of the physical world. They are, therefore, suitable for gathering the context of an entity, which is the information that can be used to characterise its situation. Individuals, locations, or any relevant object can be such entities [13].

Most researchers argue that context information has three major attributes through which it can be accessed: the identity of the entities, their location, and the time at which the information has been gathered. Since a reasonable amount of data is collected in large systems, context management systems are needed to handle them. Such systems can separate applications from the process of data processing and context fusion. Additionally, this allows a number of cooperating objects to share the gathered context.

Changes in context may also trigger actions to influence the monitored entity. Specialised actuators, for instance, may be programmed to control pipe valves when a fluid pressure reaches a certain threshold. To achieve precise actuation and detailed analysis of collected measurement data, however, the spatial distribution of sensors needs to be known. Thus, determining the *location* of cooperating objects is a requirement of a large number of applications, including monitoring of habitat, urban and indoor areas.

This section explores the design space of vertical functions for context and location management.

4.1.1 Context Management

Classical context management systems use infrastructure-based directories to store the information, for example Aura [39] and Nexus [49]. This has the advantage that the device knows where the information can be queried from. On the other hand, to avoid bottlenecks the data has to be structured

in some way and distributed among several devices. With — usually small — cooperating objects, the distribution of the context is system-driven. Also, the context often has only limited spatial relevance which is reflected by the typical wireless communication which is also spatially restricted. It is reasonable to store the context at or near the location where it is generated.

To start with the gathering of the sensor data, MiLAN [44] allows for the decoupling of application and data gathering. The user has to provide a specification of which sensors, or a set of sensors, can provide what QoS for which data. Additionally, the user specifies which data with what QoS are needed when the application is in various states. Then, MiLAN makes sure that the needed data is available. The actual source of the data is transparent to the application.

The management of data in a single device is the focus of the MobileMan project [27]. It creates a cross-layer architecture for the network protocol stack in mobile ad-hoc networks. MobileMan primarily aims at developing a network protocol stack that is optimised with cross-layer interactions. A more general approach is followed in TinyCubus [75]. Cross-layer data such as context information is stored in a State Repository. The Cross-Layer Framework ensures that data needed by an application component is provided by another.

With TinyDB [71], a whole network can be regarded as a database. It mainly focuses on external queries since they are parsed and optimised externally, but a fixed set of pre-parsed and pre-optimised queries could also be used inside the network. Such database approaches thus provide an easy way to get context data when the storage location is unknown.

In geographic hash tables [97], the storage location can be calculated from the index key. Each node has to have a geographic location, and the data is stored in the node geographically nearest to the hash of its key. Thus, context data can be queried directly from the storage node.

The cooperating objects themselves belong to the context as well. [135] presents a self-monitoring system for sensor networks. It continuously computes aggregates (sum, average, count) of network properties like loss rates, energy levels, etc., and disseminates them in the network in an energy-efficient manner. All objects can access the system context of the network.

We have shown the different parts of context management, starting with the gathering of sensor data. This data is made available to all components of an application and to other nodes in the network in different ways. Several approaches exist for these purposes, but to the best of our knowledge none covers the complete context management area. Therefore, more research is needed here.

4.1.2 Context-aware Applications

Applications are called context-aware if they adapt their behaviour based on the context. Several forms of this adaptation exist including the selection of information, the change of the presentation, or the triggering of some action based on gathered context.

The GUIDE project [25] developed a tourist guide for the city of Lancaster in the UK. Personal and environmental context are used including, for example, the visitor's interests, her current location, the time of the day, and the opening hours of attractions. The information is presented with respect to the age and the technical background of the visitor.

Sharing of context of a mobile phone user was the focus of the TEA project [110]. The user can set his current context using his mobile phone for example to 'Free' or 'Meeting'. This information is presented to the caller which can then decide to call anyway, to leave a message, or to cancel the

call.

Gaia OS [99] provides support for Active Spaces that combines physical and virtual contextual information related to the physical space and allows interaction with the physical space. Applications can be developed without strict knowledge of the infrastructure – Gaia is responsible for mapping these applications to a physical space. A framework that separates model, view, and controllers allows for runtime adaptation due to changes in the environment.

In ad-hoc environments, context-based adaptation may have different goals, e.g., lower energy consumption or lower latency. Impala [68] contains an application adapter that adapts the application protocols to different runtime conditions, which include system and application parameters. Adaptation decisions are made using a finite state machine where states represent different protocols/applications. Each directed edge carries a parameter expression for the condition under which the switch occurs. The adaptation goal is, therefore, implicitly given by these conditions.

TinyCubus [75] uses several variables in three dimensions: 'system parameters', 'application requirements', and 'optimisation parameters' of the object context to perform the adaptation. For each combination of parameter values, an algorithm is known to the system which performs best. Based on policies and different adaptation strategies, a set of algorithms is selected that provides the functionality required by the application and that fulfils the best desired optimisation.

4.1.3 Location Management

Location services for mobile ad hoc networks only offer limited context information, i.e., the position of mobile objects. Prior to storing the location in such a service, the location has to be determined. Small cooperating objects usually do not have a GPS device, so different approaches are needed.

We refer the reader to the following surveys [47, 84] for an extensive discussion on infrastructured and infrastructure-less location mechanisms. This section covers some of the most recent research advances in location management and determination.

A scalable, distributed location service is GLS [67]. Each node has a small set of other nodes as its location servers and updates them periodically with its location. Thereto, it does not have to know their actual identities but only their identifiers. All routing — also routing of location queries — use a predefined ordering of node identifiers and a predefined geographic hierarchy.

Two basic approaches are commonly used to determine the position of objects. Having distances to three objects of which the location is known, the own location can be calculated. The other possibility is to measure the angle to two known objects. Since most cooperating objects are equipped with omnidirectional antennas, a very accurate measurement of the angles is not feasible. Though, [74] shows that the location estimation is significantly more accurate and feasible on Mica2 motes using two directional antennas.

Several methods exist to determine the distances to other nodes, for example time of flight or attenuation. Besides the normal radio, ultrasound can be used [109] which is a more accurate system for distance measurements and does not suffer from some problems like relying on radio received signal strength. For computing the node locations, in [109] a global non-linear optimisation problem is set up and solved. A fully distributed approximation for the solving algorithm is presented. Some special nodes capable of long distance ranging (e.g. using long range ultrasound) are used as initial location beacons.

The location discovery algorithm in [36] works with distance measurements based on the received signal strength indication (RSSI) values delivered by the RF chip. Especially in indoor environments, RSSI can have an error as large as 50% of the measured distance. Since the error conforms to a Gaussian random variable, it can be calculated with the location. The standard deviation is used as the degree of precision of the location. The precision estimates of all values are considered in all subsequent calculations of location for undetermined nodes, thus accumulating the errors in the results of the new calculation.

While the last approach assumes static nodes, an algorithm for semi-static sensor networks is presented in [33]. It uses the properties of a randomly deployed sensor network, more precisely the average density of nodes that are uniformly distributed. Assuming a fixed transmission range for all nodes, the distance between any two nodes can be calculated using only the hop count between them. In a further step, the hop count is combined with a distance estimate obtained in a traditional way. By limiting the number of hops a distance message can travel, the precision can be increased again.

Although considerable research have been done in the area of location estimation, positioning is still too inaccurate. Also, for mobile cooperating objects no good approaches exist. Therefore, more research is needed in this direction.

4.2 VF: Data Consistency

The benefits of having several CO nodes mostly come from the fact that many nodes simultaneously monitor the same physical area. Nodes can be put into sleep mode without any loss of precision in the network. This results in conservation of energy and an increased network lifetime.

The reliability of the system is also improved with several sensor nodes. This scenario, however, raises issues regarding data inconsistency which may occur due to various reasons - for instance inherent imprecision associated with sensors, inconsistent readings and unreliable data transfer, just to name a few.

This VF provides the functionality to ensure consistency of the sensor data at various system abstraction levels:

- Data consistency may mean that data retrieved from a location in the sensor network should be consistent with data sent to the same location.
- Data consistency may also mean that all sensors sensing the same physical phenomenon should more or less agree on the measured value.
- In a rule-based system, data consistency may mean that all actuators agree on the action that needs to be taken.

Different mechanisms have been used to solve data inconsistency problems in various levels. As will be shown in the next section, while the focus of the first definition of ‘data consistency’ is on low level, the second concerns the ‘data consistency’ on higher level and is more commonly known as ‘consensus’ and ‘data aggregation’. The third definition, also at a higher level, relates to the future direction of handling data inconsistency for complex WSN applications.

4.2.1 Consistency Handling Mechanisms (Operation of WSN)

The following primitives are needed to ensure accurate and consistent operation of the WSN and cooperating objects:

Localisation In order to interpret sensor data and collaborate with other nodes, it is crucial for sensor nodes to know approximately the position of the nodes with whom they collaborate. Usually manual configuration of nodes' positions is not feasible, and in situations where nodes are mobile this approach is even impossible. The availability of computing and communication capabilities on nodes makes it possible to use automated location techniques.

Synchronisation WSNs and distributed CO systems must have a mechanism to ensure all nodes have an equal understanding of time and the moment at which events take place. Consequently, the nodes must keep their local clocks approximately synchronised with respect to a reference time, which may be one of the sensor nodes or an external source of time (e.g GPS). The time synchronisation VF is discussed in Section 4.8.

Reliable Data Transfer Applications require guaranteed delivery of information and/or customisable degrees of reliability for data transfer. As sensor nodes are ubiquitously deployed they can overcome lack of reliability through cooperation. Nevertheless, achieving dependability through collaboration among error-prone entities is a challenging task. On the one hand, collaboration mechanisms should be ingenious enough to provide best-effort robust communication of important data, even in harsh conditions. On the other hand, the overhead introduced by cooperation along with the additional energy consumption should be kept to a reasonable level. Due to the fact that standard approaches cannot be applied to WSNs, reliability remains an open research issue.

Routing Routing is an essential mechanism in networking, wireless sensor networks included. Unlike traditional networks, there are no dedicated routers in ad hoc sensor networks. Instead, data forwarding from source(s) to destination(s) is accomplished through local collaboration among neighbours. The various techniques proposed in the literature strive to achieve energy efficiency while maintaining a best effort level of reliability.

Coverage The coverage problem in WSNs generally refers to how well an area is monitored. Monitoring an area by several sensors has the advantages of *(i)* being able to turn off some of the sensors, thus, saving valuable energy, and *(ii)* enhancing the accuracy of the sensed data by averaging multiple readings, for instance.

4.2.2 Consistency Handling Mechanisms (Data Processing)

Data consistency at the data processing phase can be achieved through the following mechanisms:

State monitoring Sensor nodes can detect any change in state of the environment, in which they are placed, directly from the sensed and measured data. In many situations, the whole network will not have a single global state that needs to be monitored, but the network will monitor the states of local processes, restricted to a confined area. The state might also be unique to each sensor node, or to the object the node is operating on behalf of.

An attempt to formalise state monitoring is presented in [101]. Instead of composing a state machine that is triggered by the occurrences of events, the authors propose to describe the creations between states and the events that trigger state-change through a set of rules and predicates over events and their parameters. In this way arbitrarily complex state change conditions can be defined. In this system states have a binary nature as they are either occurring or not occurring. A change of state can be used to trigger additional events or to perform actions. Strohbach et al. [119] used a similar approach, using simple predicate logic to monitor hazardous situations such as chemical drums.

Crucial to the design of a collaborative distributed state monitoring is the use of some high-level description of the state transitions, and other relevant aspects.

Data fusion If redundancy is used to cover each point or region with multiple sensors, then the accuracy and the consistency of sensed data can be improved by merging or fusing correlated sensor data. Various schemes for data fusion have been proposed which deal with reduction in transmission rates of the radio module (an expensive sensor node's resource).

In this case, however, we are faced with the problem of fusing or combining the data reported by each of the sensors monitoring a specified point or region.

This is a challenge as measurements recorded by the sensors can differ (because of inherent imprecision in the sensors and/or the relative location of a sensor with respect to the monitored region) and the fact that sensors might be faulty. The objective of sensor data fusion is to take the multiple measurements and determine either the correct measurement value or a range in which the correct measurement lies.

The sensor fusion problem is closely related to the Byzantine agreement problem that has been extensively studied in the distributed computing literature. In [107] a hybrid distributed sensor-fusion algorithm is presented. Each sensor needs to compute a range, in which the true value lies as well as the expected value. For this computation, each sensor sends its measurement and its estimated accuracy to every other sensor. This algorithm is executed by every sensor using the measurement ranges received from the remaining sensors monitoring the same region combined with the sensor's own measurement. A typical drawback of this approach is the considerable communication overhead introduced by exchanging so many messages. This is an important issue in the context of energy constrained WSN nodes.

Event Detection Event detection is similar to state monitoring in a sense that the sensor network itself is in charge of monitoring the environment in which its nodes are placed, and it is used to detect occurrence of certain events.

Thanks to event detection mechanism exceptional situations can be detected, and consequently be reported. Each sensor node has the task of detecting a possible event based on the data it obtains through its sensors, and through communication with other nodes. Additionally, the occurrence of an

event, along with its position, magnitude or other properties need to be reported. In many situations, the value of these additional properties can only be determined, or more accurately be determined, when sensor data of multiple sensor nodes are considered. The detection and reporting of these tasks will need to be performed in an efficient way, using only limited communication between the nodes, whenever data of interest is gathered, instead of reporting every sensed data sample. Beside that, a description of the event and its properties will be needed to perform the task.

Event detection has been tried in several WSN applications already. In [128] a description is given on a networked sensor array for monitoring volcanic eruptions. Seismic and infrasonic data is gathered continuously, and when higher levels of activity are measured, the recently measured data from different sensors is to be correlated and analysed to find data on the recent event. The implementation described does not make use of in network processing of the sensor data, but a feasibility analysis is performed. Other uses of event detection are shown in systems that are concerned with the detection, identification, localisation or tracking of objects in sensor fields [100, 123, 42, 31]. In these applications sensor nodes around the object or event of interest collaborate to find the location of the object or event.

Fault Tolerance and Consensus Clouqueur et. al in [26] present two distinct approaches, value-fusion and decision fusion, for achieving fault-tolerance in collaborative target detection algorithms. When performing a target detection task, multiple sensors in a region detect the presence of an object using sound, motion, or heat associated with the object of interest. Therefore, combining their views and obtaining a consistent conclusion through fusion process is highly desirable. Value-fusion method consists of two phases: *(i)* exchanging the measured values and *(ii)* arriving to a consensus by computing the average of values and comparing it to a threshold. In the presence of faulty sensors, in order to preserve precision and accuracy, the extreme values are dropped from the set that is going to be averaged. In contrast, in decision-fusion method, each device first makes an independent decision as to whether or not a target is present and then the devices exchange their decisions to arrive at a fault tolerant consensus decision. As in value fusion, the fused data is obtained by averaging data received from all the sensors. For the situation of faulty nodes, exact agreement is used to preserve precision. The comparative results show that value-fusion is clearly preferable if the sensor network is highly reliable and fault free. However, when faulty nodes are present, the performance of value-fusion degrades faster than the performance of decision-fusion and decision fusion becomes superior to value fusion. Achieving fault tolerance through consensus is a broad problem. Generally, each node has the task of trying to obtain a confident statement of the state or situation about the environment is in. This information constitutes the state information present at the node. The state information needs to be refreshed periodically to account for changes in the topology due to movement of wireless sensors or due to nodes encountering crash failures.

The major challenge within WSN context is to devise protocols that minimise the effort for refreshing and exchanging state information over the network. The protocol described in [63] generates consensus under the assumption that the number of faulty nodes is less than the number of correct nodes. The protocol is designed to enable self-correction in the network by isolating the faulty nodes and putting them in sleep mode, thereby increasing the concentration of nodes having correct information and improving their ability to generate consensus. The consensus process relies on forming quorum, i.e. subgroups of sensor nodes with a certain minimum size. For instance, if there are k

nodes having the same area of interest, the size of the quorum should be at least $(k+1)/2$. A quorum is created after exchanging information among nodes, as follows: *(i)* the node that needs to generate consensus starts the process by sending its information, *(ii)* other nodes answer with positive or negative acknowledgements, according to their information and *(iii)* the consensus is generated if the initial node receives more than $(k+1)/2$ positive acknowledgements. As we have mentioned before, under WSN constraints, protocols should reduce as much as possible the number of messages exchanged.

One way to achieve this is using aggregation [3] within the consensus process, either by concatenating values, summarising them, or by counting of a set. Other operations as well as aggregation of other data than scalar numbers could also be possible. Key factor in using aggregation is that the operation reduces the amount of data from a whole series of numbers (or different type of data) to a single number, or a single value, or small set of values, and that the aggregation can be performed hierarchically, creating an aggregate of a larger data set from the aggregates of smaller sets. Aggregate operations do need the support of some sensor network management structure or service to ensure proper operation. Aggregation can also make the protocol more susceptible to channel errors: for low to medium bit error rate concatenation has good performance, whereas for high bit error rate aggregation is not recommended.

In conclusion, the consensus problem in WSN has additional constraints compared to traditional distributed systems. However, it is a highly important process for increasing the confidence of the overall system, both by fusing sensed data and by providing fault tolerance.

4.2.3 Consistency Handling Mechanisms (Application Programming)

The more intelligent collaborative WSN applications become, the more essential becomes the need for algorithms capable of supporting collective reasoning and actions in an efficient way. The execution space for these collaborative algorithms is represented by groups of COs that combine their efforts to be able to agree on an action to be taken.

Such collaborative algorithms need to benefit from a novel high-level description mechanism (language) oriented towards the collective model of solving tasks. A promising mechanism, extensively explored in the database research community, is to have a rule-based language [119] to implement collaborative WSN applications. In this case, a set of rules that are stated as machine-understandable statements describes legal or allowable states or situations as well as the alerts that must be given when otherwise.

Although such scenario has the advantage of providing a high adaptive framework for collaborative WSN applications, which in turn can enhance system performance, it may cause severe problems if the rules and policies have not been carefully specified. An important reason for data inconsistency in rule-based collaborative applications is due to *versioning*. To prevent this kind of inconsistency from happening, the fact that the COs are using the same (latest) version of the rules should be enforced. On the other hands, since rules and policies should be executed collaboratively, best performance of the system highly depends on consistency between such rules for all COs involved, otherwise, rules may cause conflicts and prevent the system from functioning.

Therefore, strategies are needed to check for rule consistencies of COs.

4.3 VF: Communication Functionality

The communication vertical function refers to the capability of any pair (or group) of devices to exchange information. Different kinds of communications can be performed: one to all, one to many, many to one, many to many. If we consider the case of wireless sensor networks with a single sink, one to all or one to many communications are needed for sake of interest and query dissemination, while many to one communication is exploited to gather sensed data at the sink.

The first problem that needs to be addressed is CO addressing. Communication in a cooperating objects environment is expected to be *data centric* and attribute-based. This means that more than addressing a specific cooperating object the communication infrastructure should be able to deliver data to and from groups of cooperating objects which share a set of attributes specifying the destination/source address of the information. For example, a user could issue a query on the average temperature of an area in an office building. This query should be delivered to the objects with temperature sensors. Similarly, once measures on the temperature have been taken, the cooperating objects in the specific area will send packets to the sink(s) reporting the measured values.

Other important VF parameters which should be included in a query are the time constraints and accuracy with which a given query needs to be answered. These are application-dependent and even query-dependent parameters: (a) for a query to be successfully resolved, the sensors must deliver fresh data, (b) the query must be answered fast enough, and (c) the precision with which the query is answered must meet the query requirements. This translates into a new concept of 'quality of service' requirements.

Once a communication request between two or more cooperating objects is initiated, protocols have to be adopted to deliver the transmitted information from the sender to the final destination. Such protocols will typically involve physical layer protocols, data link protocols (FEC, ARQ protocols), medium access control protocols, 'topology control' schemes (e.g. addressing the self-organisation of nodes into a hierarchical network topology or into dynamically changing communications infrastructures according to given awake-asleep schedules), and routing protocols.

The second issue to address is the fact that a one-fits-all protocol stack may not be suitable to this scenario. It is possible to identify the 'vertical functions' that should be provided and possible implementations of such functions for specific sets of possible applications. The communication protocols will benefit from and sometimes require the information provided by different vertical functions such as time synchronisation and location awareness. Not only time and location information are included in the delivered data, but some protocols such as geographic-based routing can exploit location-awareness to reduce routing overhead and the nodes' storage demand.

Mobility of objects is the third issue to be tackled. Although in many cooperating objects scenarios the devices themselves are unlikely to be mobile they can be located on mobile users or mobile stations so that their location changes in time in a predictable or unpredictable way. On one hand this may have a beneficial effect (e.g. load balancing the energy consumption among the different nodes) but on the other hand it requires mobility management or mobility-aware protocols to be added to the protocol stack. The mobility of some of the devices have been explored by some architectures such as the data mules [112], in which a group of mobile nodes move in the deployment area collecting data from the sensor nodes and delivering the collected data to the sinks.

Although TinyOS [48] has been serving as the basis for experimentation of existing and new com-

munication protocols, a common and consolidated framework for comparisons needs to be further developed.

4.4 VF: Security, Privacy and Trust

Cooperating objects are usually placed in locations that are accessible to everyone – also to attackers. For example, sensor networks are expected to consist of a couple hundred of nodes that may cover a large area. It is impossible to protect each of them from physical or logical attacks. Thus, every single node is a possible point of attack.

In a cooperating object, security is pervasive. [87] states that security must be integrated into every component to achieve a secure system. Components designed without security can become a point of attack as [54] shows. However, specific vertical functions to enforce security are available for applications.

