INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

# Providing Homogeneous Access for Sensor Data Management

*Gilberto Zonta Pastorello Jr*
*Claudia Bauzer Medeiros* *André Santanchè*

Technical Report - IC-07-012 - Relatório Técnico

May - 2007 - Maio

# Providing Homogeneous Access for Sensor Data Management *

Gilberto Zonta Pastorello Jr† Claudia Bauzer Medeiros† André Santanchè‡

†Institute of Computing
University of Campinas
Av Albert Einstein, 1251
Campinas – SP – Brasil
{gilberto,cmbm}@ic.unicamp.br

‡UECE
Universidade Salvador
Av Cardeal da Silva, 747
Salvador – BA – Brasil
santanche@unifacs.br

**Abstract**

We are facing the proliferation of several kinds of sensing devices, from satellites to tiny sensors. This has opened up new possibilities for us to understand, manage and monitor a given environment, from the small – e.g., a room – to the large – e.g., the planet. This, however, has added a new dimension to the classic problem of heterogeneous data management – how to handle increasing volumes of sensing data from a wide range of sensors. This report is concerned with the problem of sensor data publication. Our solution involves the design and implementation of a framework for sensor data management, which applies technologies based on Semantic Web standards, components and scientific workflows. Individual sensors or networks are encapsulated into a specific kind of component – DCC – which supports homogeneous access to data and software. DCCs are themselves handled by scientific workflows that provide facilities for controlling data production, integration and publication. As a result, applications that require sensor data will instead interact with workflows, being liberated from concerns such as sensor particularities, or provide separate handlers for real time streams. The report also presents initial implementation results.

*Part of this text was submitted to the VLDB 2007 Conference. This work was supported by Fapesp under the grant number 2004/14052-3 and partially financed by a CNPq grant, the WebMaps II project and an HP Digital Publishing grant.

# 1 Introduction

Wireless sensor networks, i.e., networks of communicating small sensing devices, powered by batteries and with limited storage and processing facilities, are subject to intense research. Network nodes are frequently heterogeneous, generating distinct kinds of data at different time intervals. From the data perspective, challenges include dealing with integration of heterogeneous sources, data redundancy, data streams and real-time data, all subject to node, sensor and communication failures. From the network point-of-view, challenges comprise physical device management, dynamic reconfiguration of nodes and network, and support to different simultaneous applications, among others.

Many of these problems are being studied under distinct research perspectives. Examples are: stream data processing [9, 103]; communication protocols [5]; event detection [1]; support to simultaneous applications [57, 103]; data fusion and summarization [38, 49, 71]; or power management [70, 85] and energy efficiency [24, 49, 83].

Middleware-like solutions, e.g. [2, 10, 46, 50, 110], are frequently proposed to enable applications to access the sensing data sources. These solutions usually consist in offering APIs to allow applications to access the sensor produced data. Every time there is a need for new data processing functions, it is necessary to adapt or extend the middleware.

Our main objective is to provide means to let applications transparently access sensor network data, regardless of sensor physical characteristics, specific middleware programming environments, or data publication formats. The solution is based on a framework that relies on two main axes: (a) the use of a special kind of component paradigm – *Digital Content Components* (DCC) – to homogeneously encapsulate sensor data, the software that manipulates the data and access to sensors; and (b) the use of scientific workflows and their composition to coordinate sensor data manipulation and publication procedures.

DCCs [78–80] provide mechanisms for uniformly encapsulating both data and/or data processing units and for composing them into more complex elements. They offer homogeneous access interfaces to data, data sources and software, thereby helping solve interoperability issues. Our proposal extends the scope of DCCs to let them handle sensors and sensor data. It furthermore takes advantage of our scientific workflow design and documentation environment to create a sensor data publication framework.

In order to illustrate our solution, we will use a real case study from agricultural (i) planning and (ii) monitoring [30]. Planning (i) involves integration and analysis of heterogeneous files containing sensor-produced data to recommend which is the best crop to grow in a given region. The main data sources are satellite-based sensors and networks of ground-based sensors (mainly rainfall and temperature). Satellite data is combined with historical time series of ground data and additional static sources (e.g., crop characteristics, relief, soil analysis) to derive a set of files that are the basis for the crop recommendation report. This is usually documented by means of maps. Once the crop is planted, monitoring (ii) concerns continuous use of the same kinds of sensing data sources, at different sampling time frames, to capture and control crop response to climatological (e.g., drought) and human management procedures (e.g., fertilizer usage). The goal is to maximize production while striving for ecologic balance. While data concerns in phase (i) can be mapped to a problem of spatio-temporal heterogeneous database integration, phase (ii) adds the issues of real-time data capture and management. Our framework, as will be seen, provides a homogeneous solution to both phases.

The main contributions are, therefore:
(1) providing homogeneous access to heterogeneous sensing devices;
(2) enabling applications to have multiple views of sensing data, by taking advantage of scientific workflows to mediate sensor data access. This fosters reuse of solutions for managing these data;
(3) eliminating the need for an application to concern itself with whether a data source is static or dynamic.

The report is organized as follows. Section 2 presents basic concepts and an overview of our proposal. Section 3 shows how we encapsulate data, data sources and software within DCCs. Section 4 discusses scientific workflows as a means to manage DCCs, and thus data production and publication. Section 5 concerns implementation issues, with preliminary results. Section 6 discusses related work. Section 7 presents conclusions and ongoing work.

## 2 Solution Basics

This section briefly presents basic concepts to our work, namely, sensing devices, semantic web services, the component model adopted (DCCs) and scientific workflow concepts. Then we give a general perspective of our solution.

### 2.1 Sensing Devices, Sensor Networks and Data Production

*Sensors* are devices capable of measuring physical phenomena in a given environment. These sensed physical manifestations, such as heat, light, sound, pressure, magnetism, are subject to signal processing functions. These functions create the output data that will be stored or transmitted to concentrators or directly to applications. Usually, data produced by sensors refer to a specific geographic region. A sensor can be classified [87] with respect to:

(i) mobility, i.e., change in its geographical coordinates: static or mobile;

(ii) data transmission mechanism: manual readings, wired or wireless network; and,

(iii) regularity: constant (*stream*), e.g., measuring air humidity every five minutes, or event-based, e.g., generating data only when moving objects are detected;

Sensors should be non-intrusive, i.e., change the environment being monitored as little as possible; they should endure in hostile environment conditions; they should be energy efficient, being capable of lasting long periods of time without human intervention; and, they should be able to transmit the collected data within given time constraints. Another important issue is the quality of the collected data, e.g., degree of uncertainty or probability of correctness. This directly affects the applications that will use the data.

Sensing devices range from satellites (few in quantity, large and expensive), to radio-frequency tags (lots of tiny and cheap units) [39], as shown in Figure 1.

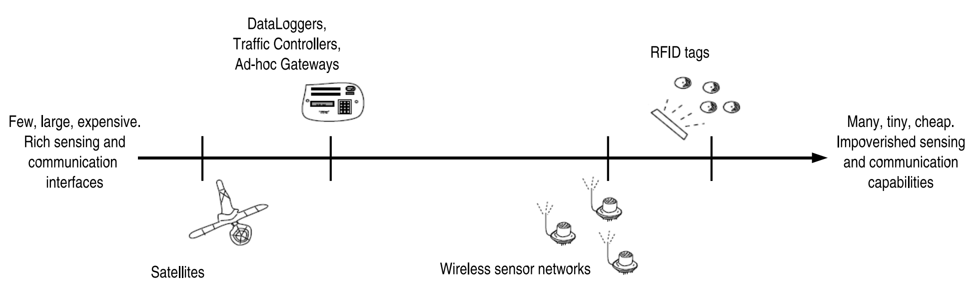


Figure 1: Sensing devices spectrum: diversity of applications (adapted from [39])

At one end of the spectrum, satellites daily produce large quantities of sensing data. There were initiatives for dealing with these data using relational database management systems (DBMS) [39]. However, they were mostly discarded for a number of factors. First, a high percentage of the generated data form the so-called “large objects”, mainly imagery data, instead of the textual data and metadata, which can be handled efficiently by a

DBMS. Second, assuming ways of dealing with this problem using hybrid approaches and data filtering, there are a few users for systems based on this kind of data; and these users are highly specialized, such as spatial agencies. This kind of user commonly develops personalized in-house solutions for dealing with these data. These two factors gave DBMS providers little reason for investing in such products [39].

At the other end of the spectrum, appear the radio-frequency tags (RFID). These devices are extremely cheap, to the point of being considered as replacements for conventional price tags. This simplicity, however, increases the cost of readers for these tags. Thus, there are still limitations to moving from bar-code based tagging. According to [39], the killer application for RFID tags is supply chain management.

In the intermediate area of the spectrum, there are several research challenges, specially those concerning sensor networks. This kind of sensing devices lever several new applications and research. The next section details this kind of sensing device, which is also one of the main topics of the thesis.

### 2.1.1 Sensor Networks

*Sensor Networks* (SN) are sets of sensors capable of communicating with other sensors and with controller nodes [22, 87]. With respect to the classification presented earlier, this kind of sensing device basically excludes manual reading. Usually, a set of sensors is coordinated by a base station that, in turn, is connected to a network server [53]. The latter is usually externally connected to a network, possibly using the Web to make the data available.

*Wireless Sensor Networks* (WSN), that use radio-based communication, are one of the most common kinds of SN, since they are much more flexible. They can be distributed in a region in order to provide sensing data, but they pose harder management problems, such as route choice and maintenance.

Calibration or verification sensors are also usual within a network. They can be deployed as part of the network or as a completely independent network. These secondary sensors provide reference values for checking the readings and are more reliable, being, for instance, tested individually.

The applications for sensor networks vary greatly. Currently, the main areas involve monitoring environmental, climatic and biodiversity conditions [3]. These applications also require georeferencing at data collection time.

*Access points* (AP) are responsible for connecting devices among themselves. This may be independent from a wired network and data sinks to the point in which the AP has mobility itself [14]. *Base Stations* (BS) are the devices that connects the sensing devices to a land-line network and the data sink, i.e., the destination of the read data. It is very usual that an AP and a BS are deployed in as single device. In modeling a WSN this combination is often called *sink* [4].

### 2.1.2 Ad Hoc Networks

Research in *Ad Hoc Networks* is very close to that on WSN as almost all WSN use ad hoc organizing techniques for establishing communication routes.

However, there are significant distinctions that must be considered, as made clear in [102]. The table in Figure 2 shows the main characteristics that differentiate between them.

| | WSNs | Wireless *Ad Hoc* |
|---|---|---|
| **Number of nodes** | Large; hundreds – thousands | Small |
| **Node density** | High | Low (relatively) |
| **Data redundancy** | High | Low |
| **Energy supply** | Non-rechargeable; non-replaceable | Rechargeable and/or replaceable |
| **Data rate** | Low; 1 – 100Kbps | High |
| **Node mobility** | Low | May be high |
| **Flow direction** | Mainly unidirectional | Bidirectional |
| **Packet forwarding** | Many to one (data centric) | End-to-end (address centric) |
| **Query nature** | Attribute based | Node based |
| **Query dissemination** | Broadcast | Node to node or broadcast |
| **Addressing** | No globally unique ID | Globally unique ID |
| **Duty cicle** | Might be as low as 1% | High |

Figure 2: Ad hoc and wireless sensor networks differences (adapted from [102])

First there is the number of nodes, which in WSN can reach the level of tens of thousands in the more ambitious applications, while in the Ad Hoc networks the number stays closer to tens. The node density and the data production redundancy can be extremely high in WSN; and the batteries are commonly not worthy of being recharged or replaced. The data rates required within the WSN are not too high, while in Ad Hoc networks it is constantly increasing. The data flow is mainly unidirectional in WSN, from nodes to base stations. The queries (or data access) in WSN is attribute based, while in Ad Hoc networks the node is almost always important. In WSN it is not common to uniquely identify each node and the nodes can stay in stand-by or even off for long periods; both characteristics are inverted on regular Ad Hoc network devices.

### 2.1.3 Data Management Aspects

Two approaches are usual in treating data produced by sensors, depending on the application domain [15]. The first consists in viewing the data as a non-interruptible source (exclusively as a *stream*), accessed as the data are produced. This first form, called a "monitoring query" is useful only for data that must be consumed in the moment that they are produced, such as detection of movement in a security system. In the second form, the data are stored in databases, usually with temporal (date and time) and spatial (place) information of when and where the data were collected. This approach is used either in stream form or by querying the sensors, and is useful when the data are valid throughout a period of time, such as the monitoring of pluviometric indexes.

The thesis will deal with both types of data treatment, emphasizing the second one. Even if the management of the devices in all their levels [73–75, 84] may affect data management, this aspect will not be directly considered; they will only be taken advantage of. Problems arise when storing the data, such as [51]: what to be stored and how to fill in the gaps left by several types of reading errors. The former deals with selecting what is going to be stored, since the amounts of data produced may be far too large to be fully stored. The latter has to do with error handling.

In this scenario, sensor generated data are continuously filtered and stored and, at the same time, their quality must be evaluated. Furthermore, querying these data presents the problem of combining data access mechanisms for both stored and stream data. In order to do that, a hybrid scheme must be elaborated combining usual database access with real time access. [15] proposes a form of combining these two techniques, providing a uniform access to the data.

Some of the unique characteristics of querying these data are [15]:

(i) Monitoring queries are usually long term queries;

(ii) The result of a query may be itself a stream or the triggering of external events;

(iii) Queries may need to establish correlations among the data produced by a series of different sensors;

(iv) Most queries restrict the set of sensors to be considered using some kind of constant, e.g., geographic location.

As an example, consider a set of heat sensors (heat readings and abnormal temperature detection) spread throughout a factory. Sample queries, showing characteristics above, are:

a) "Get all the abnormal temperatures read by all sensors";

b) "Get, every minute, the temperature measured by all sensors on the third floor";

c) "Generate a notification every time that two sensors within five meters from each other simultaneously get an abnormal reading";

d) "Every five minutes, get the maximum temperature read in the last five minutes";

Projects that focus on building frameworks for WSN include *WINS* [69] and *Smart Dust* [42]. The COUGAR system [111] is a database management system that proposes a solution combining the relational approach with distributed data sources. The Quasar system [45] has a similar goal, providing a means of accessing sensor data through a SQL-like query language.

Our work has a different approach for dealing with these problems. Data will be accessed through homogeneous interfaces in components and specific components built over data concentrators and DBMSs. Queries will be posed via requisitions to these components, and will be answered with the results of the execution of a specific workflow within a Workflow Management System.

### 2.1.4 Sensors and Databases

Technologies for managing sensing data have been mostly developed independently. There is a current trend to change this scenario. First, there is the reduction in cost and size of sensing devices. Thus, many new applications can now be considered, requiring novel schemes for managing the produced data. The second factor is the convergence of sensor and computer networks. Having communication and processing capabilities, sensors begin to form ad-hoc networks that, from the communication perspective, can be integrated with conventional networks. This poses issues in the area of computer networks, which will not be discussed here. The thesis focus lies in the data production perspective, having many open problems yet to be tackled.

One issue has to do with how to use the processing capabilities of the sensor nodes to manage their own data. Two ways of dealing with this question have shown promise, using concepts similar to those of querying DBMSs [39]: (i) independent queries to each node, subsequently combining the outcome to obtain one result; (ii) queries directed to a set of sensors, treating this set as a single source. Both approaches have limitations; one serious problem is scalability, since the number of nodes can range from hundreds to billions.

A second issue has origin in data acquisition. Differently from conventional DBMSs, in which all the data are available at query time, sensor networks can be reconfigured to produce data at different rates or answering to different stimuli in a query. Furthermore, there is the possibility of asynchronous data transmission, as in event-based data reading, and based on the sensor life time, adequating the rates to user needs and the node duration. Thus, data heterogeneity is combined with periodicity heterogeneity, complicating the integrated analysis.

Yet another aspect is energy management in large networks of small devices. The transmission costs can be orders of magnitude higher than the local processing costs, in terms of energy. Having the data pre-processed before transmission can be interesting. The problem is how to do this processing considering that each sensor has a very restricted view of the whole network. This is the main issue that *data fusion* deals with.

Energy management issues will not be directly addressed in this work.

### 2.1.5 Data Fusion

During the data gathering, the sensing devices translate physical signals (analogical) into digital signals. This involves the processing of the signals and treatment of noise in the readings. This task can be done with different levels of precision and individually or in groups.

The main functionality of the sensors it to provide the values of the physical amounts read. To do that, it is possible that the sensors themselves reach an agreement about the measured value, processing the gathered data; or they can directly transmit all the read values without any processing; or even use some mean between these two approaches. This is what characterizes, basically, the *data fusion* problem, as shown in Figure 3. Choosing how and how much of the data is going to be processed before transmission represents a determining factor in a sensor network, influencing, for instance, the selection of the communication protocol [33], the reconstruction of the transmission topology [58], or the lifespan of the batteries.