We start by describing how the hardware can be secured. Then, several encryption approaches for cooperating objects are presented. Closely connected since encryption is mostly a prerequisite, secrecy, privacy, data integrity, and trust are discussed. We conclude with a look at routing protocols.

4.4.1 Resource protection

Having physical access to the devices, without countermeasures an attacker could read out the node's memory including cryptographic keys and reprogram the node with malicious code. Therefore, security starts with the hardware of sensor networks.

Tamper resistant devices would make the integration of security in sensor networks much easier — we could rely on the strength of security protocols or cryptographic functions which have been known already for many years. Unfortunately complete tamper resistance is very hard to achieve [9]. There are a lot of known techniques to read data from devices which actually are meant to be tamper resistant. For instance, [64] describes methods for extracting protected software and data from smartcard processors. They show invasive techniques like microprobing and non-invasive techniques like software attacks, eavesdropping, and fault generation. They even go further and show that also additional countermeasures like additional metallisation layers that form a sensor mesh above the actual circuit do not protect the circuit fully. Of course, some of these methods require a well funded adversary, but thinking for instance on military applications the existence of such an adversary is highly probable. Additionally to that, achieving such a tamper resistance level is highly costly — which in most cases is not inline with the requirement that a single device should be low cost.

Thus, the conclusion is, that in sensor networks tamper resistant devices can be used only in a very limited and specialised areas — in the majority of cases the design of the security protocols has to take into account that at least some devices get compromised by the adversary. Thus, security protocols for sensor networks have to be designed in a way that they tolerate malfunctioning/attacking nodes while the whole sensor network remains functional — Karlof and Wagner [54] talk in that context about a *graceful degradation* of the network in contrast to a totally compromised network.

Besides the above security issues, there is another danger for sensor networks, which is hard to prevent: denial of service attack (DoS) on the physical layer. In such an attack, the attacker broadcasts a high-energy signal to prevent any node from communicating. The attacker can also use the features of the MAC protocol by continuously requesting channel access to eliminate all normal communication. Cryptographically secure spread-spectrum communication would be a defence against such jamming, but unfortunately such RF-devices are not commercially available yet [87]. In case of the jammed region not covering the whole network, the border nodes could create a map of this region and reroute traffic around the jammed region [130]. Also, battery exhaustion is a DoS attack. This can be counteracted for instance with rate limiting [130] which causes the network to ignore requests beyond a threshold value. Thus, although there are some approaches to cope with some kinds of DoS attacks, there is still further research needed in this area.

4.4.2 Encryption

Encryption is the basic technique for securing and authenticating transmitted data. Using asymmetric cryptography on highly resource constrained devices is often not possible due to delay, energy and memory constraints [20, 18]. It is also not expected that the devices of such sensor networks are going to have more resources in the future — there is an interesting observation by Karlof and Wagner that sensor network devices will more likely ride Moore's law *downward* [54]. They make the point that instead of doubling computational power every 18 months it may be more likely that the devices become even smaller and cheaper solutions are sought.

Using symmetric cryptographical methods, the easiest solution is a shared key for the whole network like in Secure pebblenets [11]. The disadvantages are obvious: If a single node is compromised, all the network traffic can be decrypted. Additionally, no device to device authenticity can be achieved.

If a unique key is only shared between any two nodes of the network, a single compromised node has only limited impact. The attacker can read and modify only data that was sent to it, i.e. the compromised node directly. All other network traffic remains secure. As a drawback of this approach, a node needs to store keys for all other nodes — which may not be in line with scalability and memory constraints. Moreover, these unique keys have to be established in some way.

SPINS [88] uses a central base station to establish new session keys. This introduces a single point of failure that has to be protected exceptionally. Additionally, cooperating objects exhibit strong ad-hoc character. Therefore, one can assume neither an infrastructure nor a base station. A decentralised key management system is, therefore, more practical.

Several approaches are based upon random pairwise key pre-distribution [35, 23]. That is, before deployment — for instance by the sensor manufacturer — every device is supplied with a random set of keys from a key pool. After deployment, the devices try to establish connections by finding a commonly shared key or by creating a new key through a secure path including other devices. Due to random pre-distribution, real authenticated communication between arbitrary devices is not always possible. It can only be assured with some probability that two arbitrary devices are able to communicate in a secure way. Moreover, an attacker could also reconstruct the complete key pool by compromising enough nodes.

If the key pre-distribution does not rely on a random set but on some algorithmic chosen set, the

secure communication between arbitrary devices can be guaranteed. In [125] such an approach is presented. The authors suggest physical contact[1] for establishing the initial key set — nevertheless also the manufacturer could pre-distribute this initial set of keys according to their algorithm. In their approach, when two devices do not share a key yet, they can establish a new key by sending key-shares along node-disjoint paths via secured point-to-point communication to each other. When all key-shares are available at both nodes (which want to establish a new key) they just have to perform for instance a bitwise XOR operation on all key-shares — the result of this operation is the newly unique key, shared only by those two devices. It should be clear, that the attacker would also need access to all key-shares in order to gain knowledge of the new key.

Since in-network processing is used to save power, end-to-end encryption can only be used sparsely. Along an aggregation tree, only point-to-point encryption is feasible, but an attacker in the tree has full access to the data. If the aggregation is locally bounded, the results of the aggregation can be sent directly and encrypted to the target node since no intermediate node is expected to change the data.

4.4.3 Secrecy

Encryption is not sufficient for ensuring secrecy of data. Traffic analysis on the ciphertext can reveal sensitive information about the data. When, e.g., a motion detector sends a message, one does not need to know the data but can assume that it detected a motion. On higher layers, additional dummy messages have to be generated to hide the important messages. This seems to be diametrical in resource constraint cooperating objects, but secrecy may be a more important goal than energy saving in some cases.

A dynamic virtual infrastructure for wireless sensor networks is presented in [126]. The system contains a coordinate system, a cluster structure, and a routing structure. An energy-efficient protocol is proposed to maintain the anonymity of the network virtual infrastructure by randomising communications so that it cannot be observed by an external attacker.

4.4.4 Privacy

The primary goal of sensor networks is to observe real-world phenomena. As long as nature is the target, there is no privacy issue. But if humans are observed, privacy is threatened. However, cooperating objects used in medical applications, for example, do have to monitor the status of a patient. Encryption has to ensure that data cannot be overheard and is only available to legitimate applications. If nodes are compromised, other mechanisms like intrusion detection have to exclude these nodes from receiving data.

Data of other sensor networks might be available to everyone. Such networks should only deliver data in a coarse-grained detail level that is not dangerous to privacy. For example, the system should only return average values of larger areas but not of single locations. The result has to be computed in a distributed manner so that no single node has access to the complete result.

[1] (physical contact for establishing an initial security association, i.e. a shared secret/key is also suggested by Stajano and Anderson [118])

Sensor networks are small enough and will even become smaller in the future. Therefore, they are outstandingly suitable to spy on people. But beside jamming transmitters and bug detectors, pure technological solutions are not able to solve the problem. [87] suggests a mix of societal norms, new laws, and technological responses. However, they require more research in the future.

4.4.5 Data Integrity

An attacker in the network can generate false sensor or, more severe, aggregation data. The more aggregated the data is, the more valuable it is. Therefore, there is a need to protect especially the aggregated data. [93] proposed SIA, a framework for secure information aggregation in large sensor networks: efficient protocols for the computation of the median, the average, the minimum and maximum of a value and the estimation of the network size. Using random sampling mechanisms and interactive proofs, the user is able to verify that the answer of an aggregator is a good approximation of the true value.

4.4.6 Trust

With several cooperating objects contributing to a common goal, it is necessary to assess the reliability of the information provided by an individual cooperating object. With respect to services and transactions, trust has been researched for several years. For data-centric and fully distributed architectures research has just started.

A distributed voting system is proposed in [23]. When a node detects a misbehaviour of another node, it can cast a vote against this node. Above a certain number of votes, all other nodes refuse to communicate with this node. To avoid a malicious node casting votes against many legitimate nodes, each node is limited to a number of votes.

In [38], a more general reputation-based framework for sensor networks, is presented. Each node monitors the behaviour of other nodes and builds up their reputation thereupon. This reputation is used to evaluate the trustworthiness of other nodes. Thus, nodes with a bad reputation can be excluded from the community.

Cooperating objects exhibit strong cooperation when performing an action. Thus, a single node becomes less important to the overall result. Therefore, strong cooperation will be one of the feasible approaches to secure the entire system through extensive data validation.

4.4.7 Protocols

Encryption mechanisms are not enough to defend against attackers. Careful protocol design is needed as well. [54] describes several attacks against known routing protocols for sensor networks: spoofed, altered, or replayed routing information, selective forwarding, sinkhole attacks, sybil attacks, wormholes, HELLO flood attacks, and acknowledgement spoofing. For several routing protocols, the relevant attacks are highlighted. Countermeasures could not be given for all protocols since security was not a design issue.

In [30] a routing protocol is presented that is resilient to attempts to obstruct data delivery as it sends every packet along multiple, independent paths.

Ariadne [50] uses efficient symmetric key primitives to prevent compromised nodes from tampering with uncompromised routes consisting of uncompromised nodes. It also deals with a large number of types of DoS attacks. Since it is designed for ad-hoc networks, it may be suitable only for some cooperating objects.

4.5 VF: Distributed Storage and Data Search

In cooperating objects and wireless sensor networks (WSNs) efficient storage and querying of data are both critical and challenging issues. Especially in WSNs large amount of sensed data are collected by high number of tiny nodes. Scalability, power and fault tolerance constraints make distributed storage, search and aggregation of these sensed data essential. It is possible to perceive a WSN as a distributed database and run queries which can be given in SQL format. These queries can also imply some rules about how to aggregate the sensed data while being conveyed from sensor nodes to the query owner.

Since the lifetime of a WSN is generally dependent on irreplaceable power sources in tiny sensor nodes, power efficiency is one of the critical design factors for WSNs [6]. Data aggregation techniques [17, 135] that reduce the number of data packets conveyed through the network are therefore important and also required for effective fusion of data collected by a vast number of sensor nodes [6, 43]. Data aggregation in sensor networks combines the sensed data coming from the nodes based on the parameters passed in queries. It can be classified according to one of the following approaches:

- **Temporal or spatial aggregation:** data can be aggregated based on time or location. For example, the temperature readings taken every hour or temperature readings from various regions in a sensor field can be averaged. Also a hybrid approach which is the combination of time and location based aggregation can be used.
- **Snapshot or periodical aggregation:** data aggregation can be made snapshot, i.e., one time, on the receipt of a query. Alternatively, temporarily aggregated data can be reported periodically.
- **Centralised or distributed aggregation:** a central node can gather and then aggregate data or data can be aggregated while being conveyed through a sensor network. A hybrid approach is also possible where clusters are formed, and a node in each cluster aggregates the data from the cluster.
- **Early or late aggregation:** data can be aggregated at the earliest opportunity, or aggregation of data may not be allowed before a certain number of hops hinder the collaboration among the neighbouring nodes.

Data queries can be made not only for aggregated data but also for non-aggregated data. A query in a sensor network may be perceived as the task or interest dissemination process. Sensor nodes can be queried by using continuous or snapshot queries. Continuous queries can be periodical where the sensed data are reported at certain time intervals or event driven where certain events stimulate nodes to report the sensed data.

The following characteristics of WSNs should be considered while designing a data storage, querying and aggregation scheme for WSNs:

- Sensor nodes are limited in both memory and computational resources. They cannot buffer a large number of data packets.
- Sensor nodes generally disseminate short data packets to report an ambient condition, e.g., temperature, pressure, humidity, proximity report, etc.
- The observation areas of sensor nodes often overlap. Therefore, many sensor nodes may report correlated data of the same event. However, in many cases the replicated data are needed because the sensor network concept is based on the cooperative effort of low fidelity sensor nodes [6]. For example, nodes may report only proximity, then the size and the speed of the detected object can be derived from the locations of the nodes reporting them, and timings of the reports. The collaboration among the nodes should not be hampered by the data aggregation scheme.
- Since there may be thousands of nodes in a sensor field, associating data packets from numerous sensors to the corresponding events, and correlating the data about the same event reported at different times may be a very complicated task for a single sink node or a central system to handle.
- Due to large number of nodes and other constraints such as power limitations, sensor nodes are generally not globally addressed [6]. Therefore, address-centric protocols (end-to-end routing) are mostly inefficient. Instead of address-centric protocols, data-centric or location aware addressing protocols where intermediate nodes can route data according to its content [62] or the location of the nodes [21], should be used.
- Querying the whole network node by node is impractical. So attribute-based naming and data-centric routing [114] are essential for WSNs.

Queries made to search data available in a WSN should be resolved in the most power efficient way. This can be achieved by reducing either the number of nodes involved in resolving a query or the number of messages generated to convey the results. There is a considerable research interest to develop efficient data querying schemes for WSNs.

In the section that follows we examine data dissemination techniques which are closely related to data querying, and then query processing and resolution techniques.

4.5.1 Data Dissemination

Data dissemination protocols are designed to efficiently transmit and receive queries and sensed data in WSNs. We briefly discuss five of the best known data dissemination protocols for WSNs. There are many others that can be categorised as data centric, hierarchical and location based. Since the focus is on distributed storage and search we do not list these other protocols.

The routing protocols for WSNs are generally designed for networks that have fixed homogeneous sensor nodes and are based on the assumption that all nodes try to convey data to a central node,

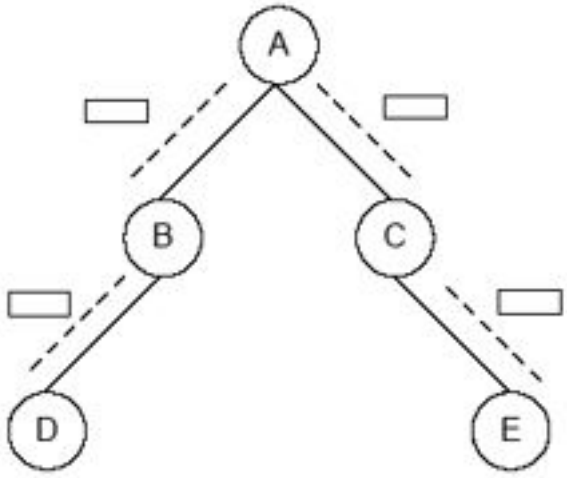


Figure 2: Classic Flooding

often named sink. However, in cooperating objects networks there will be heterogeneous nodes that can be mobile, and the sensed data will be needed by many nodes, i.e., multiple sinks. Therefore, we can say that new routing protocols will be needed for cooperating objects.

Classic Flooding In classic flooding, a node that has data to disseminate broadcasts the data to all of its neighbours.

Whenever a node receives new data, it makes a copy of the data and sends the data to all of its neighbours, except the node from which it just received the data. In Figure 2, an example is depicted. A sends the message to its neighbours B and C. Then, B and C copy the message and send the message to their neighbours D and E respectively. The algorithm finishes when all the nodes in the network have received a copy of the message.

Gossiping Gossiping [6], which uses randomisation to conserve energy as an alternative to the classic flooding approach. Instead of forwarding data to all its neighbours, a gossiping node only forwards data to one randomly selected neighbour.

SPIN SPIN [43] is based on the advertisement of data available in sensor nodes. When a node has data to send, it broadcasts an advertisement (ADV) packet. The nodes interested in this data reply back with a request (REQ) packet. Then the node disseminates the data to the interested nodes by using data (DATA) packets. When a node receives data, it also broadcasts an ADV, and relay DATA packets to the nodes that send REQ packets. Hence the data is delivered to every node that may have an interest. This process is shown in Figure 3.

Directed Diffusion In SPIN, the routing process is stimulated by sensor nodes. Another approach, namely directed diffusion [53], is sink oriented. A sink is the name given to the central node respon-

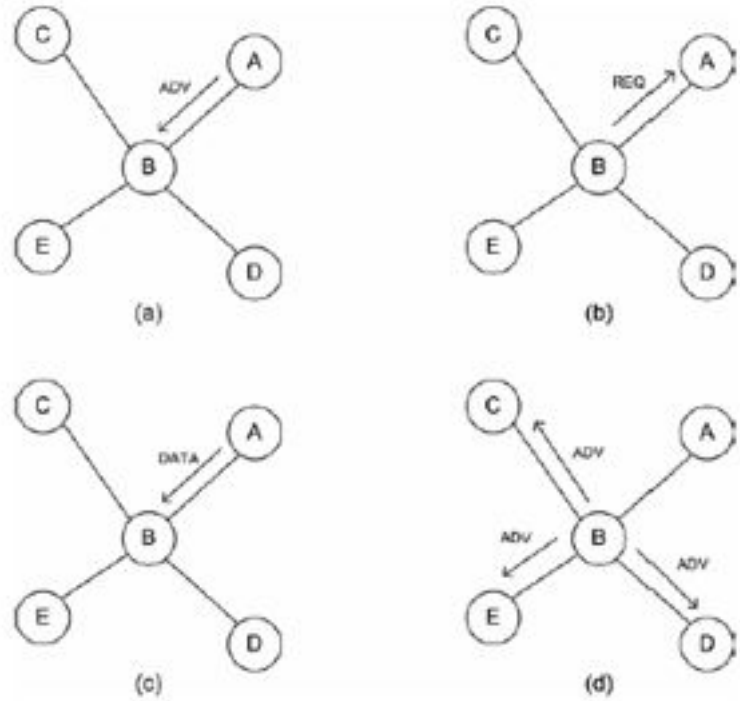


Figure 3: SPIN Data Exchange


```
Type
Interval
Duration
Rect
: four-legged animal
: 20ms
: 10 seconds
: [-100,100,200,400]
// Detect animal location
// Send back events every 20ms
// For the next 10 seconds
// From sensors within range
```

Figure 4: A Sample interest description

sible for gathering data from all the other nodes in directed diffusion where the sink floods a task to stimulate data dissemination throughout the sensor network. While the task is being flooded, sensor nodes record the nodes which send the task to them as their gradient, and hence the alternative paths from sensor nodes to the sink are established. When there is data to send to the sink, this is forwarded to the gradients. One of the paths established is reinforced by the sink. After that point, the packets are not forwarded to all of the gradients but to the gradient in the reinforced path. A sample interest description is shown in Figure 4, and data dissemination in directed diffusion is illustrated in Figure 5.

LEACH LEACH [45] is a clustering based protocol that uses randomised rotation of local cluster heads to evenly distribute the load among the sensors in the network. In LEACH, the nodes organise themselves into local clusters, with one node acting as a local cluster head. LEACH includes randomised rotations of the high-energy cluster-head position such that it rotates among the various sensors in order not to drain the battery of a single sensor. In addition LEACH performs local data fusion to compress the amount of data being sent from the clusters to the base station. Sensors elect themselves to be local cluster-heads at any given time with a certain probability. These cluster-head nodes broadcast their status to the other sensors in the network. Each sensor node determines to which cluster it wants to attach by choosing the cluster-head that requires the minimum communication energy. Once all the nodes are organised into clusters, each cluster-head creates a schedule for the nodes in its cluster. This allows the radio components of each non-cluster head node to be turned off at all times except during each nodes transmit time, thus the energy dissipated is minimised. Once the cluster-head has all the data from the nodes in its cluster, the cluster-head node aggregates the data and then transmits the compressed data to the base station. However, being cluster-head drains the battery of that node. In order to spread this energy usage over multiple nodes, the cluster-head nodes are not fixed. The decision to become a cluster-head depends on the amount of energy left at the node. In this way, nodes with more energy will perform

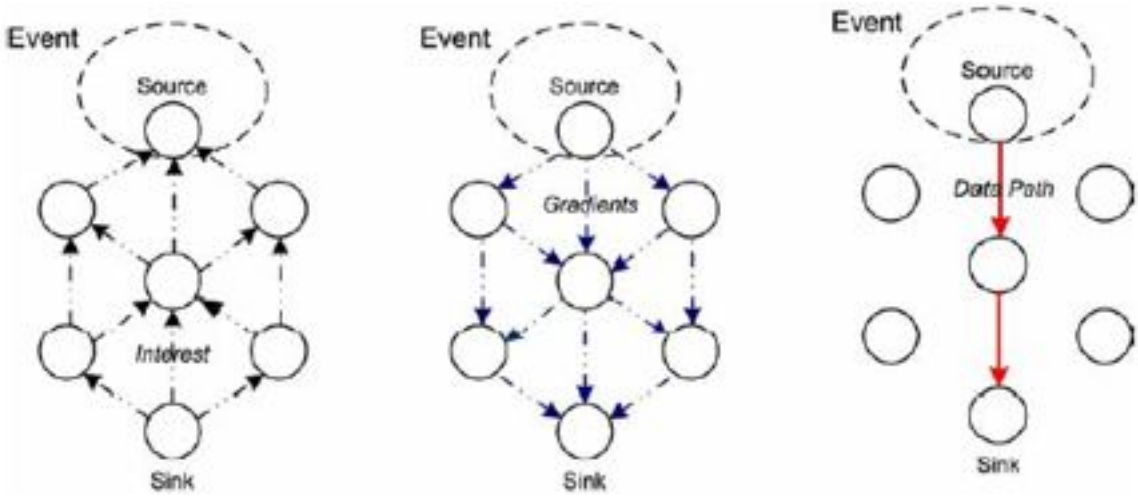


Figure 5: Directed diffusion

the energy-intensive functions of the network. Each node makes its decision about whether to be a cluster-head independent from the other nodes in the network and thus no extra negotiation is required to determine the cluster-heads. The cluster formation algorithm is depicted in Figure 6.