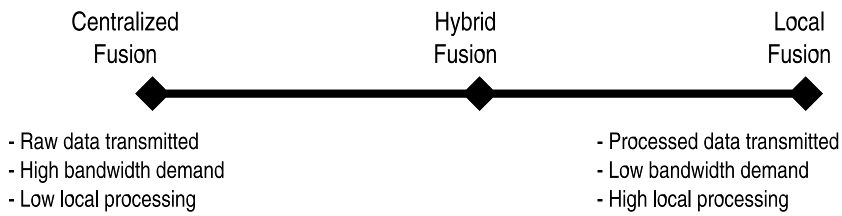


Figure 3: Strategies for sensor data fusion (adapted from [64])

## 2.2 Semantic Web Services

The *Semantic Web* [13] is a evolution of the Web [12] is based on two main ideas: the adoption of common vocabularies and the annotation of resources using terms of these predefined vocabularies. A *resource* is anything that can be identified on the Web, even if it cannot be directly retrieved on the Web [52]. Examples are a shopping Web site, description about products on that site, a person's site, this person's shopping preferences, a service and this services quality references, etc. The goal is to increase the automation level of Web systems.

The common vocabularies are often structured in *ontologies* [23, 29, 34–36, 43, 90]. There are several diverging definitions for the term *ontology* [23, 34, 59, 100]. However, all these definitions have a common core of concepts, which is going to be used here: domain of discourse, commonly agreed vocabulary of terms and an explicit representation.

The second component of the Semantic Web is the annotation of resources using the terms previously defined in the ontologies. In order to do that, there is a need for standard ways of referencing those terms and of attaching these references to a resource. Resources are univocally identified by URIs – *Uniform Resource Identifier*. A *URI* is a compact string of characters for identifying an abstract or physical resource [11].

The two primary facets of the Semantic Web (ontologies and annotations) involve data representation and interpretation. There are several problems related to data representation, ranging from formats to the interpretation of a given term. Several standards are being used/proposed to address these issues. The standards can be seen as organized according to a stack, where the lower levels support the upper levels, making it possible to describe data with several degrees of flexibility. The left side of Figure 4 depicts this stack (*Data Description Stack*). The first layer is character encoding, where Unicode encoding is used for processing textual data in any system.

Next comes the syntactical basis for representing data in a semi-structured fashion, using XML (Extensible Markup Language) and its associated standard for namespaces and definition of types, XMLSchema. RDF (Resource Description Framework) addresses semantics requirements. It forms a foundation for processing metadata and to express relationships. The Ontology Vocabulary layer uses an ontology language to formally describe

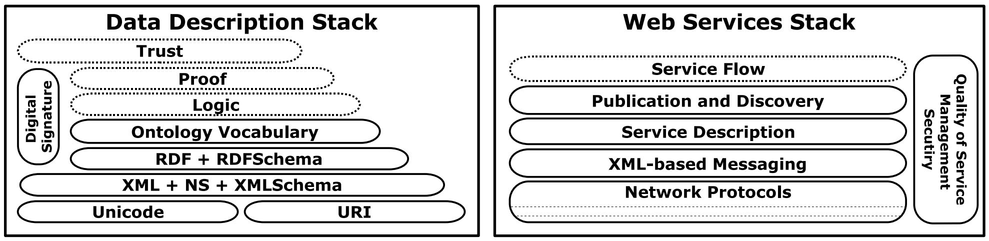


Figure 4: Semantic Web data and services description stacks

the meaning and the terminology used in Web documents. OWL (Web Ontology Language) is likely to become the standard for this layer. The Digital Signature layer gives data a certificate that guarantees their origin. The Logic layer establishes a logical system through which the Proof layer can perform inferences about the data represented in lower layers. Digital Signature combined with Proof ensures the validity of the information to be derived in the Trust layer.

Beyond data representation, in order to achieve an active state, there is a need for automated machines to do the work on processing these data. This is the executional perspective of the Semantic Web. It is concerned, among others, with having the parts described and the accessible resources and how to make them work together in an automated way. Here is where the *Web services* [7, 95] come to be.

Broadly speaking, a Web service is an application that is accessible to other applications via the Web [7, 8, 55]. This definition turns virtually anything with a URI into a Web service. Another specification chooses more specific terms to define Web services [89]: "*self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces.*" The definition from the World Wide Web Consortium (W3C) [98] is often regarded as the official one [99]: "*A Web Service is a software application identified by a URI [11], whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.*" This definition includes all we think is needed to define a Web service: an *application*; an identification (*a URI*); the fact that interfaces can be *defined, described and discovered*; uses *Internet-based protocols* and *XML artifacts* and *messages*. Other definitions [60, 104] specify further into the protocols that must be used. However, we agree with [7] in that those other protocols are not justifiable as part of this definition.

Many argue that the Web services technology should be seen as being completely independent from the Semantic Web efforts. Others, and we share this second point-of-view, say that Web services are the most promising way of making it possible for enabling the Semantic Web.

There are several efforts in standardizing Web services related technologies [93, 97], such as the specification of languages for describing Web services. There are many other specific issues to be developed, such as stateful Web services and transaction support.

The standards for Web services can also be portrayed in terms of a stack defining distinct service layers (right ide of Figure 4). The XML-based Messaging layer provides a message formatting protocol, based upon usual network protocols, offering a high level abstraction for composing and exchanging messages formatted in an XML compliant language. SOAP (Simple Object Access Protocol) is the standard recommended by W3C for this layer. The Service Description layer provides a way to describe Web Services capabilities and communication interfaces. WSDL (Web Service Description Language) is the standard for this layer. Service Publication and Discovery using UDDI (Universal Description, Discovery and Integration) as a standard provide means to make Web Services reachable. Quality of service, security and management are issues that must be considered at every layer of the Services stack.

The conceptual separation between data and services induces an implementation for the Semantic Web. On the one side there are the data that should be semantically understood in the same way wherever they are used. On the other, pieces of software should provide a satisfactory degree of automation when handling these data.

Many institutions are adopting Web services as a solution for several interoperability problems of diverse nature, profiting from the infrastructure provided by the Web and the connectivity of the Internet. Limitations on the functionality provided by this new technology is motivating further developments. One such direction is concerned with the composition of services in order to achieve more complex behaviors [61, 65, 96]. Another deals with the interoperability of Web services, trying to apply the same semantics-based solution used for data resources [27, 63, 94]. The efforts on the second problem gave origin to the so-called *Semantic Web services*, which are in an early stage of development.

There are several proposals for addressing semantics with respect to Web services. One is the OWL-S language that is being proposed as a complement to service description, publication, discovery and composition standards and can even replace them at some degree. Other languages serving the same purposes are WSDL-S (Web Service Description Language Semantics) [94] and the WSMO/WSML/WSMX initiative [26–28].

The ontologies described earlier in this section are directed to data description. When applying semantics-based techniques to Web services, another kind of ontology can be classified, the *service ontology*. In contrast to that, the data description ontologies are being called *domain ontologies*.

## 2.3 Digital Content Components

A *Digital Content Component* (DCC) is a unit of content and/or process reuse, which can be employed to design complex digital artifacts [77–80]. From a high level point of view, a DCC can be seen as digital content (data or software) encapsulated into a semantic description structure. As shown in the example in Figure 5, it is comprised of four sections:
(i) the content itself (data or code, or another DCC), in its original format. In the example, three files with rainfall maps of São Paulo state in Brazil created from processing satellite data;
(ii) the declaration, in XML, of an organization structure that defines how DCC internal elements relate to each other (here, describing the map files structure);
(iii) specification of an interface, using adapted versions of WSDL and OWL-S – in the example, the `getQuantity` and `getMap` operations;

(iv) metadata to describe functionality, applicability, etc., using OWL (in the example, the DCC is declared as belonging to the `MapSet` class).
Interface and metadata are linked to ontology terms – e.g., the output of `getQuantity` is an integer that correspond to a measure of rainfall, as defined by the "Rainfall" concept of NASA's SWEET [72] ontology.