4.5.2 Query Processing and Resolution

After a query arrives to a sensor node, it is first processed by the sensor node. If the node can resolve the query, the result of the query is disseminated. This approach is one of the simplest ways of resolving and processing a query. Sensor nodes usually take advantage of collaborative processing to resolve queries so that a smaller number of messages are transmitted in the network. Queries can be flooding-based where a query is flooded to every node in the network. Alternatively they can be expanded ring search (ERS) based where a node does not relay a query that it can resolve. In this subsection, we briefly explain data storage and querying techniques in the literature.

TinyDB TinyDB [69] is a query processing system for extracting information from a network of TinyOS sensors. TinyDB provides a simple, SQL-like interface to specify the data, along with additional parameters, like the rate at which data should be refreshed much as in traditional databases. Given a query specifying data interests, TinyDB collects that data from nodes in the environment, filters and aggregates them. TinyDB does this via power-efficient in-network processing algorithms. Some key features of TinyDB areas follows:

- Metadata Management: TinyDB provides a metadata catalog to describe the kinds of sensor readings that are available in the sensor network.
- High Level Queries: TinyDB uses a declarative query language that lets the data be described without requiring stating how to get it. This makes it easier to write applications.

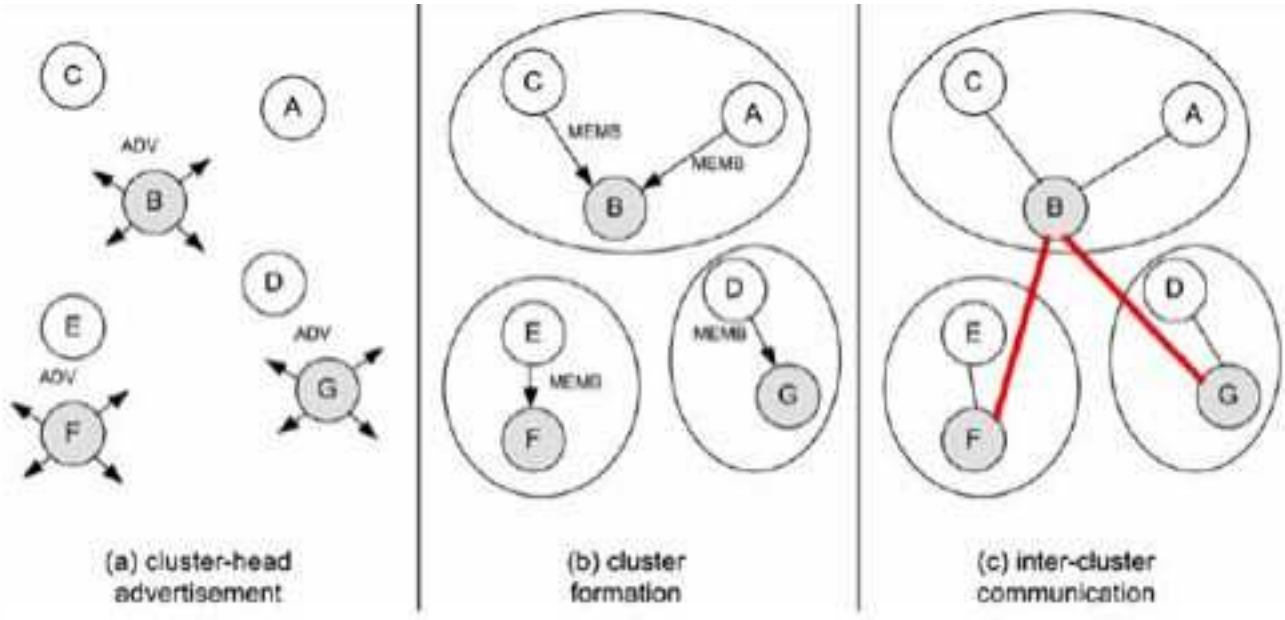


Figure 6: LEACH Cluster Formation

- Network Topology: TinyDB manages the underlying radio network by tracking neighbours, maintaining routing tables, and ensuring that every node in the network can efficiently and (relatively) reliably deliver its data to the user.
- Multiple Queries: TinyDB allows multiple queries to be run on the same set of nodes at the same time. Queries can have different sample rates and access different sensor types, and TinyDB efficiently shares work between queries when possible.
- Incremental Deployment via Query Sharing: TinyDB nodes share queries with each other: when a node hears a network message for a query that it is not yet running, it automatically asks the sender of that data for a copy of the query, and begins running it. The TinyDB system contains two applications: one application runs on the sensor platforms and another application runs on the PC side. A User requests his query using the Java application on the PC. This query is disseminated to sensor nodes and the application on the sensor platforms retrieves and returns the requested information.

TinyDB includes a facility for simple triggers, or queries that execute some command when a result is produced. A sample query is given in Figure 7 which calls SetSnd function to give alarm when the temperature is over some threshold value and this value is checked every 512 seconds. TinyDB includes the ability to run queries that log into the flash memory in the sensor nodes. TinyDB provides commands for creating tables that reside in flash, for running queries that insert into these tables, for running queries that retrieve from these tables, and for deleting these tables. One query can log to a buffer at a time, and new queries will overwrite data that was previously logged to a table. Currently, a query that selects from a Flash table and a query that writes to the same table


```
SELECT Temp
FROM Sensors
WHERE temp>threshold
TRIGGER ACTION SetSnd(512)
EPOCH DURATION
```

Figure 7: Triggering SQL query

can not be run. Logging of the query should be stopped using TinyDB Client utility prior to collecting data from flash tables.

When running queries longer than 4 seconds by default, TinyDB enables power management and time-synchronisation. This means that each sensor is on for exactly the same four seconds of every sample period. Results from every sensor node for a particular query should arrive at the base station within four seconds of each other. This time synchronisation and power management enables long running deployments of sensors. TinyDB aggregates results on the way to the sink node. TAG [72] is an aggregation service offered by TinyDB. It operates as follows: users pose aggregation queries from a powered, storage-rich base station. Operators that implement the query are distributed into the network by piggybacking on the existing ad hoc networking protocol. Sensors route data back towards the user through a routing tree rooted at the base station. As data flows up this tree, it is aggregated according to an aggregation function and value-based partitioning specified in the query. In order for users to pose declarative queries, an SQL like programming language was designed.

Aggregates are classified in four categories according to their state requirements, tolerance of loss, duplicate sensitivity and monotonicity:

- Duplicate Insensitive/Sensitive Aggregates are unaffected by duplicate readings.
- Duplicate sensitive aggregates will change when a duplicate reading is reported.
- Exemplary/Summary Aggregates return one or more representative values from the set of all values.
- Summary aggregates compute some property over all values.
- Monotonic Aggregates aggregate the property that when a function f is applied to two partial state records for all resulting values, it will be greater or lower than each of the evaluation of pairs of values. This provides increasing or decreasing values for aggregate results.
- Distributive/Algebraic/Holistic/Unique/Context-sensitive Aggregates: Depending on the function a pair of values has to be carried. For example AVERAGE function requires the number of elements used to compute the result and the result to further continue in processing. Distributive aggregates dont require other data to calculate a result; therefore, the size of the partial

records is the same as the size of the final record. COUNT, MAX and MIN are examples of distributive aggregates. Algebraic aggregates require intermediary state information to continue operation. The AVERAGE function is an example of algebraic aggregate. Holistic aggregates require whole values to be kept together prior to computing the result. MEAN is one of these operators. Unique aggregates are similar to holistic aggregates, except that the amount of state that must be propagated is proportional to the number of distinct values in the partition.

- In Context-sensitive aggregates, the partial state records are proportional in size to some property of the data values in the partition. Many approximate aggregates are content-sensitive. Fixed-width histograms and wavelets are examples of these operators.

Queries in TAG contain named attributes. When a TAG sensor receives a query, it converts named fields into local catalog identifiers. Nodes lacking attributes specified in the query simply tag the missing entry as NULL. This increases the scalability as not all the nodes are required to have a global knowledge of all attributes. Attributes can be sensor values, remaining energy or network neighbourhood information. TAG computes aggregates in network whenever possible to decrease the number of message transmissions, latency and power consumption. Given the goal of decreasing the number of transmitted messages, during the collection phase each parent waits for some time period prior to transmitting its own message in order to aggregate the child nodes' responses. How long each node waits for other nodes responses is (EPOCH DURATION)/d, where d is the maximum depth of the tree. In order to group received data, group id is tagged to each sensor's partial state record, so that response data is aggregated for the nodes with same group id. When a node receives an aggregate from a child, it checks the group id. If the child is in the same epoch as the node, it combines the two values. If it is in another epoch, it stores the value of the child's group along with its own value for forwarding in the next epoch.

By explicitly dividing time into epochs as in Figure 8, while waiting for other nodes responses to arrive, the CPU can be set to idle during this time. But, the depth of the tree has to be known for this principle to be made successful. However, waking up the processor will require synchronisation to be employed. Finally, real users of sensor networks are most likely not sophisticated software developers. Therefore, TinyDB has been supported by toolkits for easy access of data including TASK and others. The complexity of sensor network application development must be reduced and deployment must be made easy to ensure the success of sensor network technology in the real world.

COUGAR Cougar [132] is a query layer for sensor networks. The query layer accepts queries in a declarative language that are then optimised to generate efficient query execution plans with in-network processing which can significantly reduce resource requirements. Cougar is motivated by three design goals. First, declarative queries are especially suitable for sensor networks. Clients issue queries without knowing how the results are generated, processed and returned to the client. Second, it is very important to preserve limited resources such as energy and bandwidth. Since sensor nodes have the ability to perform local computation, part of the computation can be moved from the clients and pushed into the sensor network, aggregating records or eliminating irrelevant records. Third, different applications usually have different requirements, from accuracy, energy consumption to delay. For example, a sensor network deployed in a battlefield or rescue region may

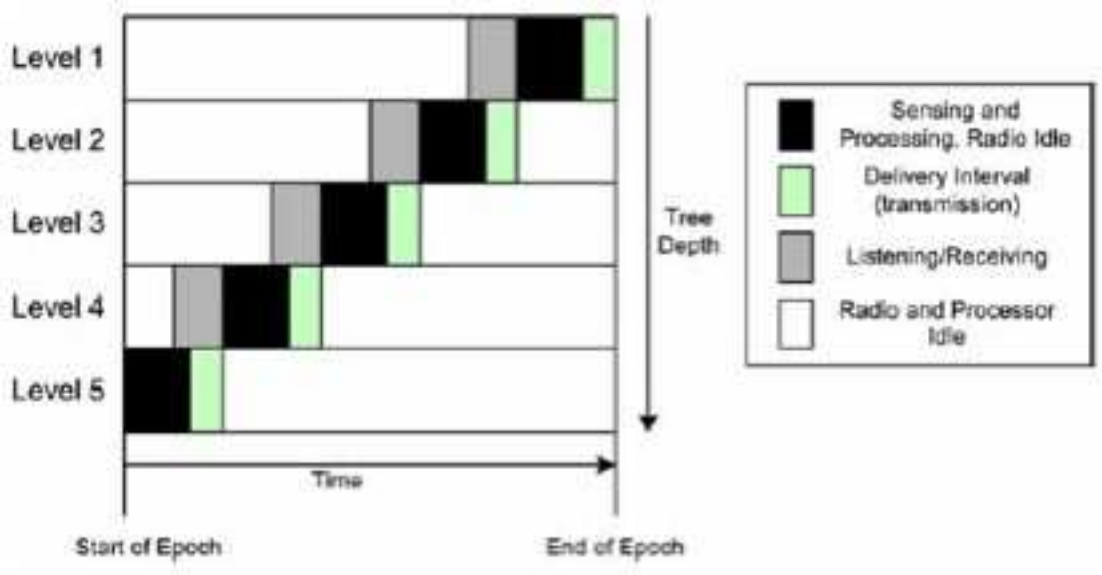


Figure 8: Partitioning of time into EPOCHS

only have a short life time but a higher degree of dynamics. On the other hand, for a long term scientific research project that monitors an environment, power-efficient execution of long-running queries might be the main concern. More expensive query processing techniques may shorten processing time and improve result accuracy, but might use a lot of power. The query layer can generate query plans with different tradeoffs for different users. The component of the system that is located at each node is called query proxy. Architecturally the query proxy lies between the network layer and the application layer and the query proxy provides higher level services through queries. Gateway nodes are connected to components outside of the sensor network through long-range communication and all communication with users of the sensor network goes through the gateway node.

Declarative queries are the preferred way of interacting with a sensor network. The queries having the form in Figure 9 are considered. It is very similar to the SQL language but it has limitations when compared to SQL. One difference between the query template and SQL is that the query template has additional support for long running, periodic queries. The DURATION clause specifies the life time of a query and the EVERY clause determines the rate of query answers. A simple aggregate query is an aggregate query without GROUP BY and HAVING clauses. In order to compute these aggregates, further processing such as in-network aggregation has to be done. In order to process data in-network, several sensor nodes transmit the packet to a central node named leader-node which calculates the aggregates of incoming messages. There are three approaches that can be taken to collect sensor data at a leader node. Messages can be directly delivered to the leader node using an ad hoc routing protocol, messages can be merged into same packet to limit the amount of packets transmitted and partial aggregation on the way to the central node can be done by intermediary nodes. Synchronisation is needed if the messages are to be merged and some duplicate sensitive operators such as SUM and AVERAGE require data to be transmitted once.


```
SELECT
FROM
WHERE
GROUP BY
HAVING
DURATION
EVERY
attributes, aggregates
SensorData S
Predicate
attributes
predicate
time interval
time span e
```

Figure 9: Query Template

Synchronisation is used to determine for each node in each round of the query how many sensor readings to wait for and when to perform packet merging or partial aggregation. Since the query processing facility has been designed as a layer, COUGAR assumes that several ad hoc routing protocols with modifications can be used for delivery of the messages. An AODV protocol has been used with extensions for simulations. According to COUGAR, routes are set up in an initialisation phase and each message carries the hop count of the message. Each node records the message receive ID as a parent node and a reverse path to the leader is set up. Two methods are used to maintain the tree. Local repair is used when a broken link is detected. Depth of the tree with sequence number is used between nodes that are spatially close to find a new parent in the case of a communication failure. Another method is to reconstruct the tree whenever the number of messages expected reach below some user defined threshold.

In order to resolve queries like 'What is the minimum average temperature during the next seven days?' two levels of aggregation have to be done. First the average temperature has to be computed and then the minimum operator has to be applied. In order to resolve these kinds of queries query plans are used. A query plan is needed to compute complex aggregate queries that a user poses. A query plan decides how much computation is pushed into the network and it specifies the role and responsibility of each sensor node, how to execute the query, and how to coordinate the relevant sensors. A query plan is constructed by flow blocks, where each flow block consists of a coordinated collection of data from a set of sensor nodes at the leader of the flow block as depicted in Figure 10. The task of a flow block is to collect data from relevant sensor nodes and to perform some computation at the destination or sensor internal nodes. A flow block is specified by different parameters such as the set of source sensor nodes, a leader selection policy, the routing structure and the computation that the block should perform. Each flow block is called a cluster and maintained by some heart beat messages transmitted by the leader of the flow. Several optimisations can be applied to query plan construction such as creating flow blocks that are sharable between different queries and use

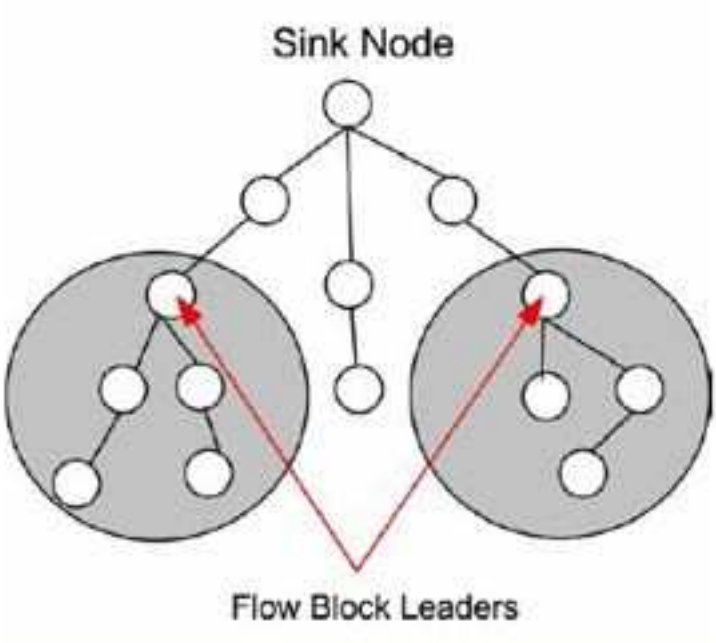


Figure 10: A Cougar Query Plan

of the join operator which enforces two conditions coming from different data flows to be true before returning a value. The join operator represents a wide range of possible data reductions. Depending on the selectivity of the join, it is possible to reduce the resulting data size.

ACQUIRE: Active Query Forwarding The active query forwarding in sensor networks (ACQUIRE) scheme [104] aims to reduce the number of nodes involved in queries. In ACQUIRE each node that forwards a query tries to resolve it. If the node resolves the query, it does not forward it further but sends the result back. Nodes collaborate with their n hop neighbours, where n is referred to as the look ahead parameter. If a node cannot resolve a query after collaborating with n hop neighbours, it forwards it to another neighbour. When n equals to 1 ACQUIRE performs as flooding in the worst case. Query resolution in ACQUIRE is depicted in Figure 11.

MARQ: Mobility Assisted Resolution of Queries Mobility assisted resolution of queries in large scale mobile sensor networks (MARQ) [46] makes use of the mobile nodes to collect data from the sensor network. In MARQ every node has contacts that are some of the other nodes. When contacts move around, they interact with other nodes and collect data. Nodes collaborate with their contacts to resolve the queries.

SQTL: Sensor Query and Tasking Language Sensor query and tasking language (SQTL) [114] is proposed as an application layer protocol that provides a scripting language. SQTL supports three types of events, which are defined by the keywords receive, every, and expire. The receive

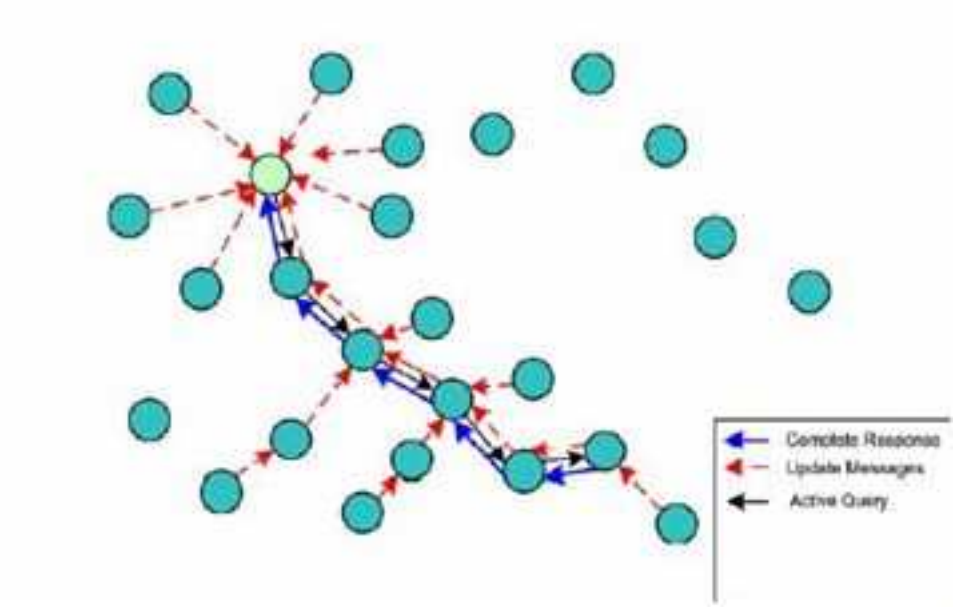


Figure 11: Query resolution in ACQUIRE

keyword defines events generated by a node when the node receives a message; every keyword defines events occurred periodically; and the expire keyword defines the events occurred when a timer expires. If a node receives a message that is intended for it and contains a script, the node executes the script.

SQS: SeMA Querying Protocol For Micro-Sensors SeMA is an enhanced client-server architecture in which clients are supported to adapt themselves to the new network resources they discover on the fly. The nodes get service information either by catching the periodically advertised services or by generating a service request. Applications establish light-weight sessions with the resources using the address returned to them. Binding is done when the service is needed. SeMA is a cross-layer protocol, capable of operating on generic wireless data link layer protocols, such as IEEE 802.11. SeMA addresses issues like service announcement, binding, session management and data routing. The monitoring of sensor networks is an application of SeMA architecture. The terrain of the sensor deployment area is assumed to be suitable for navigating by means of some mobile units, which form the information retrieval backbone of the overall monitoring application [12]. These mobile units are either wireless equipment carrying livings or autonomous robots as seen in multi robot exploration studies. The architectural view of the system is shown in Figure 12.

All available resources on the SeMA network are considered as services. They play a major role on a SeMA network, since mechanisms of the protocol are built with the aim of providing means to access those services. Specification of a service should include necessary information for SeMA clients to determine whether the service fulfils clients need. Thus, services are modelled with a generic name, and following attribute-value pairs. This definition is sufficient for a host to discover a required service without any bindings beforehand. Details of XML processing of services and ad hoc

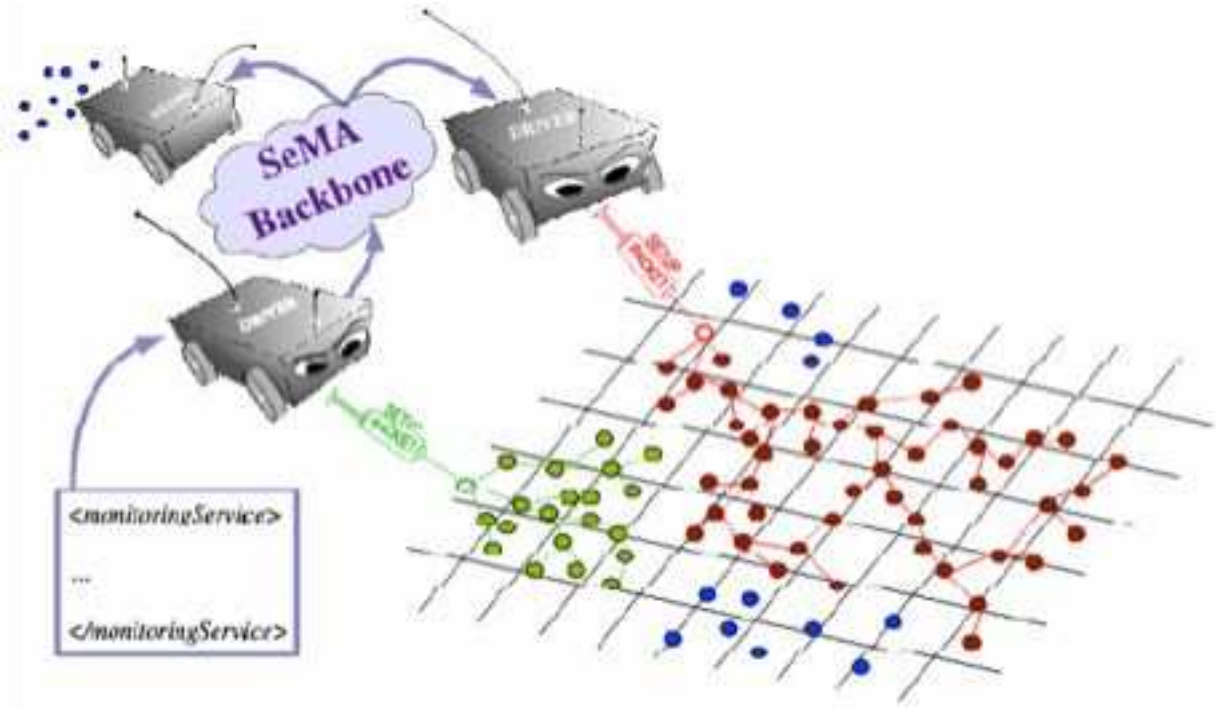


Figure 12: SeMA Architecture

routing in the backbone can be found in reference [12]. In a sensor network application, clients of the ad hoc network backbone become middleman which is defined in XML in Figure 13. They participate in forming query driven sensor trees adaptively and backbone features are used to announce queries and return collected results.

Queries for data of interest are transmitted through the backbone in service announcement packets. Service definitions contain an XQuery predicate, a result function and query timeout period (QTP), as well as geographic region boundaries of the area of interest. XQuery predicate is used to extract the readings that interest the monitoring application. Then, if specified in the service, these readings may be processed via the given result Function. The actual query result to be submitted is the returned data from this XQuery function. XQuery specifies more than 200 functions (including functions in SQL) that include numerical processing, data aggregation (sum, avg, min, max), string operations (string-join, starts-with, ends-with), pattern matching (matches, replace, tokenize) etc. and more. By using XQuery in value fetching and processing, sensor querying process conforms to XML standard, from monitoring application down to the sensor nodes. Temporarily posed sensor drivers convert service parameters and XQuery predicate to a bitwise coded format. Coded query is sent to sensors in the payload field of setup packet. In this model, sensor nodes are assumed to be very simple equipment with limited processing and battery power. Setup packet initiates a tree topology network in the area of interest among the sensors those have accepted the query.


```
<service name = monitoring application>
<keyword attribute =validUntil>20031128193044EEST</keyword>
<keyword attribute = queryPredicate>sensor:read[value>100 and
order(value)<5]</keyword>
<keyword attribute = resultFunction>fn:concat</keyword>
</service>
```

Figure 13: XML definition of a monitoring service

DADMA: Data Aggregation and Dilution By Modulus Addressing In DADMA, a sensor network is considered as a distributed relational database composed of a single view that joins virtual local tables named Virtual Local Sensor Node Tables (VLSNT) located at sensor nodes. Figure 14 shows the distributed database perception of DADMA. Records in VLSNT, are measurements taken upon a query arrival and consist of two fields: task and amplitude. Since a sensor node may have more than one sensor attached to it, task field indicates the sensor, e.g., temperature sensor, humidity sensor, etc., that takes the measurement. Since sensor nodes have limited memory capacities, they do not store the results of measurements. Therefore there can be a single reading for each sensor attached to a node, and task field is the key field in the VLSNT created upon a query arrival. Our perception of WSNs makes relational algebra practical to retrieve the sensed data without much memory requirement, which is different from the scheme explained in [70] where the sensed data for each task are maintained at a different column in a table.

Sensor Network Database View (SNDV) can be created temporarily either at the sink, i.e., the node that collects the data from the sensor network, or at an external proxy server. An SNDV record has three fields: task, location and amplitude. While data is being retrieved from a sensor node, the location of the sensor node is also added to the sensed data. Since multiple sensor nodes may have the same type of sensors, i.e., multiple sensors can carry out the same sensing task, task and location fields become the key in an SNDV. In applications where nodes are not location aware, it is also possible to replace the location field with the local identifications of the reporting nodes. The location field can also be used to identify a group of nodes according to the aggregate and dilute functions explained below. It should be noted that SNDV is a temporary view where the results of a query are collected.

For many WSN applications, the sensed data are needed to be associated with the location data. For example, in target tracking and intrusion detection WSNs, sensed data are almost meaningless without relating them to a location. Therefore, location awareness of sensor nodes is a requirement imposed by many WSN applications. There are a number of practical location finding techniques for WSNs reported in [8].

Since each query results in a new SNDV, to keep the aggregated/diluted history of a WSN it may be needed to maintain a permanent External Sensor Network Database Table (ESNDT) in a remote proxy server. In ESNDT the records obtained from queries, i.e., the records in SNDVs, are stored after being joined with a time label. For example, a daemon can generate queries at specific time intervals or at the occurrence of a specific event, and insert the records of SNDVs resulting from

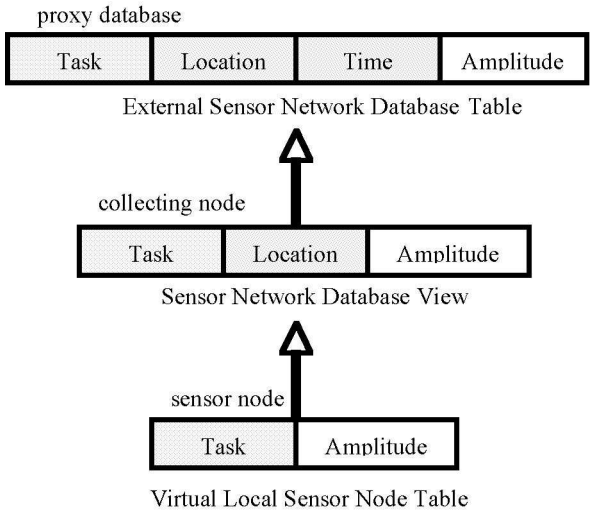


Figure 14: A sensor network perceived as a distributed database

these queries into the ESNDT. To distinguish the equal amplitudes sensed by the same node about a specific task during different periods, task, location and time fields make the key in an ESNDT.

In DADMA a statement that has the structure given in Figure 15 starts a query. This structure is largely a part of the SQL standard [10] except for the last field starting with 'based on' which will be explained later in this section. Using SQL style statements for a generic query interface has some advantages as described in [70]. Programmers and system administrators can use this practical and standard interface for all kind of WSN applications. Hardware design for WSNs can also be optimised to run this language.

In the Select keyword of the SQL statement common aggregation functions such as avg, min, max can be used to indicate how to aggregate the amplitude field. The fields to be projected from an ESNDT are also listed after this keyword. The "From" keyword indicates the nodes to be involved in the query. Any means that even a single node may be enough to resolve the query, and any node in the sensor network can do it. When the "every" keyword is used, all of the nodes in the sensor network are supposed to be involved in the query. When a task or a set of tasks is given in the query, only the sensors in the specified types carry out the measurement. Aggregate and dilute keywords are also introduced to spatially group the nodes. The 'where' keyword is for defining selection conditions according to available power and/or time and/or amplitude and/or location. The 'group by' field is used to specify the set of tasks for which the aggregation of the sensed data will be carried out. The 'based on' keyword is followed by the parameters required for the aggregation and dilution algorithm run by the sensor nodes. In [22], the distributed algorithm that process the queries in this format is explained.


```
Select   [task, time, location, [distinct | all], amplitude,
             [[avg | min | max | count | sum ] (amplitude)]
From    [any, every, task, aggregate-m, dilute-m]
Where  [power available [<|>] PA]
               Location [in | not in] RECT |
tmin<time<tmax|
amplitude [<|==|>] a]
Group by task
Based on [time limit = lt | packet limiy = lp |
                 resolution =r |   region = xy]
```

Figure 15: The structure of an SQL statement for DADMA

4.6 VF: Data Aggregation

Since Wireless Sensor Networks (WSNs) comprise large numbers of energy-constrained, inexpensive devices deployed in possibly inaccessible areas, battery re-charge or replacement is not a viable option. For this reason, energy-efficient protocols have to be developed to minimise device energy consumption, hence maximising the amount of time during which the network can be fully operational (denoted as network lifetime in the following). In such resource constrained devices the dominant component in terms of energy consumption is the wireless transceiver. This imposes the development of an energy-efficient protocol stack.

In general terms, an energy-efficient communication protocol is a protocol designed to minimise the energy-consumption associated to its operations without significantly degrading other relevant metrics performance (such as latency, throughput, etc.). Energy-conservation often involves trade-offs (energy vs. accuracy, energy vs. latency, etc.) that have to be considered when in the protocol design.

The basic building blocks to energy-efficient solutions for WSNs are the following. The wireless interface consumes a fixed amount of energy for receiving (due to the transceiver circuitry), while the cost for transmitting accounts for two components: the first is fixed and is due to the transceiver circuitry, the second component instead depends on the emission power. Depending on the transmission range of the considered technology the cost due to the fixed component may be either negligible or higher than the cost due to the emission power. Short transmission ranges (between 20-30m) have comparable costs when in idle state, or when receiving and transmitting. In these scenarios (which reflects the typical operational scenario of a cooperative object network) the network lifetime is prolonged by keeping the nodes in the so called 'asleep' state in which the interface is not operational, nodes cannot transmit or receive packets, but the energy consumption is much lower (around two order of magnitude smaller than the cost in the idle state) as long as possible. The other possible states include the idle state, in which the transceiver is operational but no communication exchange is taking place, and the transmit (receive) state in which the transceiver is operational and is currently transmitting (receiving) a packet. The exploitation of the scheduling between awake states (idle, transmit, receive) and asleep state is one of the basic tools used by energy-efficient

protocols for decreasing energy consumption. Other techniques include minimising the amount of transmitted information, avoiding the waste of resources, transmitting only when it is likely that the data will be received correctly, minimising protocol overhead, and designing awake/asleep schedules that do not increase significantly data latency. In many sensor network applications the low transmission range translates in the need of keeping the transceiver asleep as long as possible. Thus, decreasing the amount of information transmitted and the corresponding overhead is still an important knob for tuning energy-efficiency. By reducing the load it is possible to decrease the amount of time devices have to be in the awake state.

Some solutions have been developed for minimising the load of the WSNs which are based on the elimination of the redundancy in the transmitted data. In a WSN many nearby nodes detect the same or similar events, so that the information they transmit is highly correlated. Some of the relay nodes can thus buffer a few packets, and process them eliminating redundancy. This process is often referred to as data aggregation or data fusion. Data fusion reduces the amount of transmitted data (thus possibly the energy consumption) at the price of increased latency. Other trade-offs which have to be considered in the design of a good data aggregation strategy are energy vs. robustness and energy vs. accuracy. Depending on how data fusion is performed at the aggregation points the precision with which an event is reported may be compromised. It is possible to simply concatenate in the same packet the data of several packets (saving in terms of header bytes). However, the majority of the advantage that can be obtained in decreasing the load comes with techniques which combine the data. The drawback here is that this approach results in partial decrease in precision/accuracy. A second problem is that the inherent redundancy of the transmitted information makes the data delivery process more robust in presence of transmission errors. If redundancy is avoided then aggregated packets should be strongly protected to avoid their corruption or wrongful delivery. An effective solution for data fusion has to consider all these different trade-offs, so that data fusion is maximised while maintaining data latency, accuracy and robustness degradation within limits tolerable for the specific application.

Data fusion has two aspects: a scheme to select the roles of the different devices that act as aggregators (aggregation points: How many of them? Where located?), and the data fusion strategy used at these points (how to determine which data to aggregate, how long is it worth waiting or for how many packets before aggregating and forwarding, etc.). Data fusion should also be combined with other protocols (data link, routing, etc.) for further optimisation.

In the following we survey the most recent works in the area of data aggregation.

4.6.1 Types of aggregation

Packet-level aggregation This scheme works by dropping duplicate packets and by merging several packets to reduce the overhead due to header transmission. This technique can always be applied only affecting data latency. No loss occurs in terms of data precision. No compression of highly correlated data is performed.

Total aggregation In this case, all data received within a given time interval can be fully aggregated. This kind of aggregation can be realistic for given types of queries. For instance, when the average temperature in the area monitored by the networks is needed at the network collection point

(the sink), an aggregation tree could be built rooted at the sink in which each internal node can be an aggregation point. Each point has simply to collect from each of its children the average value of the temperature, and the number of samples it was obtained from. All these values are then combined to form a single packet that conveys the average temperature in the area monitored by the point sub-tree. The size of the packet which will be sent out, containing the new average and number of samples, is the same as that of the packets received at the point.

Geographic aggregation This scheme of aggregation, proposed in [2], assumes that data related to certain events can be aggregated if their sources are geographically close and the events happen within a certain time interval. This implies that data precision is maintained. (The precision here refers to both geographical place of the events and the times of their occurrence.) Such aggregation tries to model scenarios in which the sources can classify events and send a code classifying an event, together with the time of occurrence, and the location to the sink(s). With this time of aggregation, aggregation points away from the sources of the events may not be able to aggregate data. To obviate these problems, [2] proposes to enhance the method by co-locating aggregation points with cluster-heads, i.e., specific network nodes chosen in such a way that aggregation points are well-spread throughout the network.

In [115] the authors point out the major pros of data fusion. Many of the queries asked by users require to aggregate and combine the data generated by the sources (so why not to perform such processing aggregation in the network instead of at the final destination, saving energy and network resources?). In-network aggregation also allow for comparison of different measurements of a given event before transmitting the information to the sink(s). A comparison among different samples at the aggregation points allows the identification and filtering of false or inaccurate measurements performed by faulty or malicious sensor nodes.

There is a huge difference in terms of network load when answering an average or a median query: The first can be answered by delivering aggregated information, the second needs to bring to the sink(s) information on the distribution of the sensed values. The solution proposed in the paper addresses exactly this problem, proposing a representation of value distributions which allow us to answer the listed types of queries with a bounded accuracy while compressing significantly the amount of information which have to be disseminated in the network. This is achieved by defining and transmitting compressed information on the measured values distribution, denoted as quantile digest. Quantile digests can be combined by aggregation points without compromising the resulting accuracy, and can be used at the final destination to answer quantile, range and consensus queries. The amount of transmitted information is significantly reduced over the case in which all the sensed data have to be reported.

[32] describes a security problem associated with data fusion. The current data fusion process puts a great deal of trust on aggregation points. Not only a consensus-based mechanism is needed to filter out information transmitted by malicious nodes, but also a scheme has to be designed for the sink(s) to have proof of the validity of the reported aggregated information, since aggregation points themselves may be faulty or malicious. The way this problem is solved in the paper is by adopting a witness based data fusion node assurance scheme. It is assumed that there are more nodes which gather the same data and result in the same aggregated packets, even if only one aggregation point reports such results to the sink(s). Such nodes know of each other. Each node computes a summary

of the results and encrypts it sending it to the selected aggregation point which will have to transmit all such summaries together with the aggregated data to the sink(s). The final destinations will then be able to compare the summaries of the witnesses and of the aggregated data and use a voting scheme to decide if the reported data is reliable.

4.6.2 Selection of the best aggregation points

In [98] a way to select aggregation points is presented. The goal is to maximise the aggregation gain (the reduction in terms of network load achieved via data fusion) while maintaining the end-to-end latency below an application dependent threshold T. A static sensor network with a single sink is considered. The authors provide a mathematical formulation to select one single aggregation point per path in order to maximise performance. Heuristics are then derived from this formulation and integrated with WSN routing (tree-based or directed diffusion based routing). A performance evaluation shows the advantage that can be achieved by means of such way of selecting aggregation points with regard to more empirical approaches. The paper also has the merit to point out how latency paid for sake of data fusion should be rewarded by a significant amount of aggregated data, so lower or higher latency spent at aggregation points should depend on the amount of data that can be fused at that aggregation point.

In [117] the authors assume a set of aggregation points and address the problem of setting the timeouts at the different aggregation points encountered along the source-destination paths. Assumptions are periodic transmission of data from the source to the sink, and adoption of a tree-based converge-casting. Three different approaches are compared: 1) periodic simple, in which all the packets received within a time T are aggregated, 2) periodic per hop, in which an aggregation point waits to receive data from all its children in the dissemination tree before it sends the aggregated packet out (recall the traffic is periodic), and 3) the proposed approach named periodic per hop adjusted. In this latter approach the authors envision to use cascading timeouts. Not only do timeouts depend on the aggregation point position in the dissemination tree but a node timeout will happen just before its parent. In this way a cascading effect (i.e. data originating at the leaves will be clocked out first, reaching nodes in the next tree level in time to be aggregated at that level together with the data originated at that level and so on) reduces the overall latency to reach the sink. Comparative simulations among the different schemes show that, as expected, a cascading timeout based scheme allows to reduce latency without impacting the resulting received data accuracy. Similar ideas have been reported in [133] for the case in which the WSN traffic is event-based. The authors discuss the potential inefficiencies associated to data fusion in case timers are not synchronised: though the packets sent by sources are generated at similar times as they detect the occurrence of an event, unsynchronised settings of the aggregation points timers may result in an aggregation point at level i not being able to gather information from the nodes at level i+1 before performing aggregation. To avoid this problem each aggregation point starting its timer informs its neighbours triggering also their timer start, and timers duration depend of the aggregation point level in the tree. The solution proposed exploits synchronisation and different timer settings at the different levels, to allow a node at level i to gather information from the previous levels before its timer clocks out.

In [19] it is proposed to exploit clustering to select cluster-heads as aggregation points. In addition

the paper proposes to assign an interval of possible values to each metric of interest and to select critical values for each measured metric. A pattern code is then computed based on the measured values and sent to the cluster-head. The cluster-head uses this aggregated implicit information of the data gathered by the sensor node to deduce whether some of the data to transmit are redundant and to avoid redundant transmissions before they occur. Redundancy is thus eliminated by pattern code exchange between the sensor nodes and their cluster-head and by the cluster-head deciding based on such codes which sensor should send data. This also allows to avoid security problems associated to cluster heads receiving data and having to decrypt them to perform data fusion operation. In the proposed scheme, denoted EPSDA (energy-efficient and secure pattern-based data aggregation), data encryption is performed end-to-end by means of symmetric cryptography. A way to select aggregation points in a hierarchical WSN organisation is also proposed in [7], based on an ILP formulation which jointly addresses aggregation points selection and routing. In [89] the authors define a metric to evaluate the level of compression achieved via data fusion. Variants of protocols for data dissemination proposed in the literature (LEACH and PEGASIS) are then proposed in order to maximise the possibility to perform aggregation. It is shown how the extended solutions decrease network load and energy consumption without significantly affecting accuracy over the basic LEACH and PEGASIS schemes.

In [113] the authors claim that instead of grouping devices into clusters based on a distance-based criterion, the hierarchical tree-based communication infrastructure used to deliver the data back to the sink should be designed so to group in the same subtree nodes which belong to the same group. Group affiliation is attribute-based and reflects the possibility to answer similar queries and to lead to possibly redundant, merged data (in addition to the affiliation to the same group devices are organised into a tree also based on a distance criterion so that highly correlated data are likely to have such redundancy eliminated close to the sources). This groupware based tree results in significant increase in terms of capability to effectively aggregate data. In addition the TINA (temporal coherency aware in Network Aggregation) scheme is adopted. New data are reported only when they differ significantly from the last reported data (how significantly is an application-dependent factor which is communicated via interest dissemination). Overall the two schemes combined lead to significant energy saving.