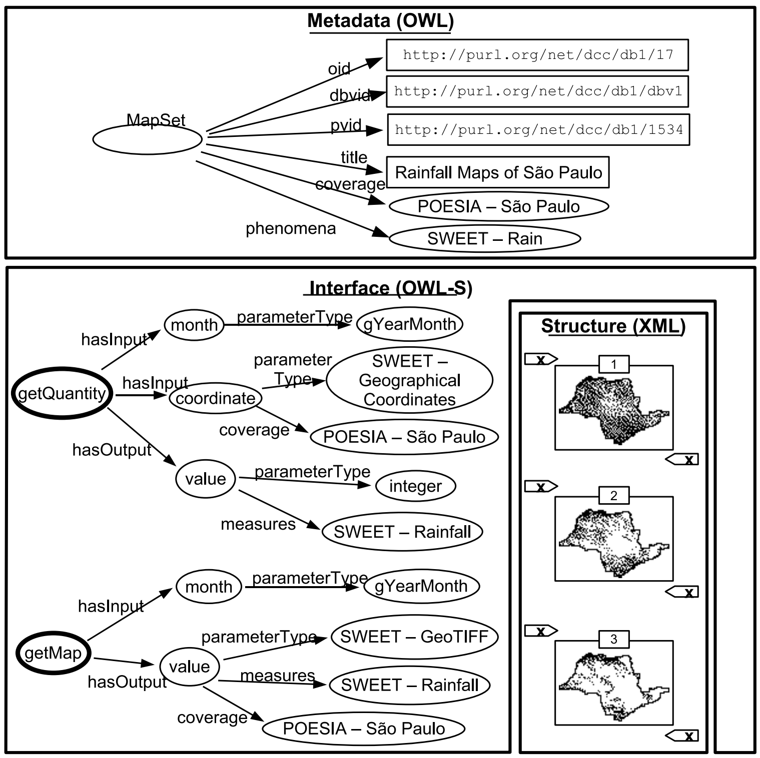


Figure 5: DCC schematic structure (from [79]).

Any DCC can encapsulate a content, which is formed by other DCCs or by *Atomic Digital Artifacts (ADAs)*, as shown in Figure 6. An ADA is a piece of digital content of any kind (e.g. image, video, text, software). It is "atomic" under the DCC model point-of-view, since its content cannot be divided. Part (a) of Figure 6 shows a DCC that directly encapsulates a set of GeoTIFF image (a bitmap format that associates geographic coordinates to pixels) files, while in part (b) the same images are encapsulated in one DCC each and the resulting DCCs are encapsulated in another DCC.

DCCs are to be stored in repositories available on the Web. Interface and metadata sections – respectively (iii) and (iv) – are used to help retrieve the appropriate DCCs from the repositories and reuse them [78]. There is furthermore a DCC infrastructure that comprises an architecture to assemble DCCs into a desired product. A *DCC composition* is considered to be any digital artifact built combining DCCs, and can vary from a multimedia document to a software application.

The DCC model was inspired by the software component paradigm of software engi-

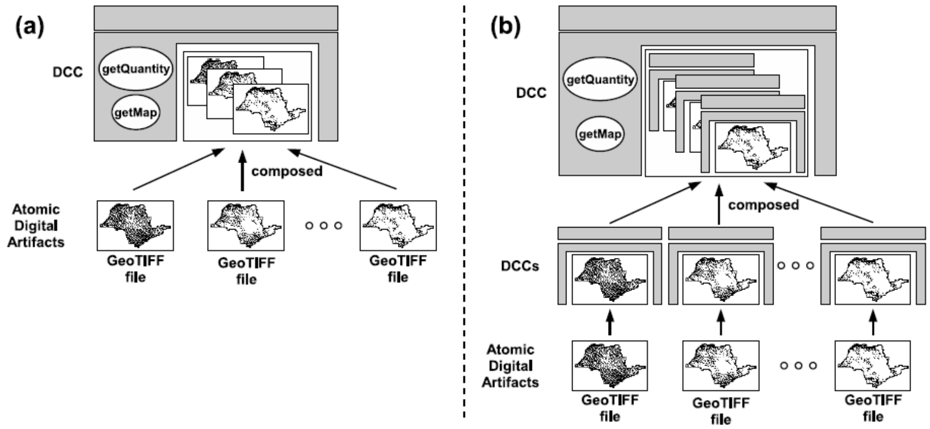


Figure 6: Composition of (a) artifacts and (b) DCCs (extracted from [76])

neering. However, unlike software components, DCCs do not need to encapsulate binary program code to be useful as part of applications. It is possible to encapsulate inside a DCC a multimedia artifact only, other kinds of executable content (such as software or workflows), or both, and use it directly to compose an application.

There are two kinds of DCC – process and passive. A *ProcessDCC* encapsulates any kind of process description that can be executed by a computer (e.g., software, sequences of instructions or plans). Their interfaces declare operations they can execute. Non-process DCCs, named *PassiveDCCs*, consist of any other kind of content (e.g., a text or video file); their interfaces declare their "*potential functionality*", in the sense that the operations can be requested from them, but their execution code is not part of the content (e.g., a video player software is not part of a video).

Since passive components by definition cannot embed executable code, operation implementations are encapsulated into special ProcessDCCs called *CompanionDCC* (e.g., a piece of music `M` stored in a PassiveDCC can be played by attaching `M` to a suitable music player in a CompanionDCC). The process of finding the appropriate CompanionDCC follows the notion of *content-type driven execution* [80]. The association between a PassiveDCC and its companion is achieved with help of ontology annotations.

One can progressively compose DCCs into more complex ones. The composition process is orthogonal to the nature of the DCC and is based only on interfaces and metadata. Thus, PassiveDCCs can be composed to create a more complex (data) component – e.g., akin to constructing complex objects in databases. Also, two ProcessDCCs can be composed into another (executable) component – e.g., akin to software component construction. Moreover, process and PassiveDCC can be composed to construct an application. This allows using the same composition mechanism to combine data and software, supporting integration of heterogeneous sources and interoperability [80].

When using DCCs, a basic infrastructure must be present, which is responsible for their creation, coordination and publication. An important part of this infrastructure is the DCC repository, which stores and provides access to DCCs. The DCC infrastructure also provides means of communication for components implemented in different languages and running on different platforms. We refer the reader to [79, 80] for details on this infrastructure.

## 2.4 Scientific Workflows

The *workflow* technology is fairly known nowadays. Nonetheless, there are several issues yet to be solved, mainly regarding its formalization and the interpretation of its basic concepts, as we shall further explore in this section.

The first known application of workflows was in office automation, to aid the organization of processes, such as sales or accounting. There is also a considerable intersection of workflows with the BPM (Business Process Management) area, the former being more concerned with the models and the latter with improvement of processes. Currently, the technology has been broadened to several areas, including software development (in composing software pieces) [61, 107], controlling production processes in factories, and knowledge management techniques. Another important area of application of this technology is the documentation of scientific experiments [54, 65, 66, 101], where workflows can be used from experiment specification and execution to its validation.

Among the advantages of the adoption of the workflow technology, [6] and [67] list the increase in the efficiency in the application of resources, task automation and improvement in organization aspects such as reduction of task allocation faults or task stalling. There is also improvement in data management and overall process control [18, 19].

With the growth of the application of the workflow related technologies, the heterogeneity problem arose. To deal with these issues, associations of users were formed, such as WfMC (*Workflow Management Coalition*) [105] and BPMI (*Business Process Management Initiative*) [16]. Several proposals for standards regarding workflows are being developed currently [17, 61, 107]. These proposals, however, are subject to dispute and are far from becoming *de facto* standards.

In the remainder of this section we will first describe workflow related concepts and vocabulary (section 2.4.1). Then we proceed describing Workflow Management Systems (WFMSs) and their functioning (section 2.4.2), showing how the concepts work. Last, we briefly describe the WOODSS project (section 2.4.3), which is currently being integrated with DCCs.

### 2.4.1 Workflow Concepts

In this section we will first see some basic concepts related to workflows, constructing a basic vocabulary. Then we proceed to the description of the levels of specification of a workflow, how we use data types and taxonomies to make interfaces more precise and how a workflow can be made available on the Web.