In [41] an adaptive scheme to determine how many packets to wait for (and how much time to wait for) before performing aggregation is proposed. First the authors propose a solution in which, instead of waiting for a fixed amount of time or a fixed amount of packets, an aggregation point simply tries to exploit the availability of the wireless channel at most. If the channel is available the aggregation point transmits a packet aggregating the queued packets. This reflects for example the following way of reasoning. If data aggregation and awake/asleep schedules are decoupled there will be some times in which the aggregating device has its transceiver operational and in which the channel is free. Transmitting or delaying transmission does not significantly change the energy consumption (as the costs associated with transmitting or being idle when the transceiver is operational are similar in these networks), but can severely affect the end-to-end latency so it may be convenient to transmit anyway. This scheme is named the on-demand scheme. What is not considered in this scheme is that a single device decision to transmit may impact the possibility of another device to do the same in a CSMA/CA based network. What would be interesting to have is a dynamic scheme which adapts its decision to make the MAC layer operate at a desirable operational point, limiting on one side the

amount of transmitted data (aggregating several packets in the queue in each packet sent out) but also maintaining a low latency. This is the idea behind the proposed dynamic feedback scheme which adopts a dynamic controller to achieve this goal. Simulation results show that schemes which rely on fixed thresholds the proposed solutions achieve the best compromise between decrease of network load and latency. The dynamic feedback scheme is particularly valuable as it is able to operate efficiently under varying network load conditions.

4.7 VF: Resource Management

Generally speaking, COs are battery powered devices that usually lack resources, such as CPU, memory, and power, to name a few.

COs may form a self-organising network, in which COs arbitrarily join and leave the network or even move during an operation. Since, system components are distributed, any action often involves multiple COs at a time. Due to the distributed, dynamic, and uncertain (since both COs and communication are associated with uncertainty) nature of system components, the design of such embedded wireless collaborative systems prove to be difficult and requires an scheme to facilitate the system design and application development.

Resource management aims at providing a way to manage the resources of a system in a way by enabling high-level system primitives to hide unnecessary low-level details. It enables the application to be independent of the actual underlying distributed system. It should also address the dynamic nature of available resources, such as variable network bandwidth. Because the system is built of fault-prone components, failures should be handled as normal and not as exceptions.

4.7.1 Design challenges

Traditionally, *system software* aims at providing a unified way of using underlying resources. System software operates the system and manages available resources in a way that provides higher-level services that are usually required. System software consists of three major components:

- *Operating system (OS)*: the OS aims at providing a set of services that are generally needed by application developers, such as starting and stopping processes or allocating memory and other resources. These services make the system much easier to manage, since the actual low-level details are kept hidden from the application developer. The operating system provides a unified interface to use resources and also manages its allocation among incoming requests.
- *Protocol stack*: communication software is decomposed into a set of standardised layers, referred to as a *protocol stack*. Different layers provide different abstraction levels of generally needed networking services. Layers use some of the underlying services to provide some higher-level services. It aims at hiding the heterogeneous low-level details of the network. The standardisation of these layers played a significant role in the success of Internet deployment.
- *Middleware*: it is a reusable system software that bridges the gap between the end-to-end functional requirements and the lower-level OSs and protocol stacks. Thus it is a piece of software running in each member of the network, and its goal is to provide high-level services that are independent of the heterogeneous underlying systems.

The layered design has proved to be a very successful approach to design complex systems. Defining high-level interfaces to easily use underlying complex functionality makes application development easier, but unfortunately makes the tuning of *non-functional properties* extremely difficult. These properties usually remain spread across several system layers, thus, it is impossible to make modifications to them in a single place. Energy-efficiency or fault-tolerance are such non-functional properties. Optimisation issues, such as energy-efficiency, usually require special details that may be hidden by the interfaces. Higher layers can make better decisions by working together with the operating system, and sometimes it is beneficial to influence the behaviour of the OS to meet some higher level needs. Optimisations in separated layers may not achieve a global and synchronised goal. A well known example is the unfortunate action of the operating system to put the system into stand-by while the user is giving a presentation.

In networking, the layers of the protocol stack are defined in a way to let them be developed independently. However, a number of dependencies and redundancies among different protocol layers make the communication *energy inefficient*. In [40] and [28], in which the cross-layer nature of low-power ad hoc wireless network design is characterised, it is shown that cross-layer design is more energy efficient.

Therefore for energy-efficient design, the traditional strict modularisation or layering is not appropriate. For example in sensor networks, the monolithic design of communication software is common to reach the required energy-efficiency needs. However such a design makes system development and management very difficult. In the next sections we will present solutions that allow both development and management easy, and at the same time allow for optimisation of non-functional properties.

4.7.2 Adaptation in Resource Management

In embedded wireless collaborative systems, the availability of the underlying resources is highly dynamic. For example the networking conditions may change frequently, thus, the available bandwidth and the certainty of communication varies. Such a change may influence the chosen networking layers or possibly would change the whole behaviour of the application. The traditionally applied approach of a resource request and its allocation does not provide the needed flexibility any more [94]. The system must adapt to changes in the surrounding physical and virtual environment. Instead of allocation, a *resource negotiation* should take place and allocation of available resources should be adaptive. *Adaptation* is the ability of a system to change its behaviour in response to environmental changes.

Early experiments highlighted that in the case of dynamic underlying resources, the ultimate goal of resource-management is to combine adaptation with allocation. It is not beneficial to handle the system as a transparent entity with fixed services; instead the system should allow the negotiation of resource demands and supplies. To manage such dynamic resources, the system software should meet new requirements. It should drive the negotiation of resource demand and supply together with the applied mechanisms based on the environment.

4.7.3 Adaptation and Enabling Technologies

Over the years, various technologies have contributed to the evolution of adaptation and reconfigurable software design [5, 76]. Since the main aim is to enable the software to reconfigure itself dynamically based on some conditions, the core of the majority of approaches is the interception and redirection of interactions among software entities.

Middleware Middleware is defined as a piece of higher-layer software present in each member of a possibly heterogeneous network [29]. Its purpose is to provide high level services to make the development of networked applications easier by hiding details of the underlying infrastructure such as operating system and network protocol details. Since it represents another level of indirection and transparency, it is a straightforward way to provide adaptive behaviour.

Middleware is commonly divided into four layers [111]. *Host infrastructure middleware* is directly on top of the operating system and provides a high-level API to hide lower level heterogeneity. *Distribution middleware* provides high-level programming models, such as remote objects, enabling the developers to use local and remote objects in a similar way. *Common middleware services* define higher-level domain independent components to help managing resources. Finally *domain-specific middleware* is usually designed especially for a particular class of applications. Any of these middleware layers may be a suitable place to provide adaptive behaviour.

Many of the adaptive middleware platforms work by intercepting and modifying messages to provide a level of indirection to influence behaviour of the application. Most of the adaptive middleware approaches are based on some popular object-oriented middleware platform such as CORBA or JavaRMI. A huge number of adaptive middleware approaches exist, their taxonomy can be found in [77].

However middleware based adaptive frameworks may not be sufficiently flexible; their application is straightforward from the software development point of view. These solutions may provide a way to extend heterogeneous, layered systems with the ability to adapt.

Component-based design Software decomposition to separate modules has become a fundamental approach to make the development of reusable components independent of each other. Software components are reusable software entities that third parties can develop independently and can easily be composed and deployed as general components in the future [122]. Popular platforms include COM/DCOM [79], .NET [78] and Enterprise Java Beans [120]

Well-defined interface specifications enable service clients and providers to be developed independently, however, it turns out that the specification of the functional interface usually does not provide enough flexibility to reuse the component. The details of how the component would behave in certain conditions are usually crucial to design choices. To trust a component, specific guarantees should be available regarding to its non-functional properties. These guarantees are called *contracts* [15].

Contracts may be categorised to four levels. Level 1 is the basic or syntactic level that defines the functional interface needed to use the module. Level 2 provides behavioural level contracts, which gives more information about how the functionality reacts in different conditions. Level 3 is the synchronisation level, which describes synchronisation properties required in parallel environments. Finally, level 4 describes quality-of-service properties, such as response delay.

The composition of modules can be categorised into two large groups. In *static composition*, the developer chooses the appropriate composition at compile time and it cannot be changed later in run-time. Contrarily, *dynamic composition* makes it possible to add, remove or reconfigure components during the runtime operating. *Late binding* is the technology that enables coupling of compatible components at run-time.

The system may also support multiple different versions of a given component with different contracts. In such cases, this may be an enabling technology to adaptively select the versions that are the most appropriate for the given environmental conditions.

Computational reflection Computational reflection is the ability of a software system to reason about and possibly alter its own behaviour [73]. Reflection exposes a system's implementation at a level of abstraction that contains enough details and enables necessary changes. The ability of a system to observe and change itself is a key enabling technology towards self-modifying code.

Computational reflection involves two activities. *Introspection* allows the system to observe its own structure or behaviour. This may involve revealing its structure or evaluation of software-sensors to collect statistics. The other main act is *intercession* that enables the system to act on the observations and make necessary modifications. This may be the replacement of a module, or installation of new monitoring components.

In reflective systems, the actual *base-level* software includes its abstract self-representation as a special object. This is called a *meta-level* representation, which represents the actual objects of the base-level. The two levels should remain causally connected, remaining consequent during the changes made.

The direct exploration of the meta-level would provide many details about the base-level objects, but it also makes the process difficult and error-prone. A *meta-object protocol* (MOP) [57] is a higher level interface that enables easier introspection and intercession by enabling systematic access to meta-level objects. MOP can be categorised as enabling either structural or behavioural reflection. *Structural reflection* reveals inner structural details, such as class hierarchy, object interconnection and data types. In contrast, *behavioural reflection* focuses on higher-level computational semantics of the application, such as the application of a new mechanism.

Reflection may be provided by a programming language in a native way, such as in Lisp [116], Python [95] or in some Java derivatives such as Reflective Java [131]. Even if the programming language does not directly support reflection, similar functionality can be implemented in a higher level, using a middleware platform [16], such as KAVA [127] or OpenCorba [66]. Instead of programming construct representation, reflective middleware deals with the self-representation of middleware services.

Such reflective abilities of a software system obviously play a key role to provide adaptation.

Separation of concerns Over past years, some major difficulties in software decomposition have been revealed [86]. Consequently, it was realized, that the same functional behaviour can be implemented with several different decompositions, which are completely different when future changes have to be made.

Separation of concerns [1] is a software development technology that aims at separating the development of the application's functional behaviour (business logic) from other non-functional re-

quirements. The non-functional concerns are usually cross-cutting concerns spread across several modules, such as QoS, energy consumption or fault tolerance. This approach enables the separate development and maintenance of concerns, while providing a straightforward opportunity for adaptation.

The most widely used approach is *Aspect Oriented Programming* (AOP) [59]. It includes abstraction techniques and language constructs to manage different concerns of the system separately. The codes implementing different concerns, called aspects, can be developed separately. *Pointcuts* are placed into the functional code to sign the locations where different aspects may be woven. Finally a specialised compiler called *aspect weaver* is used to combine aspects with the business logic of the application (figure 16). Examples of such AOP frameworks are the AspectJ compiler [58] and KAVA [127].

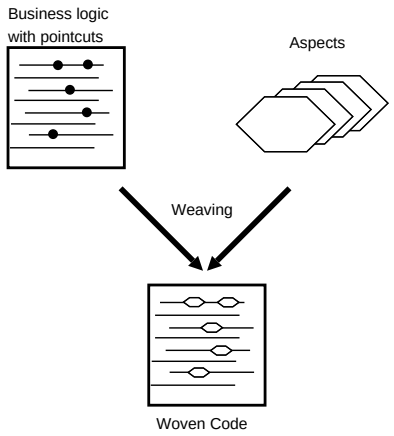


Figure 16: Aspect weaving in AOP

Composition Filters [14] is an approach that dynamically intercepts messages that are sent and received by components. Filters can be applied to all input and output messages or can select particular messages. Such filters are developed independently and can be compiled into the source code or can be preserved as run-time manipulation objects.

Similarly, aspect weaving at compile-time results in a tangled executable, which is not reconfigurable later. However some recent approaches delay the weaving process till link- or run-time [124], thus providing a way to tune different aspects of the system during execution.

4.7.4 Adaptation frameworks

This section introduces some of the most important cross-layer adaptation frameworks and projects. These frameworks assume mobile devices that are more resource rich than wireless sensor nodes, while in the project COs are of interest. However, these frameworks present interesting adaptivity concepts that could be at least partially implemented in smaller devices to achieve a desirable degree of resource management.

Odyssey *Odyssey* [85] was one of the early attempts to demonstrate *application-aware adaptation*. It demonstrated how application-awareness would result in enhancements of system performance.

Odyssey supports the change of application quality based on low-level system conditions. When the system cannot guarantee the required supply, the application should regulate its demand. The project demonstrated the effectiveness of the approach by scaling down video quality in case the bandwidth decreased.

The key point in Odyssey was to extend the operating system resource allocation API with resource call-backs. Applications register a tolerance window in Odyssey and receive a callback when the required resource levels cannot be kept within the tolerance window. At that point, the application can adapt and the new resource needs may be registered again. Only minor changes are needed to port applications to Odyssey, and it is demonstrated that in some cases it is also possible to avoid any changes.

In [82] Narayanan extends Odyssey to achieve a cross-layer adaptation solution especially to support mobile interactive applications. These kind of applications require low response times and long battery life, while the available resources are varying. Their work introduces the term *fidelity* as a metric of accuracy. *Data fidelity* is the extent to which the degraded version of a data object matches its reference version, on the other hand *computational fidelity* is a runtime parameter of an algorithm that can change the quality of its output. The framework uses *multi-fidelity computation* to achieve adaptive behaviour.

A major difficulty addressed in their framework is to bridge the gap between the three independent parameter dimensions:

- user satisfaction metrics (utility)
- application parameters (fidelity choice)
- resource supply and demand of the system (resources)

The presented solution uses *resource supply predictors* to predict the available amount of system resources, such as CPU availability. *Performance predictors* give performance metrics, such as latency or battery lifetime, as a function of resource supply and demand, and *resource demand predictors* compute application resource needs as a function of fidelity and non-tuneable properties.

Some of these predictions are application dependent. This is solved by providing an application dependent binary module called *hint module*. Moreover the system has to be aware what fidelity values are possible for the given application and what are the actually used non-tunable parameters. These are declared in the *application configuration file*.

The architecture of the run-time system can be seen in figure 17, note that only the hint module is application-dependent. The central component that drives the adaptation is the *solver*. Its task is to search the state space of tuneable parameters and find a set of values that maximises the utility. When the application starts, it passes in the values of non-tunable parameters. A set of generic supply predictors predict the applications' resource supply for the near future. The solver searches the space of tuneable parameters for the best possible combination.

The evaluation of candidates is done by computing their utility, which is application-dependent. The solver sends the tuneable and non-tunable parameters as well as the resource supply prediction to the utility function. The utility function invokes performance predictors to compute latency and battery

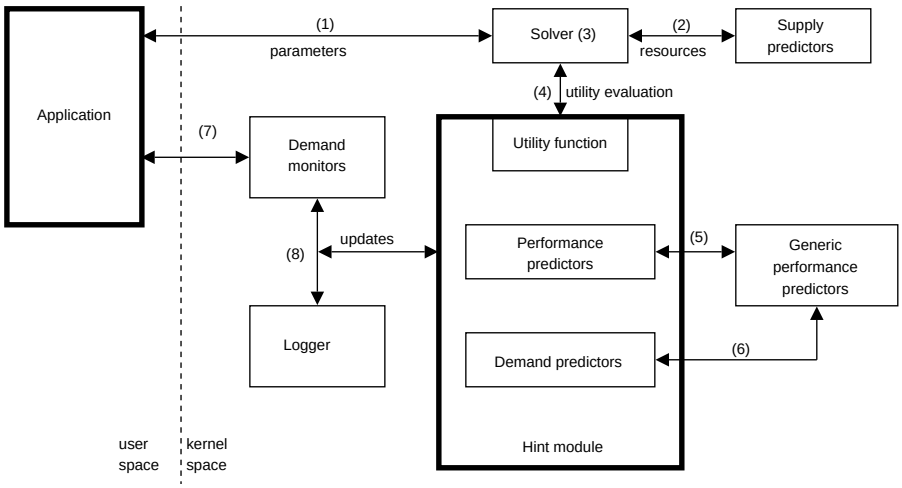


Figure 17: System architecture for multi-fidelity API

drain for the particular choice. Performance predictors call application-specific resource demand predictors to compute performance.

Resource demand monitors compute the resource demand of the operations performed. The logger records these values together with the applied parameters to update demand-predictors based on online learning techniques.

The effectiveness of the framework is demonstrated by several experiments ranging from augmented reality to speech recognition.

RAPIDware The aim of the RAPIDware project [96] is to design an adaptive, component-based middleware for dynamic mission critical systems. These systems must continue to operate correctly even during exceptional situations. Such systems require run-time adaptation, including the ability to modify and replace components, in order to survive hardware component failures, network outages, and security attacks. RAPIDware is a significant research effort, which applies the combination of computational reflection and aspect-oriented programming to support software recomposition at run-time.

Sadjadi et al. [105] present an adaptable communication component called *MetaSocket*. It is an extension of a regular Java socket to provide adaptable communication services. A metasocket is created using Adaptive Java [55], a reflective extension to Java.

To convert an existing Java class into an adaptable component two main steps are required. Figure 18 illustrates the two steps on a multicast socket component, which enables it to adapt to meet different QoS needs. The first step is called *absorption*, which transforms the class to a base-level Adaptive Java component. This base-level class has a set of mutable method invocations, which expose the functionality of the absorbed class. In the example, the absorption steps create a send-only multicast socket, which implements only the *send()* and *close()* adaptable invocations. Links between invocations and the methods are highlighted by lines in the figure. The following step is

called *metafication*, which enables the systematic investigation and changing of base level invocations. In the example, this means to insert or remove filters to the multicast stream, such as different error-correction filters. By this, the socket component is adaptable to meet different quality requirements.

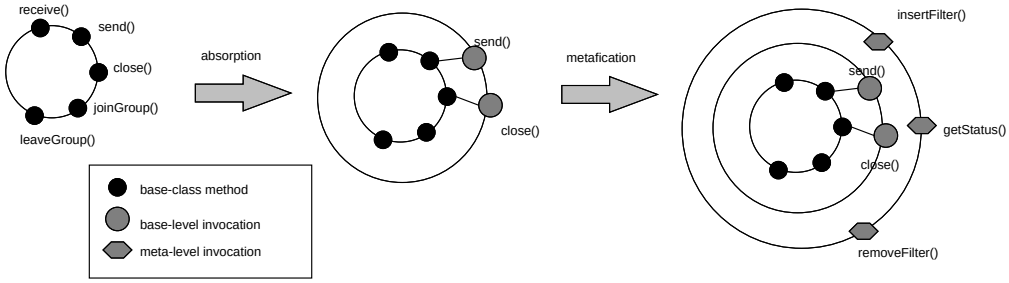


Figure 18: Metasocket absorption and metafication

These steps use TRAP/J [106], a Java implementation of TRAP, to weave in the required adaptive behaviour. The framework takes an existing Java class together with the list of required adaptable invocations as its input, and makes the required adaptable class as its output. It uses the AspectJ compiler [58] to generate the adapt-ready version of the application.

A MetaSocket enables the middleware or an application to monitor the communication status and to insert, remove and configure adaptation filters at run time. With the help of MetaSockets, a kernel-middleware interaction framework was provided in [108] to support both horizontal and vertical cooperating of components. The Kernel Middleware Exchange (KMX) project addresses the interaction between the middleware and operating system layers to support such an universal adaptation. This approach is especially well-suited to mobile ad hoc networks and overlay networks, where a set of nodes collaborate to accomplish collaborative services.

This framework thus provides an interesting mixture of aspect-oriented programming and reflection to easily turn software components to adaptable ones. Their work also points out, how an universal adaptation framework can be feasible by using such components.

GRACE The Illinois GRACE project [4, 103] aims at providing an adaptation framework for saving energy in mobile multimedia systems. In the considered system all the system layers and applications are allowed to be adaptive. These adaptive entities cooperate with each other to achieve a system-wide optimal configuration in the presence of variations in the available resources and application demands.

One of the main questions addressed in this research is how to achieve global adaptation with an acceptable overhead, since adaptation with global scope potentially results in larger overhead. GRACE supports three levels to provide a balance between the scope and the temporal granularity of adaptation:

Global adaptation is invoked only infrequently, in response to large changes in resource supply or demand. It considers all applications and all the system levels. A global coordinator per-

forms the resource allocation by searching through the space of all possible configurations and evaluating the overall utility and resource usage for the particular choice.

Per-application adaptation considers only one application but all system layers at a time. At this time, the accuracy about the resource demands of the actual application is more accurate, thus making finer granularity decision possible.

Internal adaptation adapts only a single system layer or application at a time. It does not need to consider the cross-product of configurations, it can be very efficient. Unfortunately, the system may remain only in a local minimum of optimality.

Another main question this framework addresses is how to make the prediction of future demands and supplies. All the adaptation levels require predicting the resource demands for each application. Global adaptation requires long-term prediction, while per-application adaptation requires prediction only for the next job. For the short-term, history-based prediction techniques are commonly used.

The prediction is not straightforward, since the resource usage depends on the actual system and application configuration in use. The approach in GRACE to address this problem is to divide the prediction problem into two parts. The *system-independent* part can be predicted entirely by the application. The other is the *system-dependent* part, which is handled by the specific system layer. For example the CPU time demand of a job can be divided into the number of instructions and the time per instruction parts. The former is independent of the system, while the latter depends on the actual configuration.

GRACE provides an architecture for the optimisation process as well. This framework controls when different levels of adaptation are triggered and also finds the optimal configuration. In the proposed solution, local and global adaptations together allow the system to find a better configuration. The simulation-based evaluation shows the benefits of the solution.

DEOS The DEOS project addresses the problem of how to best meet the dynamic Quality of Service (QoS) requirements of end users in face of dynamic changes in the underlying hardware/software platform's resources. However the project mostly focuses on QoS issues. The proposed solution involves both the system and the application.

In the project, *Q-fabric* [90, 91] is presented. Q-fabric is a kernel level abstraction for cooperative, distributed resource management and adaptation. The solution is based on a kernel-level event notification service, which follows the publish/subscribe paradigm to exchange specific events. Such events may be related to monitoring activity, or can also be the trigger event of an adaptation process. Since the event service is fully anonymous, it is a flexible way to let separate local resource managers cooperate.

A resource manager can be seen in figure 19. Its task is to distribute system resources among a number of competing applications based on a policy. The monitor entity keeps track of changes in resource allocations and keeps statistics, and it also submits monitoring events. The adaptor receives such events and decides about triggering changes either in local or remote resource allocations.

The approach can be combined with middleware level mechanisms [92]. By combining a middleware with such a kernel event service allows the flexibility of a global resource management. Such a resource management framework may also be important to coordinate adaptation.

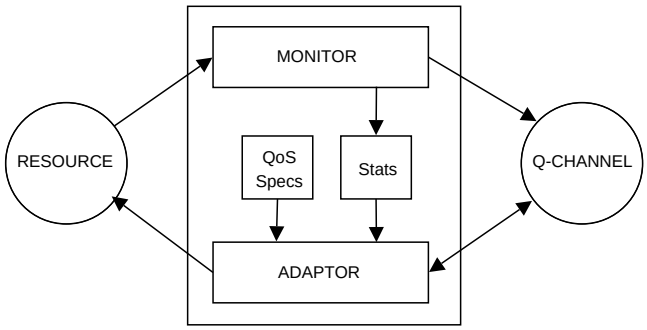


Figure 19: A resource manager

ECOSystem Zeng et al. [134] presents ECOSystem (Energy-Centric Operating System), which is an approach to let the operating system manage energy as a first-class system resource. It manages energy consumption at the OS level and also influences applications in their energy consumption.

Their energy-centric resource management is based on *currentcy* (current currency). Currentcy is a model reflecting the relative price of the energy, and thus influencing the amount the system can spend on. Currentcy reflects the ratio of supply and demand and provides a way of efficient resource management in case of resource shortages.

The project has demonstrated the effectiveness of the approach in a Linux environment. Its basis is the accounting of energy consumption of processes. The system specifies each tasks' relative importance and provides an energy-conserving allocation strategy as well.

Neugebauer et al. [83] introduces a similar approach, it applies energy pricing as well to use ideas from microeconomy to efficiently manage resources. In the presented model, a price is calculated for resources that are congested. These prices capture the ratio of demand and supply. These resources are charged afterwards, thus price represents a meaningful feedback to applications.

The project uses the Nemesis OS [51]. The Nemesis OS shifts many functionalities (e.g. protocol stack) to the user level, thus avoiding anonymous resource allocations. This makes more accurate resource accounting possible.

Both of the approaches use the operating system to account resource prices and to figure out the relative usage costs. They enable a distributed solution to resource management and adaptation.

4.7.5 Adaptation categorisation and its parameters

One criterion for classification of adaptation is whether the system is capable of the free recomposition of its components or just uses some built-in settings. *Parameter adaptation* is the modification of variables that influence the behaviour of the component. This requires built-in support for the given parameter and also the interface to modify it. In contrast, *compositional adaptation* involves even the adaptation of new algorithms and the run-time recomposition of the software. The latter enables a very general ability to adapt, however, the former is much easier to implement. In the former, the time of composition may entirely influence the flexibility of the solution. Development time adaptation results in a hardwired ability to change behaviour. Composition may occur at compile, link, or even

load time, however they all result in a *static* composition. In contrast, runtime adaptation results in a completely *dynamic* system.

To choose the most suitable adaptation approach, following parameters should be taken into account:

- *The best configuration*: when the adaptation of the system occurs, the goal is to find the most appropriate configuration for the new environmental conditions. In the majority of cases selecting the most optimal configuration leads to further complications. When investigating the possible candidates, the outcome of choices should be predicted by the system. This is not easy even in a simple case, moreover a number of adaptable components and all their available cross-compositions may result in a huge configuration space. The system should search through this space and figure out the new configuration. This may result in a significant overhead.

- *Scope of adaptation*: to limit the number of available configurations, it is crucial to investigate the scope of adaptation. A global adaptation involves all the adaptable components of possibly several networked entities. The scope may be limited to *(i)* a node, *(ii)* to one of its applications, or *(iii)* even to a single module. Unfortunately, limiting the scope may result in the miss of the best configuration.

- *Time of adaptation*: an important choice is when the adaptation should occur. Adaptation may happen continuously, however in more complex cases it may result in a large overhead. A way to avoid this is to allow adaptation on a periodic basis, however this may still result in a large amount of unnecessary overhead. The most sophisticated approach is to do adaptation on a reactive basis. This means that the occurrence of some conditions would trigger the system to find a more optimal configuration.

- *Resource supply and demand*: there may be severe reasons for a need to change in the behaviour of the system. To capture these conditions, some monitoring functionality is required. Continuously monitoring a large number of quantities may result in a significant overhead, so it is better to monitor only those parameters that are actually necessary. Conditions that may trigger adaptation may the result of a change of either resources or demands.

  The triggering event may be related to some *hardware* level resource changes. For example the change from AC to battery supply would result in changes that the application would do. On the other hand, a significant change in the *application* resource needs may also trigger an adaptation, since it would influence the applied allocation strategies. In many cases, changes in *network* conditions should result in an adaptation as well. Network bandwidth is the classical example of a highly dynamic resource that highly influences services that higher layers may use. Another and more general category of operating conditions is the *context*, for example the detection of the user or location in which the system may operate in.

- *Agility*: an important property of adaptation is the speed at which it can occur. A straightforward approach would be to let the system react as quickly as possible, but other problems may arise. It is possible that the new configuration is valid only for a very short time, thus the system would never settle down in a stable state. For example when resources change very quickly, it is better

to select a safe parameter set and not to follow the bandwidth changes. The adaptation should result in a stable configuration.

- *Place of redirection*: the basis of most adaptations is the redirection of module interactions. The question arises where the redirection should occur. The entity that drives the adaptation may be built entirely in the operating system, but may also lay outside and handling the entire OS as an adaptable layer. A large number of higher level solutions are available that push the interception and redirection of modules up to the middleware.

4.7.6 Future direction of adaptivity in WSN

DCOS (Data Centric Operating System) [34] is an operating system for wireless sensor networks. In these networks, the resources on the nodes are extremely constrained. DCOS was designed to fit inside the limited memory of a sensor node while being able to provide real-time guarantees and energy-efficient operation.

Sensor networks are usually driven by events, which can range from environmental phenomena to software events. DCOS applies the so called *Data Centric Architecture*, where data is the main abstraction of events. The system is built of software components called *Data Centric Entities (DCE)*. These entities produce data and they can be triggered by other data. Therefore, the operation of the resulting system depends on what kind of data types are produced and what entities are triggered.

The central element of this architecture is the *Data Centric Scheduler*. It is the component that keeps track of all the entities inside a node and decides which ones to activate. Moreover, this element is responsible for managing how components should be connected and enables the data flow between them.

DCOS is a dynamic system, which is able to adapt its functionality. Several DCEs may exist with the same behaviour, and possibly several sequences of DCEs result in the same functionality. DCOS supports run-time reconfiguration to adapt its functionality to create the most efficient configuration.

Thus, the architecture provides a simple and straightforward way to indirect software interactions. This ability is the main enabler of adaptive behaviour. A next step to have adaptive behaviour on a WSN-wide scale is to extend the concept of Data Centric Architecture to the whole network.

4.8 VF: Time Synchronisation

Time synchronisation is an important vertical function. Applications need to establish a common sense of time among the cooperating objects participating in their sensing and actuation goals.

Forest fire monitoring is a scenario that requires not only the information of whether there has been an indication of fire but also where and when this is happening. The collected sensor data provides the basis for the decision making process which may trigger actions to be taken in the monitored environment in order to address abnormal circumstances. Thus, the true time of the observed events is crucial to the prompt action of fire fighting.

It is sufficient to associate local clock timestamps with the sensor data when decisions can be made locally. For instance, the decision of regulating the fluid flux of an industrial pipe may be taken locally and actions would follow through local CO actuators, for example, a regulating valve.

However, there are cases where the collected sensor data will be aggregated from various cooperating objects at various points inside the network. To perform such a function, the data need to be timestamped at the source.

In this case, the data originates from multiple sources which may or may not agree on a similar time. Thus, a common view of time among the COs is an essential aspect to correlate multiple sources of sensor data in order to produce a consistent and reliable result.

We list two time frameworks that could suit the CO applications:

- *Relative time:* the sense of time is established with respect to an agreed time reference which could be an elected CO among a specific group. This is applicable to scenarios where preserving the order of events is the only required function from a time synchronisation service.
- *Absolute time:* in other cases preserving the order of events is crucial but it is not the only functionality needed. Applications might require event data to be timestamped when they occurred with an absolute time value with respect to a true time standard such as the universal coordinated time (UTC).

Messages containing the event data could take different routing paths in the network to be transmitted from the CO sources to the sinks. When the application requires strict order preserving, relative time could be used. It is important to note that absolute time is applicable to scenarios where the applications need both order preserving and a sense of true time.

The measurement of structural vibration in a building requires that the data collected from the instrumented cooperating objects be associated with an absolute timestamp. As it is important to correlate the time of day of the happenings in this case, the clocks of the COs must be synchronised with each other with respect to a true time. This would allow engineers to analyse data regarding the entire building structure and from the neighbouring buildings.

Some of the challenges that impact the design of ad-hoc time-synchronisation protocols are discussed in [102, 121]. We present a summary of these issues here:

- Energy efficiency: sensor network applications often require nodes to be small and cheap. The device size poses physical limitations to the amount of energy that can be stored in a battery using the current technology. In some scenarios the charging of a depleted battery or the replacement of discharged ones could be extremely difficult and expensive. In some oceanography monitoring applications where the physical access to the deep water is through a boat expedition the COs deployed should strive to be energy efficient. Thus, it is important that the time synchronisation service strive to achieve energy efficiency. This brings the question of whether GPS devices often used for time synchronisation on the Internet could be adopted in inexpensive sensor networks. The issues to be addressed are cost, size and power consumption of GPS receivers.
- End-to-end latency: in some scenarios, the network may be partitioned because of changes to the network topology (e.g. mobility of nodes or addition/removal of COs). In this case, the latency between any two points in the network may be difficult to estimate because of the large variations in latency. This can introduce problems to the design of protocols that rely on stable latency characteristics of network paths to achieve high and predictable accuracy.

- Infrastructure: in some applications the network will be created in an ad-hoc setup. Although such foreseen applications would represent a niche market today, self-organising protocols that can create an overlay for time synchronisation should be considered in the design space of protocols. The Network Time Protocol (NTP) [80] hierarchical structure may not be easily adapted to this type of ad-hoc network topology.

- Configuration: a few human operators will be responsible for a large number of cooperating objects possibly in the order of thousands. The configuration usually carried by system administrators who manually specify the time servers for each node could be a serious problem. The dynamics of this scenario demands a more automated system configuration than the one found in time protocols such as NTP.

An important question to ask is whether the Network Time Protocol (NTP) [80] currently in use on the Internet could be adopted without further modification in low cost wireless sensor networks. This protocol has been used for more than ten years and proved to work reasonably well on the Internet with thousand of nodes synchronised worldwide. The protocol is well known in the research community and we refer the reader to a text book [29].

In summary, the NTP relies on an external time reference to synchronise the top layer of time servers called *stratum 1* servers. Servers in the second highest layers are called *stratum 2* and so on. The primary source of time in NTP is often the time pulse received with a 1 Hz frequency by a GPS device locally attached to a server.

As the NTP protocol has been originally developed considering other design issues, some of the challenges discussed above cannot be addressed if the protocol is adopted unmodified in sensor network applications. One of the issues is energy efficiency as the external source of time (e.g GPS) tends to consume more power than other components of a sensor node and does not work in indoor areas which have no line-of-sight to the satellites. This is certainly not an issue in the case of the Internet where the NTP server is a standard PC plugged to a power socket.

However the issue raised in [102] that the NTP protocol maintains a synchronised system clock by regularly adding small increments to a system counter is an important one. This behaviour precludes the sensor node processor of being switched to a power-saving idle mode. In addition, NTP servers must be prepared to handle synchronisation requests at any point in time. This could potentially introduce a problem as the communication module of a CO (e.g. radio) shows the highest cost among the other components when it is transmitting or listening to any signal from the network. Thus, this module could not be put in sleep mode in order to save energy.

GPS devices of small size and low power consumption have been developed and are now commercially available. The Leadtek 9546 low-power GPS receiver [65] was recently integrated into a sensor board of the Crossbow Mote [52]. These devices consume a current of 60 mA at 3.3V in full operation. Despite the fact that such a consumption profile is higher compared to the micro-controller current draw of 8mA, it is comparable to the energy consumption profile of the radio module.

However, such sensor nodes are optimised to operate in sleep mode most of time. If the GPS is configured to operate occasionally to acquire the time signal, its energy consumption may not be an issue. Although adding this particular GPS module to a sensor board increases its cost by 50% today, we believe that the cost will drop significantly because of economy of scale. In contrast, this

is also the case for the current cost of sensor nodes which are similar to the cost of a sensor board equipped with GPS.

We list below some open issues with time synchronisation in large-scale distributed cooperating objects and wireless sensor networks:

- The characteristics of the network topology in ad-hoc wireless sensor networks may introduce severe delays because of disconnected parts of the network. To compensate for such uncertainties, the time synchronisation protocol needs to be *tolerant* to the message delay irrespective of its dominant source (e.g. processing, transmission).
- The clock drift in cheap oscillators could have values ranging from 10ppm to 100ppm which represent a drift between 0.036 seconds (10ppm) and 0.36 sec (100ppm) every hour. Therefore, the update frequency used in time synchronisation protocols needs to be carefully considered with respect to the characteristics of different types of hardware.
- The scope of time synchronisation is rather important. Some applications require ordering of events occurring within a specific area, which means that only a subset of nodes deployed in that area may be synchronised. However, other applications will need to correlate events collected from different areas in a global scale.
- When some sensor nodes become responsible to provide the time reference to other nodes in the network, the issue of trusting the source of time arises. Also, if GPS time signal is used in some reference nodes a related question is how we can guarantee the authenticity of the GPS time.
- Romer et al [102] suggest that calibration of sensors is a much more general and complex problem than time synchronisation. The interesting question is whether the current proposed time synchronisation techniques can be used for general sensor calibration problems. Also, it may be the case that time synchronisation (at least for outdoor sensor deployment) can be achieved by instrumenting low-cost, low-power GPS devices to sensor nodes as previously discussed.

5 Summary and conclusions

This section presents summaries of the vertical system functions surveyed in this study. The summary of each VF follows a similar format. An overview of the VF is given, which is then followed by a discussion of identified trends and relevant open issues.

The study WP3.1.1 (Applications and Application Scenarios) discussed a list of characteristics and requirements of selected cooperating objects applications. Table 5 relates each of these requirements to the VFs. The reader should refer to the document WP3.1.1 for a comprehensive discussion on selected cooperating objects applications and their requirements.

Characteristic/requirement (WP3.1.1)	**Vertical System Function**	**Section**
Network Topology – topologies to support either single or multihop communication.	Communication; Distributed storage and search	4.3; 4.5
Scalability – necessary system support to growing number of COs.	Distributed storage and search; Data consistency	4.5; 4.2
Fault Tolerance – provides the mechanisms for supporting the resilience of the system in failures.	Data consistency; Communication; Security	4.2; 4.3; 4.4
Localisation – determining a node's location is the basic required functionality for various applications.	Context and location management	4.1
Time Synchronisation – COs need to establish a common sense of time.	Time synchronisation	4.8
Security – this is pervasive and must be integrated into every system component to achieve a secure system.	Security, privacy and trust	4.4
Data Traffic Characteristics – the system should provide support for various types of application traffic.	Communication; Distributed storage and data Search; Data aggregation	4.3; 4.5; 4.6
Networking Infrastructure – CO networks can be infrastructured or infrastructureless (ad hoc).	Context and location management; Communication	4.1; 4.3
Mobility – the physical components of the system in some applications may be static whereas in others, the architecture may contain mobile nodes.	Context and location management; Communication	4.1; 4.3
Node heterogeneity – most of CO applications include nodes that have distinct hardware and software factors.	Data lookup mechanisms; Data consistency; Data aggregation	4.5; 4.2; 4.6
Power awareness – power consumption is one of the performance metrics and limiting factors almost in every CO application.	Communication; Resource management	4.3; 4.7
Real-time – the system delay requirements are very stringent in real-time applications. The broad meaning of delay in this context comprises the system data processing and network delay.	Resource management	4.7
Reliability – guarantees that the data is properly received by the applications.	Data consistency; Communication; Security, privacy and trust	4.2; 4.3; 4.4

Table 1: Application requirements and vertical system functions

5.1 VF: Context and Location Management

Distributed Cooperating Objects systems are designed to measure properties of the physical world. They are, therefore, suitable for gathering the context of an entity, which is the information that can be used to characterise its situation. Individuals, locations, or any relevant objects can be such entities [13]. Since a reasonable amount of data is collected in large systems, context management systems are needed to handle them. Such systems can separate applications from the process of sensor processing and context fusion. To achieve precise actuation and detailed analysis of collected measurement data, however, the location of sensors and actuators need to be known.

Changes in context may trigger actions to influence the monitored entity. Specialised actuators, for instance, may be programmed to control pipe valves when a fluid pressure reaches a certain threshold.

Trends. Classical context management systems use infrastructure-based directories to store the information, for example Aura [39] and Nexus [49]. In the last years there has been a clear trend towards ad-hoc (infrastructure-less) systems. Such systems reflect better the limited spatial relevance of context since this is stored at or near the location where it is generated. Current systems focus on the gathering of sensor data using quality of service specifications (MiLan [44]), the management of context as cross-layer data in the network stack (MobileMan [27]) or more general for the whole node (TinyCubus [75]), the storage of information in the network of cooperating objects at calculated geographic locations (Geographic Hash Table [97]), the querying of information using a SQL-like language based on the 'network as a database' abstraction (TinyDB [71]), and the self-monitoring of cooperating objects ([135]) since they belong to the context themselves. Most of the systems consider hybrid cooperating objects as well. MiLan presents also a further direction of research, the simplification of the use of cooperating objects. The user specifies what it wants and not how.

Applications that use context to adapt their behaviour are called context-aware. Such applications include visitor information systems (e.g. GUIDE), support in workspaces e.g. context dependent configurations of mobile phones (TEA), or smart environments (e.g. Gaia) where facilities in the environment interact with user devices. Also, applications based on ad-hoc networks are on the rise. Due to this fact, adaptation of applications to the context is crucial. This can either be done application-driven, i.e. the application decides which actions should be taken, or system-driven, i.e. the system manages the adaptation transparently. The latter class has drawn attention in the last few years. A few adaptive middlewares or frameworks have been proposed including Impala [68] which is based on finite state machine or TinyCubus [75] which tries to select the best set of algorithms based on several parameters, policies, and different adaptation strategies.

Location services for mobile ad hoc networks only offer limited context information- in this case the position of mobile objects. A scalable, distributed location service is, e.g., GLS [67]. Prior to storing the location in a such a service, the location has to be determined. Small cooperating objects usually do not have a GPS device, so different methods have been developed in the last few years. Two basic approaches are commonly used: having distances to three objects of which the location is known, the own location can be calculated. The other possibility is to measure the angle to two known objects. Since most cooperating objects are equipped with omnidirectional antennas, a very accurate measurement of the angles is not feasible. Though, [74] shows that is it feasible and more

accurate to use two directional antennas on small Mica2 nodes. Several methods exist to determine the distances to other nodes: [36] uses received signal strength indication (RSSI) values delivered by the RF chip in combination with an error model, [33] presents an algorithm for semi-static sensor networks that works with hop counts, and [109] uses more accurate ultrasound and maps the location estimation to a global non-linear optimisation problem which is approximated fully distributed.

Some open issues. Current context management solutions for ad-hoc networks focus only on one part of context management. None cover the complete context management area with gathering of data, in-node data management, in-network data distribution, and in-network data querying.

Positioning is still too inaccurate, better algorithms are required here. Also, for mobile cooperating objects no good approaches exist.

5.2 VF: Data consistency and adaptivity in WSN

The benefits of having several nodes in a wireless sensor network (WSN) mostly come from the fact that many nodes simultaneously monitor the same physical area. Nodes can be put into sleep mode without any loss of precision in the network. This results in energy conservation increasing, therefore, the network lifetime.

The reliability of the system is also improved with several sensor nodes. This scenario, however, raises issues with data inconsistency which may occur due to various reasons - for instance inherent imprecision associated with sensors, inconsistent readings and unreliable data transfer, just to name a few.

This VF provides the functionality to ensure consistency of the sensor data at various system abstraction levels:

- Data consistency may mean that data retrieved from a location in the sensor network should be consistent with data sent to the same location.
- Data consistency may also mean that all sensors sensing the same physical phenomenon should more or less agree on the measured value.
- In a rule-based system, data consistency may mean that all actuators agree on the action that needs to be taken.

The following parameters are needed to ensure accurate and consistent *operation* of WSNs and distributed COs:

- **Localisation:** to interpret sensor data and collaborate with other nodes;
- **Synchronisation:** to ensure that all nodes have an equal understanding of time and the moment at which events take place;
- **Reliable data transfer:** to guarantee delivery of information and/or customisable degrees of reliability for data transmission;

- **Routing:** to accomplish forwarding data from source(s) to destination(s) through collaboration among neighbours in an energy efficient manner while maintaining a best effort level of reliability;
- **Coverage:** to cover a large area while being able to turn off some of the sensors, thus, saving valuable energy, as well as enhancing the accuracy of the sensed data.