**Process.** A *process* is a set of inter-dependent steps needed to complete a certain action, usually within the context of an organization;

**Workflow.** A *workflow* is a model of a process, representing it partially or completely. There are two phases of a workflow life-cycle: (i) specification time, while it is suffering changes; and (ii) execution time, when a workflow is put to execute the designed task. These two phases are not all that clear-cut; in some cases an executing workflow can be altered, as will be seen;

**Activity.** An *activity* can be a basic unity of work or another workflow. A basic unity of work, or a *basic activity*, exists within the scope of a process and can be either automated or manual. When an activity is another workflow it is called a *subworkflow* and is explained next. Activities are the basic elements for constructing workflows;

**Subworkflow.** A *subworkflow*, or *subflow*, is a workflow that is used within the scope of another workflow, represented as an activity. In our approach [65] an interface encapsulates a (sub)workflow specification, allowing access to it without violating encapsulation;

**Transition.** A *transition* is a directed connection between two activities, establishing a dependency relation between them. This relation expresses the flow sequence of a process and can also express a data dependency between the activities;

**Pre-condition.** A *pre-condition* is associated with an activity and is a logical expression that must be evaluated as true to allow the execution of the activity. System variables (such as time or resource availability) and data variables can be used to express a pre-condition;

**Post-condition.** A *post-condition* is associated with an activity and is a logical expression that must be evaluated as true to allow the flow to carry on after the execution of an activity. System variables (such as time or resource availability) and data variables can be used to express a post-condition;

**Activation condition.** An *activation condition* is associated with a transition and must be evaluated as true to allow the transition to pass on the flow to the next activity. System variables (such as time or resource availability) and data variables can be used to express a activation condition;

**Agent.** An *agent* is a person (manual) or a system (automated) responsible for the execution of one activity on an execution of a workflow;

**Role.** A *role* specifies the expected behavior of an agent when executing an activity on an execution of a workflow;

**Cycle.** A *cycle* or a *repetition* is the execution of one or more activities of a workflow a certain number of times. We deal with cycles establishing the number of repetitions combined with the use of pre-conditions on activities and activation conditions on transitions [65];

**WFMS.** A *workflow management system*, or *WFMS*, is an automated software to specify, store, retrive and instantiate (execute) workflows (see section 2.4.2). The execution of one workflow is controlled by one execution engine, which can interpret the workflow specification and can interact with agents;

**Scientific Workflow.** A *scientific workflow* is the model of a process that describes a scientific experiment [101]. A few characteristics are inherent to this kind of workflows: (i) high degree of flexibility; (ii) uncertainty; (iii) exceptions; (iv) unfinished executions; (v) specification from the execution; (vi) on-the-fly customization; (vii) highly data centric flow.

Most proposals for workflow construction are based on the notion of patterns [91, 92, 109] and composition. We derived a specification framework centered in the ideas of reuse and storing workflow elements in databases. This is based on a notion of specifying workflows in abstraction levels [54, 65]. The levels are:

(i) activity types;

(ii) abstract workflows;

(iii) concrete, or executable, workflows;

(iv) workflow instances;

Figure 7 illustrates the levels (i), (ii) and (iii).

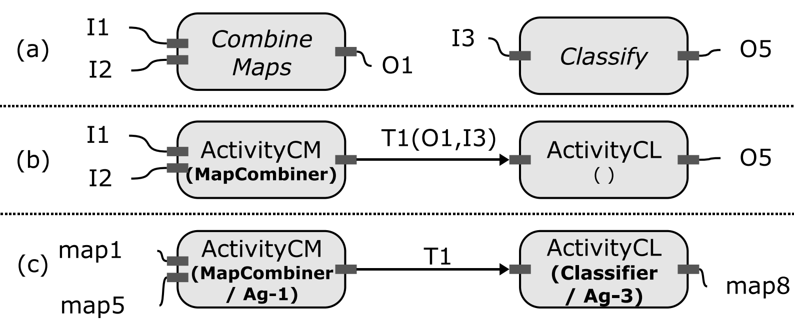


Figure 7: Workflow specification levels

The model induces a methodology for correctly constructing workflows, whereby users must first specify types of workflow building blocks, next combine them into abstract specifications (*abstract workflows*), and finally instantiate these specifications into executable (sub)workflows (*concrete workflows*). All these documents are stored in a relational database.

The building blocks can be split in three major levels, which can be refined into many intermediate levels. The three levels reflect the methodology – see Figure 7. The first level corresponds to definitions of data and activity types, and roles to be fulfilled by agents. Figure 7(a) shows two activity types (*Combine Maps* and *Classify*), typical of environmental tasks. Activity types are specified in terms of their interfaces, which are based on the previously defined data types. In the figure, *Combine Maps*'s interface has two inputs, *I1* and *I2*, and one output *O1*.

Activity and data types are used to create blocks at the second abstraction level, where a workflow structure – the abstract workflow – is specified, through transitions and dependencies among activities – e.g. the activity labeled *ActivityCM* is of type *Combine Maps* – see Figure 7(b). At this level, it is also possible to refine activity types by associating roles to them – e.g., *ActivityCM* is associated with role *MapCombiner*. Transition *T1* connects interface elements *O1* and *I3*. The specification of types and of abstract workflows captures the notion of *designe workflow* independent of execution aspects (agents and data connections), allowing design reuse.

The third level involves creating an executable version of an abstract workflow – i.e., the concrete workflow. This is achieved by associating agents with activities and actual data sources with activities' interfaces. Figure 7(c) shows *map1* and *map5* data sources as the input parameters for the *ActivityCM* activity, which can be executed by invoking the specific software *Ag-1*.

The fourth level considers intances of workflows, i.e., workflows running in a WFMS.

Moreover, we introduced the notion of design workflow. A *design workflow* is an abstract workflow intended to reuse and stays within the limits of levels (ii) and (iii). Its major goal is to make it possible to reuse several aspects of a process specification, such as solution overview, manipulated resources, process organization, resource combination, and precedence factors, and others.

### 2.4.2 Workflow Management Systems

A Workflow Management System is mainly responsible for workflow execution. The WfMC has defined an abstract model representing a generic architecture for a WFMS, depicted in Figure 8. The model specifies the general organization of the main modules of a WFMS.

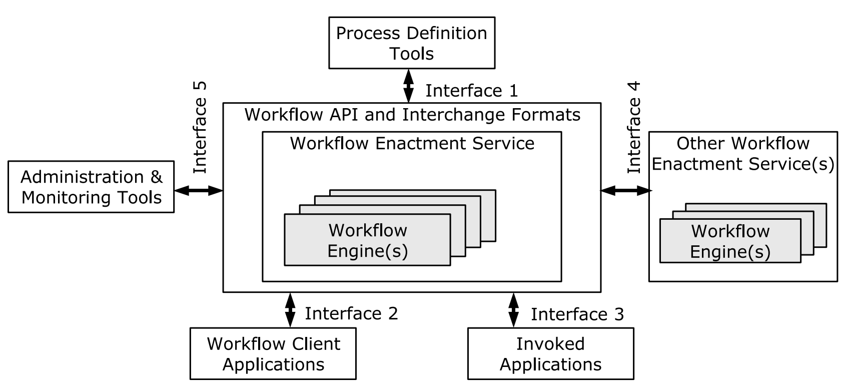


Figure 8: WfMC's architecture model for a WFMS (extracted from [106])

The model comprises an execution (enactment) service, a specification tool, a management and monitoring tool. It also defines WFMS client applications and invoked applications, which can be considered as external to the architecture. Other execution services may be interconnected, even being able of distributing the execution of a workflow.

An execution service is responsible for instantiation of workflows. It is composed by execution engines. A new execution engine is created whenever a new workflow is instantiated so that each workflow is controlled by one execution engine.

A specification tool serves to specify workflows. This tool often offers an interface for the execution service and is integrated with the management and monitoring tool. A management and monitoring tool supports starting an execution of a workflow and showing its execution status.

The client applications use the WFMS features, invoking the operations offered by it. On the other hand, the invoked applications are called by the WFMS' execution service in order to accomplish tasks specified in the workflow within activities.

### 2.4.3 WOODSS

WOODSS (*WorkflOw-based spatial Decision Support System*) is a scientific workflow infrastructure developed at IC - UNICAMP, initially conceived to support environmental planning activities. It was implemented on top of a commercial Geographic Information System (GIS) and has been tested in several contexts, mostly within agro-environmental applications.

The original idea was to dynamically capture user interactions with a GIS in real time, and document them by means of scientific workflows, which could then be re-executed invoking the GIS functions.

WOODSS translates these interactions into elements of a scientific workflow, which is stored in a relational database. Workflows are used by WOODSS in three roles: (i) as a means for documenting a scientific experiment; (ii) as high-level specifications of some simulation model; and (iii) as executable parametrized specifications of decision procedures, which can be reused and adapted for similar situations.