Data consistency at the *data processing level* can be achieved through the following mechanisms:

- **State monitoring:** to detect any change in state of the environment, in which sensors are placed, directly from the sensed data;
- **Data fusion:** to take the multiple measurements, some of which may be faulty, and determine either the correct measurement value or a range in which the correct measurement lies;
- **Event detection:** for the WSN to be in charge of monitoring the environment in which its nodes are placed, to detect occurrence of certain events;
- **Fault tolerance and Consensus:** to preserve precision and to agree on measured data and required actions in the presence of faulty collaborative sensors;

Trends. For state monitoring, describing the creations between states and the events that trigger state-change through a set of rules and predicates over events and their parameters has proved to be popular. The authors in [101] proposed this approach for the first time, which was later also utilized and modified by Strohbach et al. [119]. Research in this direction is on rise. Also hybrid distributed algorithms similar to the one proposed by Sahni et al. [107], which is executed by every sensor using the measurement ranges received from the remaining sensors monitoring the same region as data, accompanied with the sensor's own measurement have received considerable attention.

Environmental monitoring, detection, identification, localization or tracking of objects in sensor fields have created a new domain for event detection. Research reported in [100, 123, 42, 31, 128] are a few example to mention.

Introducing, defining and implementing the concept of collaboration in various levels of abstraction is the newest trend in the area of data consistency, which seems to have large potential to solve various issues of the operation of WSN, data processing, and application programming.

Some open issues There are still open issues in the area of data consistency for WSN and cooperating objects. Here we elaborate on some of the most important ones.

A reasonable number of location techniques have been proposed in the literature. An integration framework (e.g. standardised APIs) of such mechanisms would be useful for the applications. This stems from the fact that the application requirements are dynamic and often change. The ability to adapt to these frequent changes are thus essential. Since various localization techniques have been defined, the integration framework may choose a proper localization technique to be utilized based on the requested accuracy at the time and the environmental status.

Reliable and energy-efficient data transfer which is application dependent for improved performance is required. With no doubt collaboration between sensor nodes will increase the reliability

of the transferred data in the WSN. However, the overhead introduced by cooperation along with the additional energy consumption should be in a reasonable level. Indeed, in order to avoid being suboptimal, protocols should be developed according to particular application requirements.

Data fusion issues that require further investigation are the associated overhead of the fusion process and the heterogeneity of data due to different sensor sources.

Also, high-level descriptive language oriented towards the collective model of solving tasks is necessary in order to implement distributed collaborative WSN applications. This language should support data-centric architectures and provide real-time guarantees for energy efficient operations.

5.3 VF: Communication functionality

The communication vertical function refers to the capability of any pair (or group) of devices to exchange information. Different types of communications can be performed: one-to-all, one-to-many, many-to-one, many-to-many. If we consider the case of wireless sensor networks with a single sink, one to all or one to many communications are needed for query dissemination, while many to one communication is explored to gather sensed data at the sink.

Trends. Communication in COs environment is expected to be data centric and attribute-based. This means that more than addressing a specific cooperating object, the communication infrastructure should be able to deliver data to and from groups of COs which share a set of attributes specifying the destination/source address of the information. For example, a user could issue a query on the average temperature of an area in an office building. This query should then be delivered to the objects with temperature sensors. Similarly, once measures on the temperature have been taken, the cooperating objects in the specific area will send packets to the sink(s) reporting the measured values.

The trend of data-centric and attribute-based data dissemination may lead to the selection of different communications overlays depending on the values to be reported. A given overlay interconnects cooperating objects of the same 'group' sharing a common set of attributes (attribute-based routing). Attribute-based routing could leverage the burden of delivering queries to objects which cannot answer the query. Also attribute-aware communication infrastructures may optimise sensor data fusion by selecting routes which maximise the chances of aggregating a given type of data, overall decreasing the network load and energy-consumption [113].

Some open issues. Important VF parameters which should be included in a query are the time constraints and accuracy with which a given query needs to be answered. These are application-dependent and even query-dependent parameters: (a) for a query to be successfully resolved, the sensors must deliver fresh data, (b) the query must be answered fast enough , and (c) the precision with which the query is answered must meet the query requirements. This can be regarded as new concept of *quality of service* requirements that remains to be explored.

The second issue to address is the fact that a one-fits-all protocol stack may not be suitable to this scenario. It is possible to identify the 'vertical functions' that should be provided and possible implementations of such functions for specific sets of possible applications. The communications

protocols will benefit from and sometimes require the information provided by different vertical functions such as time synchronisation and location awareness. Not only time and location information are included in the delivered data, but some protocols such as geographic-based routing can exploit location-awareness to reduce routing overhead and the nodes storage demand. The interactions between complementary vertical functions from the application viewpoint should be further investigated.

Mobility of objects is the third issue to be tackled. Although in many cooperating objects scenarios the devices themselves are unlikely to be mobile they can be, however, located on mobile users or mobile stations so that their location changes in time in a predictable or unpredictable way. On one hand this may have a beneficial effect (e.g. load balancing the energy consumption among the different nodes) but on the other hand it requires mobility management or mobility-aware protocols to be added to the protocol stack. The mobility of some of the devices have been explored by some architectures such as the data mules [112], in which a group of mobile nodes move in the deployment area collecting data from the sensor nodes and delivering the collected data to the sinks.

5.4 VF: Security, Privacy and Trust

COs and WSNs are usually placed in locations that are accessible to everyone – also to attackers. For example, sensor networks are expected to consist of a couple of hundred of nodes that may cover a large area. It is impossible to protect each of them from physical or logical attacks. Thus, every single node is a possible point of attack.

In a CO, security is pervasive. [87] states that security must be integrated into every component to achieve a secure system. Components designed without security can become a point of attack as [54] shows. However, specific vertical functions to enforce security are available for applications.

Since fully tamper resistant devices are hard to build [9] and would cost too much money, protocols for sensor networks have to be designed in a way that they tolerate malfunctioning/attacking nodes while the whole sensor network remains functional.

Trends. Karlof and Wagner [54] describe several attacks against known routing protocols for sensor networks and present countermeasures for some attacks. This shows that secure routing protocols are a trend themselves since security was only a minor issue before. New routing protocols were developed that are resilient to black-hole attacks ([30]) or that use efficient symmetric key primitives to prevent compromised nodes from tampering with uncompromised routes consisting of uncompromised nodes (Ariadne [50]). It also deals with a large number of DoS attacks, as well as [130] which maps jammed regions and reroutes around it. [93] proposed SIA, a framework for secure information aggregation in large sensor networks which uses random sampling mechanisms and interactive proofs to verify that the answer of an aggregator is a good approximation of the true value.

Encryption is the basic technique for securing and authenticating transmitted data. Using asymmetric cryptography on highly resource constrained devices is often not possible due to delay, energy and memory constraints [20, 18]. With symmetric cryptographical methods, two communicating cooperating objects need a common key. Secure pebblenets [11] use a shared key for the whole network but since the compromise of one single node leads to a security failure of the whole network pairwise approaches have been developed in the last years. SPINS [88] uses a central base station

to establish new pairwise session keys, [35, 23] is based on random pairwise key pre-distribution. [125] does not rely on infrastructure or pre-distribution: physical contact between nodes is used for establishing the initial key, new keys are then established by sending key-shares along node-disjoint paths via secured point-to-point communication to each other.

Traffic analysis on the ciphertext can reveal sensitive information about the data even if it is encrypted. To provide better secrecy, dummy messages can be generated. This seems to be diametrical in resource constraint cooperating objects, but secrecy may be a more important goal than energy saving in some cases. [126] presents an energy-efficient framework that maintains the anonymity of a virtual infrastructure including routing by randomising communications.

Using cooperating objects, especially sensor networks, humans can be observed which leads to privacy problems. Encryption tackles overhearing and data coarsening ensures that no conclusions can be drawn from the data to a single person. The usage of sensor networks to spy on individuals cannot be met with technology alone, but only with a mix of societal norms, new laws, and technological responses [87].

With several cooperating objects contributing to a common goal, it is necessary to assess the reliability of the information provided by an individual cooperating object. With respect to services and transactions, trust has been researched for several years. For data-centric and fully distributed architectures research has just started. A distributed voting system is proposed in [23] where votes can be cast against misbehaving nodes until all other nodes refuse to communicate with this node. [38] is a more general reputation-based framework with each node monitoring the behaviour of other nodes and building up their reputation thereupon. Nodes with a bad reputation can be excluded from the community.

Some open issues. Existing protocols have to be made resilient to attacks, and more new protocols have to be designed as well. Intelligent intrusion detection systems for cooperating objects are also needed.

Encryption concepts for aggregation are needed since along an aggregation tree, only point-to-point encryption is feasible, but an attacker in the tree has full access to the data. No general solutions to this problem exist so far.

Some of the key distribution protocols consume a lot of power, therefore energy-efficient algorithms without assumptions about the available infrastructures are required.

Energy-efficient solutions are needed for secrecy. Systems are needed where the user can specify the amount of energy it should spend for dummy messages and the maximum delay it wants to accept to enable mix cascades in cooperating objects.

Another important question is how the illegitimate use of sensor networks, e.g. to spy on individuals, can be prevented.

Concerning trust, energy is the main problem, since the exchange of votes and reputation data consumes reasonable power.

5.5 VF: Distributed Storage and data Search

In cooperating objects and wireless sensor networks (WSNs) efficient storage and querying of data are both critical and challenging issues. Especially in WSNs large amounts of sensed data are

collected by a high number of tiny nodes. Scalability, power and fault tolerance constraints make distributed storage, search and aggregation of these sensed data essential.

It is possible to perceive a WSN as a distributed database and run queries which can be given in SQL format. These queries can also imply some rules about how to aggregate the sensed data while being conveyed from sensor nodes to the query owner. Data aggregation [17], [135] techniques that reduce the number of data packets conveyed through the network are therefore important and also required for effective fusion of data collected by a vast number of sensor nodes [6, 43]. A query in a sensor network may be perceived as the task or interest dissemination process. Sensor nodes process the task or interest and return data to the interest owner.

The following characteristics of WSNs should be considered while designing a data storage, querying and aggregation scheme for WSNs:

- Sensor nodes are limited in both memory and computational resources. They cannot buffer a large number of data packets.
- Sensor nodes generally disseminate short data packets to report an ambient condition, e.g., temperature, pressure, humidity, proximity report, etc.
- The observation areas of sensor nodes often overlap. Therefore, many sensor nodes may report correlated data of the same event. However, in many cases the replicated data are needed because the sensor network concept is based on the cooperative effort of low fidelity sensor nodes [6]. For example, nodes may report only proximity, then the size and the speed of the detected object can be derived from the locations of the nodes reporting them, and timings of the reports. The collaboration among the nodes should not be hampered by the data aggregation scheme.
- Since there may be thousands of nodes in a sensor field, associating data packets from numerous sensors to the corresponding events, and correlating the data of the same reported event at different times may be a very complicated task for a single sink node or a central system.
- Due to a large number of nodes and other constraints such as power limitations, sensor nodes are generally not globally addressed [6]. Therefore, only the use of address-centric protocols are mostly inefficient. Instead, data-centric or location-aware addressing protocols where intermediate nodes can route data according to its content [62] or the location of the nodes [21], should be used.
- Querying the whole network node by node is impractical. So attribute-based naming and data-centric routing [114] are essential for WSNs.

Trends. Queries made to search data available in a WSN should be resolved in the most power efficient way. This can be achieved by reducing either the number of nodes involved in resolving a query or the number of messages generated to convey the results.

There is a considerable research interest to develop efficient data querying schemes for WSNs. Data querying systems in general have two major components; interest/data dissemination and query processing and resolution. Query resolution usually involves data aggregation for energy

efficient processing. We first examine interest and data dissemination techniques which are closely related to data querying, and then query processing and resolution techniques.

Interest and Data Dissemination. Protocols for data dissemination protocols are designed to efficiently transmit and receive queries and sensed data in WSNs. In this subsection, we briefly explain five of the best known protocols.

- *Classic Flooding:* in classic flooding, a node that has data to disseminate broadcasts the data to all of its neighbours.
- *Gossiping [6]:* this technique uses randomisation to conserve energy as an alternative to the classic flooding approach. Instead of forwarding data to all its neighbours, a gossiping node only forwards data to one randomly selected neighbour.
- *SPIN [43]:* this protocol is based on the advertisement of data available in sensor nodes. When a node has data to send, it broadcasts an advertisement (ADV) packet. The nodes interested in this data reply back with a request (REQ) packet. Then the node disseminates the data to the interested nodes by using data (DATA) packets. When a node receives data, it also broadcasts an ADV, and relay DATA packets to the nodes that send REQ packets. Hence the data is delivered to every node that may have an interest.
- *Directed Diffusion [53]:* in SPIN the routing process is stimulated by sensor nodes. Another approach, namely directed diffusion, is sink oriented. A sink is the name given to the central node responsible for gathering data from all the other nodes in directed diffusion where the sink floods a task to stimulate data dissemination throughout the sensor network. While the task is being flooded, sensor nodes record the nodes which send the task to them as their gradient, and hence the alternative paths from sensor nodes to the sink are established. When there is data to send to the sink, this is forwarded to the gradients. One of the paths established is reinforced by the sink. After that point, the packets are not forwarded to all of the gradients but to the gradient in the reinforced path.
- *LEACH [45] :* is a clustering based protocol that employs randomised rotation of local cluster heads to evenly distribute the load among the sensors in the network. In LEACH, the nodes organise themselves into local clusters, with one node acting as a local cluster head. LEACH includes randomised rotations of the high-energy cluster-head position such that it rotates among the various sensors in order not to drain the battery of a single sensor. In addition LEACH performs local data fusion to compress the amount of data being sent from the clusters to the base station.

Query Processing and Resolution When a query arrives at a sensor node, it is first processed by the node. If the node can resolve the query, the result of the query is disseminated. This approach is one of the simplest ways of resolving and processing a query. Sensor nodes usually take advantage of collaborative processing to resolve queries so that smaller number of messages are transmitted in the network. Queries can be flooding-based where a query is flooded to every node in the network.

Alternatively they can be expanded ring search (ERS) based where a node does not relay a query that it can resolve. Currently available query processing systems are summarised below:

- *TinyDB [69]:* is a query processing system for extracting information from a network of TinyOS sensors. TinyDB provides a simple SQL-like interface to specify the data along with additional parameters such as the rate at which data should be refreshed much as in traditional databases. Given a query specifying data interests, TinyDB collects data from nodes in the environment, filters and aggregates them. TinyDB does this via power-efficient in-network processing algorithms. Some key features of TinyDB areas follows: TinyDB provides metadata management, provides a declarative query language, supports multiple query resolution on the same set of nodes and supports different levels of in-network aggregation. It also includes a facility for simple triggers, or queries that execute some command when a result is produced.
- *COUGAR [132]:* is a query layer for sensor networks which accepts queries in a declarative language that are then optimised to generate efficient query execution plans with in-network processing which can significantly reduce resource requirements.
- *Active Query Forwarding scheme (ACQUIRE) [104]:* aims at reducing the number of nodes involved in queries. In this scheme each node that forwards a query tries to resolve it. If the node resolves the query, it does not forward it further but sends the result back. Nodes collaborate with their n hop neighbours, where n is referred to as the look ahead parameter. If a node cannot resolve a query after collaborating with n hop neighbours, it forwards it to another neighbour. When n equals to 1, ACQUIRE carries out flooding in the worst case.

Some open issues. The routing protocols for WSNs are generally designed for networks that have fixed homogeneous sensor nodes and are based on the assumption that all nodes try to convey data to a central node, often named sink. However, in cooperating objects networks there will be heterogeneous nodes that can be mobile, and the sensed data will be needed by many nodes, i.e., multiple sinks. Distributed data storage and search (DSS) solutions should have support for heterogeneity. Additionally, open research issues on DSS can be summarised as follows: replication, concurrency and consistency control of distributed data, mobility support, general purpose solutions for declarative query languages, interoperability with open standards, query driven architectures with QoS support.

5.6 VF: Resource management

COs may form a self-organising network, in which COs arbitrarily join and leave or even move during this operation. Since system components are distributed, any action often involves multiple COs at a time. Due to the distributed, dynamic, and uncertain (since both COs and communication are associated with uncertainty) nature of system components, the design of such embedded wireless collaborative system proves to be difficult and requires a scheme to facilitate the system design and application development.

Resource management aims at providing a way to manage the resources of a system by enabling high-level system primitives to hide unnecessary low-level details. It enables the application to be

independent of the actual underlying distributed system. It should also address the dynamic nature of available resources, such as variable network bandwidth. Because the system is built of error-prone components, failures should be handled as normal and not as exceptions.

Trends. Over the years various enabling technologies have contributed to the evolution of adaptation and reconfigurable software, each having its own advantages and disadvantages. The most important ones are, middleware, component-based design, computational reflection, and separation of concerns. Recent efforts have been directed towards replacing the traditional strict modularisation or layering design with more energy-efficient one, i.e., cross-layered design. However, most of the important cross-layer adaptation frameworks and projects that have been proposed assume mobile devices that are richer than wireless sensor nodes in terms of resources. Therefore, adaptation frameworks designed specifically for WSN are emerging.

Operating systems for wireless sensor networks with data centric architecture and especially designed for limited memory are considered to form one of the main trends in the area of adaptation in WSN. Due to increasing interest in having adaptive behaviour on a WSN-wide scale, recently attentions have been paid to extending the concept of data centric architecture to the whole network.

Some open issues. One of the most important open issues regarding resource management is designing an architecture tailored to support different adaptation models. Since application requirements, environmental conditions as well as available resources and services may frequently change, depending on goals of the application at the time, sensor nodes must be able to adapt accordingly. However, since these goals may change as well, different adaptation models are required from which the most proper one can be automatically selected. As an example, the application may at a time aim at frequent and high accuracy sensor readings and at the other time at less accurate data and more energy efficient readings.

A direct result of the above-mentioned issue is the need for multi-dimensional optimisation techniques for resource management.

5.7 VF: Time synchronisation

Applications need to establish a common sense of time among the cooperating objects participating in their sensing and actuation goals. Such a functionality can be offered through a time synchronisation vertical function.

Forest fire monitoring is a scenario that requires not only the information of whether there has been an indication of fire but also where and when this is happening. The collected sensor data provides the basis for the decision making process which may trigger actions to be taken in the monitored environment in order to address abnormal circumstances. Thus, the true time of the observed events is crucial to the prompt action of fire fighting.

It is sufficient to associate local clock timestamps with the sensor data when decisions can be made locally. For instance, the decision of regulating the fluid flux of an industrial pipe may be taken locally and actions would follow through local CO actuators, for example, a regulating valve.

However, there are cases where the collected sensor data will be aggregated from various cooperating objects at various points inside the network. To perform such a function, the data need to be

timestamped at the source. In this case, the data originates from multiple sources which may or may not agree on a similar time. Thus, a common view on time among the COs is an essential aspect to correlate multiple sources sensor data in order to produce a consistent and reliable result.

Trends. An important question to ask is whether the Network Time Protocol (NTP) [80] currently in use on the Internet could be adopted without further modification in low cost wireless sensor networks. This protocol relies on an external time reference to synchronise the top layer of time servers called *stratum 1* servers.

As the NTP protocol was originally developed considering other design issues, some of the challenges that arise in CO and WSN scenarios cannot be addressed if the protocol is adopted unmodified in sensor network applications. One of the issues is energy efficiency as the external source of time (e.g GPS) tends to consume more power than other components of a sensor node and does not work in indoor areas which have no line-of-sight to the satellites.

However the issue raised in [102] that the NTP protocol maintains a synchronised system clock by regularly adding small increments to a system counter is an important one. This behaviour precludes the sensor node processor of being switched to a power-saving idle mode. In addition, NTP servers must be prepared to handle synchronisation requests at any point in time. This could potentially introduce a problem as the communication module of a CO (e.g. radio) shows the highest cost among the other components when it is transmitting or listening to any signal from the network. Thus, this module could not be put in sleep mode in order to save energy.

GPS devices of small size and low power consumption have been developed and are now commercially available. The Leadtek 9546 low-power GPS receiver [65] was recently integrated into a sensor board of the Crossbow Mote [52]. These devices consume a current of 60 mA at 3.3V in full operation. Despite the fact that such a consumption profile is higher compared to the micro-controller current draw of 8mA, it is comparable to the energy consumption profile of the radio module.

However, such sensor nodes are optimised to operate in sleep mode most of time. If the GPS is configured to operate occasionally to acquire the time signal, its energy consumption may not be an issue. Although adding this particular GPS module to a sensor board increases its cost by 50% today, we believe that the cost will drop significantly because of economy of scale. In contrast, this is also the case for the current cost of sensor nodes which are similar to the cost of a sensor board equipped with GPS.

The current trend is towards the design of self-configuring protocols on the assumption that NTP cannot be directly used in cooperating objects and WSNs applications. The latest research results address the time synchronisation problem with approaches that do not rely on GPS time signals. As researchers realise that low-power and low-cost GPS devices are becoming commercially available, the research trend may shift to hybrid protocol designs where GPS is used as the primary source of time. Such information is then disseminate throughout the network to non-GPS nodes. How such time servers ought to be organised in the network (e.g. hierarchical such as NTP) would be an important research question.

Some open issues The characteristics of the network topology in ad-hoc wireless sensor networks may introduce severe delays because of disconnected parts of the network. To compensate for

such uncertainties, the time synchronisation protocol needs to be *tolerant* to the message delay irrespective of its dominant source (e.g. processing, transmission).