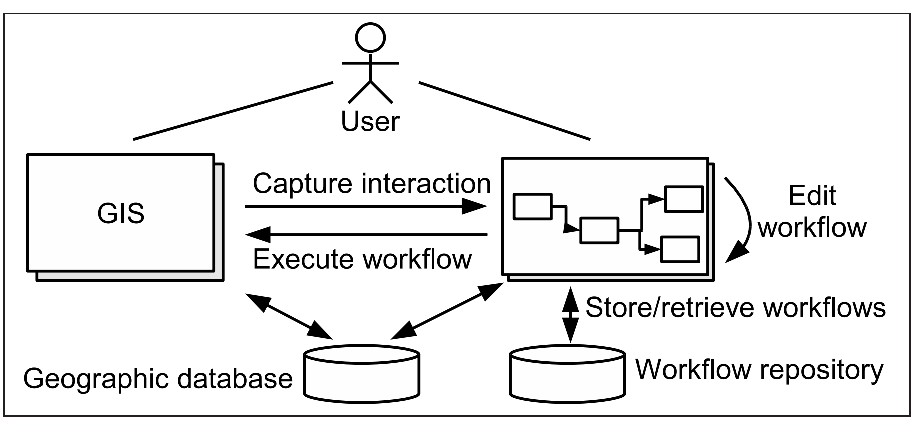


Figure 9: WOODSS interactions - adapted from [44]

In environmental applications, user interactions with a GIS express models for solving a given problem. Figure 9, adapted from [44] illustrates the interaction modes of WOODSS.

In the first mode, users interact with the GIS, to implement some model, whose execution is materialized into a map. WOODSS processes this interaction and generates the corresponding scientific workflow specification, stored in the Workflow Repository. In the second interaction mode, users access WOODSS' visual workflow editing and annotation interface to query, update and/or combine repository elements, annotating and constructing workflows that can be stored in the repository and subsequently executed invoking the GIS. This allows workflow evolution and lets scientists find out about previous solutions to a similar problem. The geographic database contains domain-specific data.

The present version of WOODSS is implemented in Java$^{\text{TM}}$, with the repository in PostgreSQL [68]. It contains scientific workflows (and parts thereof) and associated annotations (keywords, free text, metadata and domain ontologies [29, 30]). The graphical interface allows user-friendly workflow editing and manipulation. More details appear in [44, 54, 66, 81].

We recall that the idea behind this thesis is to encapsulate sensors within DCCs and manipulate sensor produced data using workflows. As will be seen in section 5.1, we will take advantage of WOODSS, integrated with the DCC support infrastructure, to achieve this goal.

## 2.5 Overview of Our Solution

The usual approach to access sensor generated data is either to communicate with the sensor directly in its specific protocols or to use a wrapper implemented for each type of sensor. We propose instead to encapsulate all the data production behind DCCs. Besides providing data access, a DCC can aggregate several functionalities. Examples include data delivery rate, stream data management, data annotation, and others that will be explained throughout the text.

Figure 10 gives an overview of our solution, which involves encapsulation of sensing devices and data into DCCs (section 3) and management of these encapsulated data using scientific workflows (section 4). The figure is organized in layers to help the explanation.

The bottom layer contains the sensors, their auxiliary devices and communication features. The second layer, detailed in section 3, contains the DCCs that provide access to sensor data, called *SensorDCCs*, which play a role comparable to that of a mediator to access the data produced by a sensor. One SensorDCC can encapsulate one (DCC **B**) or more sensors (DCC **A**) by, for instance, encapsulating an access point of a wireless sensor network (DCC **C**). Data can be delivered to the next layer in its original format (DCC **G**) or encapsulated in another DCC, as done by DCC **C**, delivering DCC **D**. DCC **D** is an example of a *SensorDataDCC*, a PassiveDCC used for sensor data encapsulation and annotation. Layer 3 (**Y** and **Z** in the figure) has data organization and centralization features (pre-processing, summarization and fusion). Finally, the fourth layer has the publication and access control features, and offers raw and processed data to applications in Layer 5. As will be seen, the features on layers 3 and 4, detailed in section 4, can be implemented as software or workflows (our proposal). Applications in Layer 5 and are regarded as clients of Layers 3 and 4. This report is not restricted to any specific application domain; rather, it is centered on the interfaces offered by several kinds of DCC to provide homogeneous access to data.

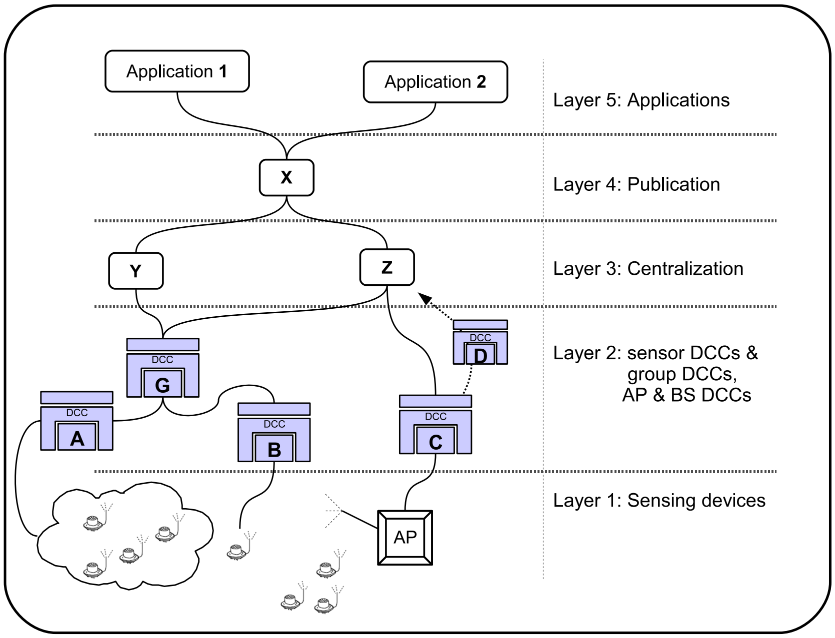


Figure 10: Management layers

# 3 Encapsulation of Resources

This section explains the encapsulation of data (section 3.1), data sources (section 3.2), i.e., sensing devices, and manipulation software (section 3.3) within DCCs. Data encapsulation consists of wrapping data with accessibility rules (potential functionality interfaces), descriptive metadata and structure. The encapsulation of sensing devices consists in offering a uniform access to these data sources. Encapsulation of processing software in the same framework makes it possible to homogeneously access the functions. Data, devices and software are accessed through the same interface scheme.

## 3.1 Encapsulating Sensor Data

Sensing data is encapsulated into a specialization of a PassiveDCC – *SensorDataDCC*. A satellite image, or a file containing a temporal series of rain data are typical examples of data to be encapsulated into a SensorDataDCC. These DCCs can be used for homogenizing access to data sources used in crop planning (our running example from section 1). Like any DCC, a SensorDataDCC is annotated using ontology terms, e.g., data type, the physical phenomenon being measured (temperature, level of moisture, etc), the geographical location of the reading, etc.

An issue here is the granularity of the encapsulation. A SensorDataDCC can encapsulate: (i) a single measure received; (ii) all measures received within a time frame; (iii) all measures received within a memory window, e.g., only a full buffer generates a new SensorDataDCC; (iv) one of the results of a query; (v) all of the results of a query. There is also the matter of dealing with stream data, which we explore next.

**Stream Data Encapsulation**

DCCs have, by nature, a closed scope specification, i.e., they are clearly defined and can only be changed by an authorized user. Thus, they do not directly support data stream encapsulation. We propose two approaches for dealing with this incompatibility: data forwarding and data caching. Since SensorDataDCC are passive, both approaches require a ProcessDCC – say, `P` – to pre-process the data to be encapsulated.

In the first approach (forwarding), `P` accesses the stream source only when needed, i.e., when an operation is posted to `P`. In this case, `P` ignores the data stream production until data is requested. A query could be time constrained,i.e., collect five minutes of data, or unbounded, i.e., the data is transmitted until the invoker issues an explicit stop signal.

In the second case, `P` polls the stream source in order to acquire the data at some constant rate, and stores it somewhere. There are at least four ways for `P` to store the data: (i) directly into a database management system; (ii) in files, using some kind of structure for the data; (iii) `P` sends on the data to one or more additional ProcessDCCs that will take care of the data from that point on; (iv) `P` keeps the data available (e.g., in main memory) using a window of limited size combined with a discarding policy, e.g., first-in-first-out, least-used, or least-recently-used. In case (ii), the files can be encapsulated by SensorDataDCCs.