The second issue is related to trust. When some sensor nodes become responsible to provide the time reference to other nodes in the network, the issue of trusting the source of time arises. Also, if GPS time signal is used in some reference nodes a related question is how we can guarantee the authenticity of the GPS time.

Finally, Romer et al [102] suggest that calibration of sensors is a much more general and complex problem than time synchronisation. The interesting question is whether the current proposed time synchronisation techniques can be used for general sensor calibration problems. Also, it may be the case that time synchronisation (at least for outdoor sensor deployment) can be achieved by instrumenting low-cost, low-power GPS devices to sensor nodes as previously discussed.

References

[1] Software fundamentals: collected papers by david l. parnas. *SIGSOFT Softw. Eng. Notes*, 26(4), 2001.

[2] Marcucci A., Nati M., Petrioli C., and Vitaletti A. The Impact of Data Aggregation and Hierarchical Organizazion on Wireless Sensor Networks. Technical report, Rome University, 2004.

[3] H. Karl A. Köpke and A. Wolisz. Consensus using aggregation - a wireless sensor network specific solution. Technical Report TKN-04-004, Technical University of Berlin, April 2004.

[4] Sarita V. Adve, Albert F. Harris, Christopher J. Hughes, Douglas L. Jones, Robin H. Kravets, Klara Nahrstedt, Daniel Grobe Sachs, Ruchira Sasanka, Jayanth Srinivasan, , and Wanghong Yuan. The illinois grace project: Global resource adaptation through cooperation. In *Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN)*, june 2002.

[5] Mehmet Aksit and Zièd Choukair. Dynamic, adaptive and reconfigurable systems overview and prospective vision. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)*, May 2003.

[6] I.F. Akyildiz, W. Su, Y.Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, pages 102–114, August 2002.

[7] J.N. Al-Karaki and A.E. Kamal. On the correlated data gathering problem in wireless sensor networks. In *Proceedings. of the Ninth International Symposium on Computers and Communications (ISCC 2004)*, volume 1, pages 226 – 231, June 2004.

[8] A.Nasipuri and K.Li. A directionality based location discovery scheme for wireless sensor networks. In *1st ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[9] Ross Anderson and Markus Kuhn. Tamper resistance - a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1–11, November 1996.

[10] ANSI. SQL Standard. Technical Report X3, 135-1992.

[11] Stefano Basagni, Kris Herrin, Danilo Bruschi, and Emilia Rosti. Secure pebblenets. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 156–163, New York, NY, USA, 2001. ACM Press.

[12] S. Baydere and M.A. Ergin. An architecture for service access in mobile ad hoc networks. In *Proceedings of IASTED WOC*, Banff, Canada, July 2002.

[13] Christian Becker. *System Support for Context-Aware Computing*. Fakultt Informatik, Elektrotechnik und Informationstechnik, Universitt Stuttgart, 2004.

[14] Lodewijk Bergmans and Mehmet Aksit. Composing crosscutting concerns using composition filters. *Commun. ACM*, 44(10):51–57, 2001.

[15] Antoine Beugnard, Jean-Marc Jzquel, Nol Plouzeau, and Damien Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.

[16] Gordon S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, London, 1998. Springer-Verlag.

[17] A. Boulis, S. Ganerival, and M.B. Srivastava. Aggregation in Sensor Networks: An Energy Accuracy Tradeoff. In *Proceedings of IEEE SNPA'03*, pages 128–138, May 2003.

[18] Michael Brown and Donny Cheung. PGP in constrained wireless devices. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[19] H. Cam, S. Ozdemir, P. Nair, and D. Muthuavinashiappan. ESPDA: Energy-efficient and Secure Pattern-based Data Aggregation for wireless sensor networks. In *Proceedings of IEEE Sensors 2003*, volume 2, pages 732–736, October 2003.

[20] D. Carman, P. Kruus, and B. Matt. Constraints and approaches for distributed sensor network security. Technical Report #00-010, NAI Labs, September 2000.

[21] E. Cayirci. Addressing in wireless sensor networks. In *COST-NSF Workshop on Exchanges and Trends in Networking (Nextworking03)*, Crete, 2003.

[22] E. Cayirci and T. Coplu. Data Aggregation and Dilution by Using Modulus Addressing in Wireless Sensor Networks. In *Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference*, pages 373–379, December 2004.

[23] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 197, Washington, DC, USA, 2003. IEEE Computer Society.

[24] E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A Programming Model for Event-Driven Embedded Systems. In *18th Annual ACM Symposium on Applied Computing (SAC'03)*, Melbourne (FL), USA, March 2003.

[25] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 2000. ACM Press.

[26] T. Clouqueur, P. Ramanathan, K. K. Saluja, and K.-C. Wang. Value-fusion versus decision-fusion for fault-tolerance in collaborative target detection in sensor networks. In *Proceedings of the 4^{th} Annual Conference on Information Fusion*, 2001.

[27] Marco Conti, Gaia Maselli, Giovanni Turi, and Silvia Giod ano. Cross-layering in mobile ad hoc network design. *IEEE Computer*, 37(2):48–51, 2004.

[28] Marco Conti, Gaia Maselli, Giovanni Turi, and Silvia Giordano. Cross-layering in mobile ad hoc network design. *IEEE Computer*, pages 48–51, feb 2004.

[29] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems, Concepts and Design*. Addison-Wesley, 3 edition, 2001.

[30] Jing Deng, Richard Han, and Shivakant Mishra. A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks. In *Proceedings of the 2nd IEEE International Workshop on Information Processing in Sensor Networks (IPSN 2003)*, pages 349–364, April 2003.

[31] D. M. Doolin, S. D. Glaser, and N. Sitar. Software architecture for GPS-enabled wildfire sensorboard. In *TinyOS Technology Exchange*, February 2004.

[32] W. Du, W. Du, J. Deng, Y.S. Han, and P.K.A. Varshney. A witness-based approach for data fusion assurance in wireless sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, volume 3, pages 1435 – 1439, December 2003.

[33] Stefan Dulman and Paul Havinga. Statistically enhanced localization schemes for randomly deployed wireless sensor networks. In *Proceedings of the DEST International Workshop on Signal Processing for Sensor Networks*, Melbourne, Australia, 2004.

[34] Stefan Dulman, Tjerk Hofmeijer, and Paul Havinga. Wireless sensor networks dynamic runtime configuration. In *Proceedings of ProRISC 2004*, 2004.

[35] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47, New York, NY, USA, 2002. ACM Press.

[36] Leon Evers, Stefan Dulman, and Paul Havinga. A Distributed Precision Based Localization Algorithm for Ad-Hoc Networks. In *Proceedings of the Second International Conference on Pervasive Computing 2004*, pages 269–286, Linz/Vienna, Austria, apr 2004.

[37] J. Feng, F. Koushanfar, and M. Potkonjak. *Sensor Network Architecture*, chapter 12. CRC Press, July 2004.

[38] Saurabh Ganeriwal and Mani B. Srivastava. Reputation-based framework for high integrity sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 66–77, New York, NY, USA, 2004. ACM Press.

[39] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments"*, 1(2):22–31, 2002.

[40] Andrea J. Goldsmith and Stephen B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communications*, 9:8–27, August 2002.

[41] Tian He, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. AIDA: Adaptive application-independent data aggregation in wireless sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(2), May 2004.

[42] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 270–283. ACM Press, 2004.

[43] W.R. Heinzelan, J. Kulik, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *ACM MobiCom99*, pages 174–185, Seattle, Washington (USA), August 1999.

[44] Wendi B. Heinzelman, Amy L. Murphy, Hervaldo S. Carvalho, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, January/February 2004.

[45] W.R. Heinzelman, A.Chandrakasan, and H.Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *IEEE JSAC*, 17(8):1333–44, August 1999.

[46] A. Helmy. Mobility-assisted resolution of queries in large-scale mobile sensor networks. *Computer Networks (Elsevier)*, 43(4):437–458, November 2003.

[47] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *Computer*, 34(8):57–66, August 2001. This article is also excerpted in "IT Roadmap to a Geospatial Future," a 2003 report from the Computer Science and Telecommunications Board of the National Research Council.

[48] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, Cambridge (MA), USA, November 2000.

[49] Fritz Hohl, Uwe Kubach, Alexander Leonhardi, Kart Rothermel, and Markus Schwehm. Next century challenges: Nexus - an open global infrastructure for spatial-aware applications. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 249–255, New York, NY, USA, 1999. ACM Press.

[50] Yih-Chun Hu, A. Perrig, and D. B. Johnson. Ariadne: A Secure on-demand Routing Protocol for Ad Hoc Networks. In *Proceedings of the 8th annual international conference on Mobile computing and networking (Mobicom)*, Atlanta, Georgia, USA, sep 2002.

[51] Eoin Hyden. *Operating System Support for Quality of Service*. PhD thesis, Wolfson College, University of Cambridge, February 1994.

[52] Crossbow Tech Inc. *MTS 420 Datasheet*. http://www.xbow.com.

[53] C. Intanagonwiwat, R.Govindan, and D.Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *ACM MobiCom00*, pages 56–67, Boston, Massachusetts (USA), August 2000.

[54] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and coutermeasures. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, AK, may 2003.

[55] Eric P. Kasten, Philip K. McKinley, S. M. Sadjadi, and Kurt Stirewalt. Separating introspection and intercession to support metamorphic distributed systems. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 465–472, Washington, DC, USA, 2002. IEEE Computer Society.

[56] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer Magazine*, 36(1):41–50, January 2003.

[57] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The art of metaobject protocol*. MIT Press, Cambridge, MA, USA, 1991.

[58] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355, 2001.

[59] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[60] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[61] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli. Fault Tolerance in Wireless Ad-Hoc Sensor Networks. *IEEE Sensors*, 2:1491–1496, June 2002.

[62] B. Krishnamachari, D.Estrin, and S.Wicker. Modelling Data-Centric Routing in Wireless Sensor Networks. In *IEEE INFOCOM'02*, New York, USA, June 2002.

[63] M. Kumar. A consensus protocol for wireless sensor networks. Master's thesis, Wayne State University, 2003.

[64] O. Kmmerling and M. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99)*, pages 9–20, Chicago, Illinois, USA, May 1999.

[65] Leadtek. *GPS 9546 Module Technical Specification*. www.leadtek.com, August 2003.

[66] Thomas Ledoux. Opencorba: A reflective open broker. In *Reflection '99: Proceedings of the Second International Conference on Meta-Level Architectures and Reflection*, pages 197–214, London, UK, 1999. Springer-Verlag.

[67] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, Boston, Massachusetts, August 2000.

[68] Ting Liu and Margaret Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *Proc. of the 9th ACM SIGPLAN Symp. on Principles and Practice of P arallel Programming*, pages 107–118, 2003.

[69] S. Madden, J. Hellerstein, and W. Hong. TinyDB: In-Network Query Processing in TinyOS. Technical Report TinyOS Document, version 4, September 2003.

[70] S. Madden, R.S. Szewczyk, M.J. Franklin, and D.Culler. Supporting Aggregate Queries over Ad-Hoc Wireless Sensor Networks. In *4th IEEE Workshop on Mobile Computing Systems and Applications*, New York (USA), June 2002.

[71] Samuel Madden, Michael J. Franklin, and Joseph M. Hellerstein a nd Wei Hong. The design of an acquisitional query processor for sensor network s. In *Proceedings of the 2003 ACM SIGMOD international conference on Ma nagement of data*, pages 491–502. ACM Press, 2003.

[72] S.R. Madden, M.J.Franklin, J.M.Hellerstein, and W.Hong. TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *OSDI Conference*, Boston, MA, USA, December 2002.

[73] Pattie Maes. Concepts and experiments in computational reflection. In *OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications*, pages 147–155. ACM Press, 1987.

[74] N. Malhotra, M. Krasniewski, C. Yang, S. Bagchi, and W. Chappell. Location Estimation in Ad-Hoc Networks with Directional Antenna. In *Proceeding of the 25th International Conference on Distributed Computing Systems (ICDCS)*, Columbus, Ohio, jun 2005.

[75] Pedro José Marrón, Andreas Lachenmann, Daniel Minder, Jörg Hähner, Robert Sauter, and Kurt Rothermel. Tinycubus: A flexible and adaptive framework for sensor networ ks. In Erdal Çayırcı Şebnem Baydere, and Paul Havinga, editors, *Proc. of the 2nd European Workshop on Wireless Sensor Networks*, pages 278–289, Istanbul, Turkey, January 2005.

[76] Philip K. McKinley, S. Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *IEEE Computer*, pages 56–64, July 2004.

[77] Philip K. McKinley, S. Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. A taxonomy of compositional adaptation. Technical Report MSU-CSE-04-17, Department of Computer Science, Michigan State University, East Lansing, Michigan, May 2004.

[78] Microsoft .net.

[79] Com: Delivering on the promises of component technology, 2000.

[80] D. L. Mills. Internet Time Synchronization: the Network Time Protocol. *IEEE Trans. Communications*, 39(10):14821493, October 1991.

[81] H. Naguib and G. Coulouris. Towards automatically Configurable Multimedia Applications. In *ACM Multimedia Middleware workshop at MM'01*, Ottawa, Canada, October 2001.

[82] Dushyanth Narayanan. *Operating System Support for Mobile Interactive Applications*. PhD thesis, School of Computer Science, Computer Science Department, Carniege Melon University, August 2002.

[83] Rolf Neugebauer and Derek McAuley. Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, Schloss Elmau, Germany, May 2001.

[84] D. Niculescu and B. Nath. Position and Orientation in Ad hoc Networks. *Journal of Ad Hoc Networks*, 2(2):133–151, April 2004.

[85] Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Tilton, Jason Flinn, and Kevin Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles October 1997, St. Malo*, October 1997.

[86] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[87] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, 2004.

[88] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, 2002.

[89] T. Pham, Eun Jik Kim, and M. Moh. On data aggregation quality and energy efficiency of wireless sensor network protocols - extended summary. In *Proceedings of the First International Conference on Broadband Networks (BroadNets 2004)*, pages 730–732, October 2004.

[90] Christian Poellabauer, Hasan Abbasi, and Karsten Schwan. Cooperative run-time management of adaptive applications and distributed resources. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 402–411. ACM Press, 2002.

[91] Christian Poellabauer and Karsten Schwan. Kernel support for the event-based cooperation of distributed resource managers. In *RTAS '02: Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, page 3. IEEE Computer Society, 2002.

[92] Christian Poellabauer, Karsten Schwan, Sandip Agarwala, Ada Gavrilovska, Greg Eisenhauer, Santosh Pande, Calton Pu, and Matthew Wolf. Service morphing: Integrated system- and application-level service adaptation in autonomic systems. In *Proceedings of the 5th Annual International Workshop on Active Middleware Services (AMS 2003)*, June 2003.

[93] B. Przydatek, D. Song, and A. Perrig. SIA: Secure Information Aggregation in Sensor Networks. In *Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems*, pages 255–265, may 2003.

[94] Anu Purhonen and Esa Tuulari. *Ambient intelligence and the development of embedded system software*, pages 51–67. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[95] Python web site. `http://www.python.org/`.

[96] RAPIDware Project web site. `http://www.cse.msu.edu/rapidware/`.

[97] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. Ght: a geographic hash table for data-centric storage. In *Proceedings of the 1st ACM international workshop on Wireless sen sor networks and applications*, pages 78–87. ACM Press, 2002.

[98] Uts Roedig, Andre Barroso, and Cormac Sreenan. Determination of aggregation points in wireless sensor networks. In *Proceedings of the 30th EUROMICRO conference (EUROMICRO' 04)*, 2004.

[99] Manuel Roman and Roy H. Campbell. Gaia: enabling active spaces. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 229–234, New York, NY, USA, 2000. ACM Press.

[100] K. Römer. Tracking real-world phenomena with smart dust, 2004.

[101] Kay Römer and Friedemann Mattern. Event-based systems for detecting real-world states with sensor networks: A critical analysis. In *DEST Workshop on Signal Processing in Sensor Networks at ISSNIP*, pages 389–395, Melbourne, Australia, december 2004.

[102] K. Rmer, P. Blum, and L. Meier. *Time Synchronization and Calibration in Wireless Sensor Networks*, chapter 7. Wiley and Sons, September 2005.

[103] Daniel Grobe Sachs, Wanghong Yuan, Christopher J. Hughes, Albert Harris, Sarita V. Adve, Douglas L. Jones, Robin H. Kravets, and Klara Nahrstedt. Grace: A hierarchical adaptation framework for saving energy. Technical report UIUCDCS-R-2004-2409, Computer Science, University of Illinois, Februrary 2004.

[104] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active Query Forwarding in Sensor Networks (ACQUIRE). *Elsevier Journal of Ad Hoc Networks*, January 2005.

[105] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten. Architecture and operation of an adaptable communication substrate. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, pages 46–56, May 2003.

[106] S. M. Sadjadi, P.K. McKinley, R.E.K. Stirewalt, and B.H.C. Cheng. Trap: Transparent reflective aspect programming. Technical Report MSU-CSE-03-31, Department of Computer Science, Michigan State University, East Lansing, Michigan, November 2003.

[107] S. Sahni and X. Xu. Algorithms for wireless sensor networks. *International Journal on Distributed Sensor Networks*, pages 35–56, 2004.

[108] F. A. Samimi, P. K. McKinley, S. M. Sadjadi, and P. Ge. Kernel-middleware interaction to support adaptation in pervasive computing environments. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 140–145. ACM Press, 2004.

[109] Andreas Savvides, Heemin Park, and Mani B. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 112–121, New York, NY, USA, 2002. ACM Press.

[110] Albrecht Schmidt, Antti Takaluoma, and Jani Mäntyjärvi. Context-Aware Telephony Over WAP. *Personal Ubiquitous Comput.*, 4(4):225–229, 2000.

[111] Douglas C. Schmidt. Middleware for real-time and embedded systems. *Commun. ACM*, 45(6):43–48, 2002.

[112] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Network. In *IEEE Workshop on Sensor Network Protocols and Applications (SNPA)*, May 2003.

[113] A. Sharaf, Jonathan Beaver, Alexandros Labrinidis, and K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *The International Journal on Very Large Data Bases (VLDB)*, 13, December 2004.

[114] C. Shen, C. Srisathapornphat, and C.Jaikaeo. Sensor Information Networking Architecture and Applications. *IEEE Personal Communications Magazine*, pages 52–59, August 2001.

[115] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Aggregation: Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, November 2004.

[116] Brian Cantwell Smith. Reflection and semantics in lisp. In *POPL '84: Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 23–35, New York, NY, USA, 1984. ACM Press.

[117] I. Solis and K. Obraczka. The impact of timing in data aggregation for sensor networks. In *IEEE International Conference on Communications 2004*, volume 6, pages 3640 – 3645, June 2004.

[118] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *7th International Workshop on Security Protocols*, LNCS, pages 172–194, 1999.

[119] Martin Strohbach, Hans-Werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In *Adjunct Proceedings of UbiComp 2003*, September 2003.

[120] Enterprise javabeans technology, 2000.

[121] B. Sundararaman, U. Buy, and A.D. Kshemkalyani. Clock Synchronization in Wireless Sensor Networks: A Survey. *Ad-Hoc Networks*, 3(3):281–323, May 2005.

[122] Clement Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesely, January 1998.

[123] The Ohio State University NEST Team. A line in the sand, August 2003.

[124] E. Truyen, B. Jrgensen, W. Joosen, and P. Verbaeten. Aspects for run-time component integration, 2000.

[125] Arno Wacker, Timo Heiber, Holger Cermann, and Pedro Marron. A fault-tolerant Key-Distribution Scheme for Securing Wireless Ad-Hoc Networks. In *Proceedings of the Second International Conference on Pervasive Computing 2004*, pages 194–212, Linz/Vienna, Austria, apr 2004.

[126] A. Wadaa, S. Olariu, L. Wilson, M. Eltoweissy, and K. Jones. On Providing Anonymity in Wireless Sensor Networks. In *Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)*, pages 411–418, Newport Beach, CA, USA, jul 2004.

[127] Ian Welch and Robert J. Stroud. Kava - a reflective java based on bytecode rewriting. In *Proceedings of the 1st OOPSLA Workshop on Reflection and Software Engineering*, pages 155–167. Springer-Verlag, 2000.

[128] Geoffrey Werner-Allen, Jeff Johnson, Mario Ruiz, Jonathan Lees, and Matt Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *to appear in Proceedings of 2nd European Workshop on Wireless Sensor Networks,Istanbul, Turkey*, February 2005.

[129] J. Wong, S. Megerian, and M. Potkonjak. Design Techniques for Sensor Appliances: Foundations and Light Compass Case Study. In *40th IEEE/ACM Design Automation Conference*, pages 66–71, June 2003.

[130] Anthony D. Wood and John A. Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.

[131] Zhixue Wu. Reflective java and a reflective component-based transaction architecture. In *Proceedings of Workshop on Reflective Programming in C++ and Java*, 1998.

[132] Y. Yao and J.E. Gehrke. Query Processing in Sensor Networks. In *First Biannual Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, California, USA, January 2003.

[133] Wei Yuan, S.V. Krishnamurthy, and S.K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *IEEE Global Telecommunications Conference (GLOBECOM '03)*, volume 1, pages 221–225, December 2003.

[134] Heng Zeng, Carla S. Ellis, and Alvin R. Lebeck. Experiences in managing energy with ecosystem. *IEEE Pervasive Computing*, January-March 2005.

[135] Jerry Zhao, Ramesh Govindan, and Deborah Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of First IEEE International Workshop on Sensor Net work Protocols and Applications*, pages 139–149, May 2003.