## 3.2 Encapsulating Sensing Devices

Sensing devices are encapsulated within a specialization of a ProcessDCC, which we call *SensorDCC*. They are also annotated using the DCC ontology. Examples of annotations are: physical phenomenon reading capabilities; possible reading rates; distribution of the sensing devices (sparse, regular, unknown, etc); processing capabilities. Unlike SensorDataDCCs, which encapsulate static content elements, SensorDCCs are similar to proxies to accessing the data. Thus, they can be used for real time crop monitoring.

Since each kind of sensing device has a specialized format for outputting data, a driver must be built for each different SensorDCC. As an example, using TelosB [21] devices (sensor network enabled device), a specific SensorDCC implementation must be built to access it. Different drivers must be implemented for distinct platforms, resulting in different SensorDCCs. Communication between heterogeneous sensors pass through the DCC infrastructure, while uniform sensors can use their own protocols.

### 3.2.1 Encapsulating a Single Sensor and a Network

When a sensor is encapsulated within a DCC, its features are exposed through the uniform interface provided by the DCC. The major advantage of this approach is the interoperability it provides when developing new applications for a set of sensors. Moreover, if the sensors are changed, the applications do not need to be recoded.

If one single sensor is encapsulated, we call it a *SingleSensorDCC*. If a sensor network is encapsulated in one DCC, we have a *SensorNetworkDCC*. Both are SensorDCCs. A SensorNetworkDCC is responsible for all the sensors it encapsulates. Any message sent to this DCC is relayed to all sensors it encapsulates, and it controls the forwarding of the data generated by its sensors. The sensors are not aware of the existence of the DCCs, thus suffering no interference in their functioning.

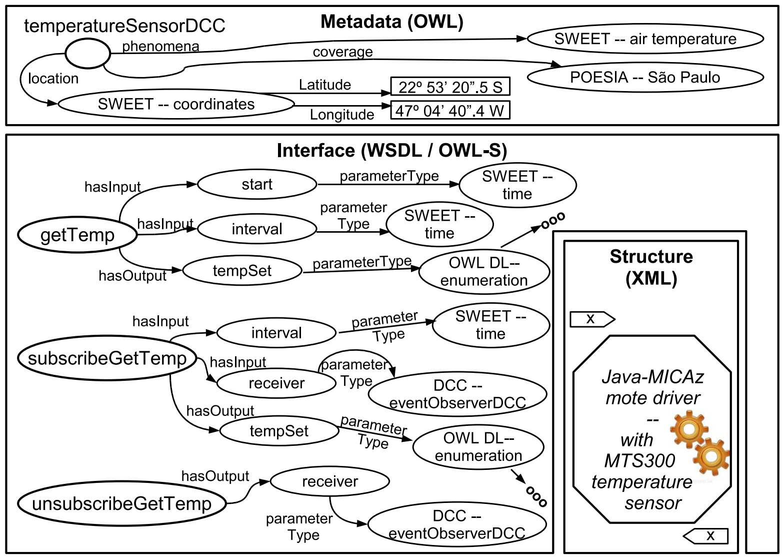


Figure 11: A schematic view of a SingleSensorDCC

Figure 11 shows an example of a SingleSensorDCC, with details omitted. The structure section describes the organization of a software module that communicates with the sensor to access its data, i.e., it is the driver that establishes the communication between the SensorDCC and the sensor. In the example, the driver implements a Java communication interface with a MICAz [20] mote coupled with a temperature sensor. The first operation (`getTemp`) receives a time interval in which the readings (floating-point numbers representing temperature in the Celsius scale) will be returned to the caller. The second operation (`subscribeGetTemp`) receives the frequency in which it should pack and send the polled sensor data, until the (`unsubscribeGetTemp`) operation is invoked. The metadata section describes the SensorDCC: `sensorType` indicates the type of sensing device that is encapsulated within the DCC, in this case a TemperatureSensorDCC; `phenomena` indicates which kind of measure the produced data represents; `coverage` shows in which region the sensor is acting; finally, `location` specifies the exact point where the sensor is located.

Figure 12 shows an example of a SensorNetworkDCC that assembles data from multiple sensors in a network. The schematics are similar to the SingleSensorDCC, only adapting the structure part to take care of multiple sensors. An external request sent to the DCC (e.g., `getTemp`) is translated into a request that is retransmitted to the sensors (through the wireless network) by the wireless driver on the server machine. The query can be answered by every sensor individually or with data condensed within the network. The results make the inverse flow path. The DCC can also play the role of data receiver, processing and encapsulating the data, regardless of the existence of any request (see stream encapsulation mechanisms in section 3.1).

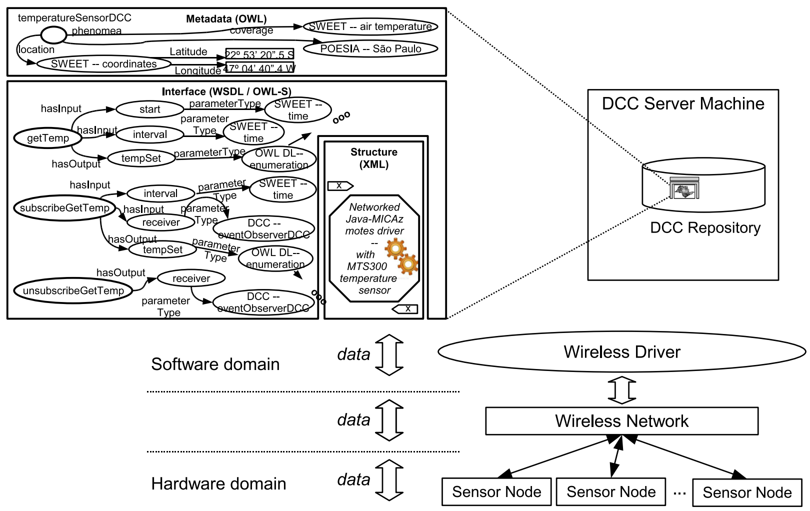


Figure 12: A sensor network encapsulation on a DCC

A SensorNetworkDCC can exert its control over the produced data in several ways, ranging from simple data forwarding to sophisticated fusion schemes. It is particularly valuable in networks that do not univocally identify each sensor, or in situations where it is not interesting or feasible to control each sensor node individually. In this case, an entire network can be represented as one DCC.

A sensor network can actually be encapsulated through its access point or base station. A DCC that encapsulates a base station (*BaseStationDCC*) creates an interface to the entire network. Through these interfaces, the applications can query the sensor network.

Figure 13 depicts these kinds of DCCs. **H**, for instance, is a SensorNetworkDCC, while **I**, **J** and **K** are SingleSensorDCCs. **W** is a workflow coordinating data publication and access. To **W**, **H** is a single source of sensing data. It can even be aware that the source is composed by many sensors, but does not need to access each one individually.

### 3.2.2 Group Formation

Encapsulated sensors create the possibility of *group formation*, bringing together sets of DCCs. A *GroupDCC* encapsulates any number of SensorDCCs. It is a specialization of SensorDCC, and therefore also of a ProcessDCC. Both SingleSensorDCCs and SensorNetworkDCCs can be part of a group. A group denotes a collection of (possibly heterogeneous) sensing nodes, encapsulated within DCCs, brought together based on some attribute or rule, such as: geographic position, functionality or feature, available batteries, energy efficiency, management capabilities, etc. A given set of SensorDCCs can be grouped in several different ways at the same time, each being represented by a GroupDCC, which can be regarded as special kind of view mechanism to sensor produced data.

There are two main advantages of using groups: simplifying the creation of general-purpose management solutions and exposing heterogeneous data sources homogeneously. In more detail, solutions for managing sensor produced data can be created based on DCC

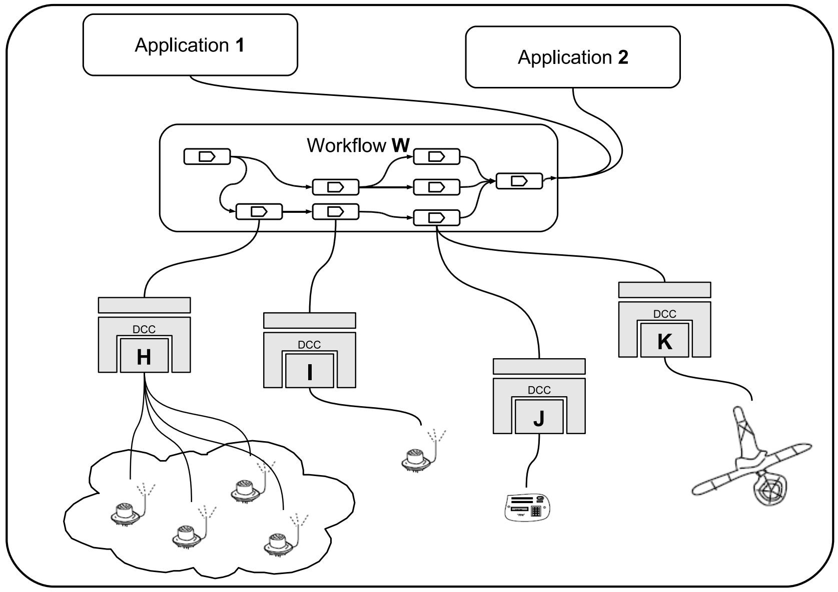


Figure 13: SensorNetworkDCCs (**H**) for sensor coordination schemes

interfaces, instead of having to rely on specific protocols implementations. Thus, the same solution can be reused in different deployed environments. The second aspect makes it possible to bring together heterogeneous data sources, exposing them in a single way, while, for instance, big sensing devices and sensor networks are the actual source.

Since the DCC model supports recursive definitions of collections of DCCs, a group can also be put in another group. Hence, a group can be seen as several sensing devices in one level of abstraction and as a single member of a group at another level. This offers the possibility of propagating reconfiguration changes, updating group formation, processing approaches and so on. This requires relatively little extra DCC design effort, provided that mechanisms are created to propagate changes to sensors within a group.

Group formation can also be done according to specific sensor network protocols, such as clustering or partitioning algorithms. This opens many possibilities to be explored. First, groups can be formed in a manual or automated fashion. Manual group formation requires that the user specifies each SensorDCC that is going to take part in the group. Automatic group formation can be based directly on the SensorDCC's semantic annotations or in rules to be validated based on both annotations and dynamic characteristics such as produced data. Another aspect has to do with group-specific operations, i.e., operations that are made available in a GroupDCC's interface and can only be executed on groups. Examples of these operations include inclusion and exclusion of SensorDCCs, and their regrouping or subgrouping. There can also be control groups, i.e., groups specially created to inspect data generation, assessing their accuracy based on specific indications – e.g., energy dispersion. In the crop monitoring example, for instance, experts can use group exclusion to ignore readings from faulty sensors in a group.

### 3.3 Providing Management Capabilities

Up to now, all examples in the report have involved accessing sensor data. However, DCCs can also support management capabilities – e.g., setting and reconfiguring data generation parameters.

Each SensorDCC can offer individual management methods in its interface. However, controlling each SensorDCC on an individual basis may not be feasible. Higher level management layers can be created in order to further facilitate the management of data production and annotation. We thus introduce a specialized DCC, called *ManagementDCC*, which aggregates operations whose implementation is based on the individual management operations of each SensorDCC.

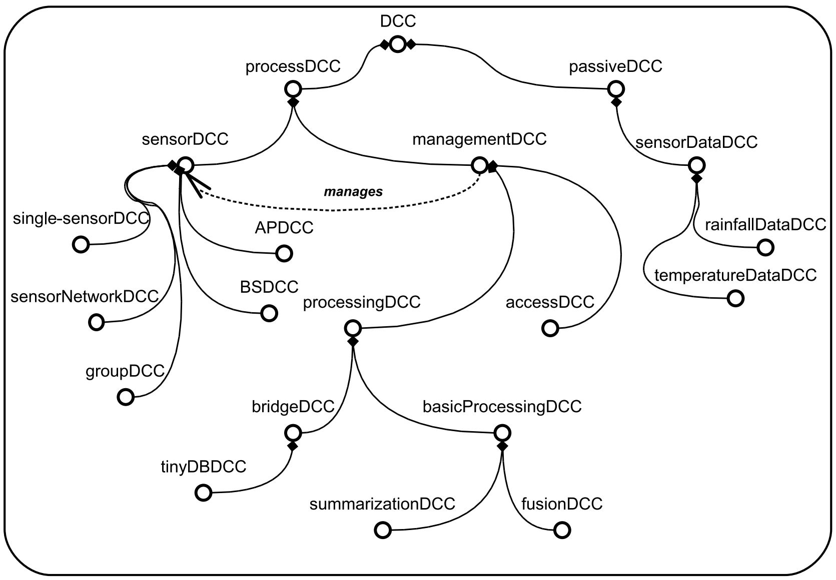


Figure 14: SensorDCC taxonomy, extension of the DCC taxonomy [78]

Figure 14 shows an extension of the DCC taxonomy proposed by [78–80] in order to cover the new DCC types introduced in this work. The diamond ended lines represent the subclass relationship, i.e., a SensorNetworkDCC *is a* (or is a subclass of) SensorDCC, in the figure. The SensorDCC is the basic DCC to serve as a proxy for the sensing and the auxiliary devices. The ManagementDCC is the generic component that can execute management tasks over the SensorDCCs (represented in the figure by the relationship *manages*). Here we consider two specializations of ManagementDCC: (i) ProcessingDCC (section 3.3.1), which encapsulates basic data manipulation capabilities; (ii) AccessDCC (section 3.3.2), which actually makes the data available to the applications.

The task of aggregating individual management functions can be developed by programming a new DCC but also by specifying a workflow. This workflow may use the individual operations as activities and the logic behind the management is created as the flow is specified. The approach has the advantage of being able to use the workflow design environment to tailor management solutions. This represents the second part of our contribution (section

4).

### 3.3.1 Encapsulating Data Processing

The processing of sensor data requires several specialized functions – e.g., summarization. We propose to encapsulate such functionalities within *ProcessingDCCs*, a specialization of ManagementDCC. In Figure 10, the ProcessingDCCs are located in layer 3 (centralization). There are two kinds of ProcessingDCCs: *BasicProcessingDCCs*, which are implemented to directly compute functions on sensor-produced data, and *BridgeDCCs*, which serve as a connection point to sensor middlewares.

Functionalities that can be implemented in BasicProcessingDCCs include data filtering, clustering and classification, application of association rules and regression techniques, multi-source data fusion, data summarization and sampling, among others. These DCCs, can be combined to obtain more complex processes.

BridgeDCC exist in order to take advantage of the many solutions already implemented in other frameworks. Consider, for instance, TinyDB [50], a popular middleware solution for accessing sensor data. Its SQL-like query interface can be offered by operations on a BridgeDCC interface. Here, instead of having to become familiar with TinyDB, the application sends a request to the corresponding BridgeDCC, which translates it into the appropriate syntax, forwards the request and returns the result. The query proxy system from Cougar [110], or application adaptation update mechanisms from Impala [48], or Maté's [46] reprogramation techniques can also be made available through a BridgeDCC's high level interface. The same interface specification can thus serve to access these distinct systems.

### 3.3.2 Offering Data Access

The *AccessDCC* is a ManagementDCC that offers operations for the higher level data access (application access). Going back to Figure 10, AccessDCCs are in layer 4, publication. These operations reflect all the features offered by DCCs in lower layers. Examples are: if a BridgeDCC that implements access to the TinyDB middleware is available, an AccessDCC operation could receive as a parameter a TinyDB's SQL-like query; if a SensorDCC that offers access to reprogramming sensor nodes is available, a parameter could be the compiled software to reprogram the node; if a classification ProcessingDCC is available, a parameter of an operation could be the maximum number of categories desired, or the characterization of the known categories.

Another feature to be implemented by AccessDCCs is access control. A useful approach here is to create one GroupDCC per specific policy. This way, AccessDCC can control which applications have access to which SensorDCCs and their features.

# 4 Managing Components using Workflows

The second aspect of our proposal is to use workflow-based techniques to manage the data produced by sensors. Workflows represent a flexible way of coordinating the management of sensing data.

The idea is to take advantage of DCC interfaces and composition mechanisms. Thus, a solution constructed for one scenario or one set of sensors can be easily reused in different environments. Applications can select or post adequate workflow specifications in order to obtain the desired data. Since there are several systems and languages to specify and compose workflows, this solution adds usability and flexibility for sensor data management.

## 4.1 Management Workflows

The goal of a *management workflow* is to manage the sensor data in order to support complex application requests. More specifically, it controls ManagementDCCs and SensorDCCs coordinating the tasks of data extraction, processing and interpretation. Furthermore, a management workflow can access SensorDataDCCs as additional data sources. The latter can provide the needed parameters or even operations, using the companion mechanism.

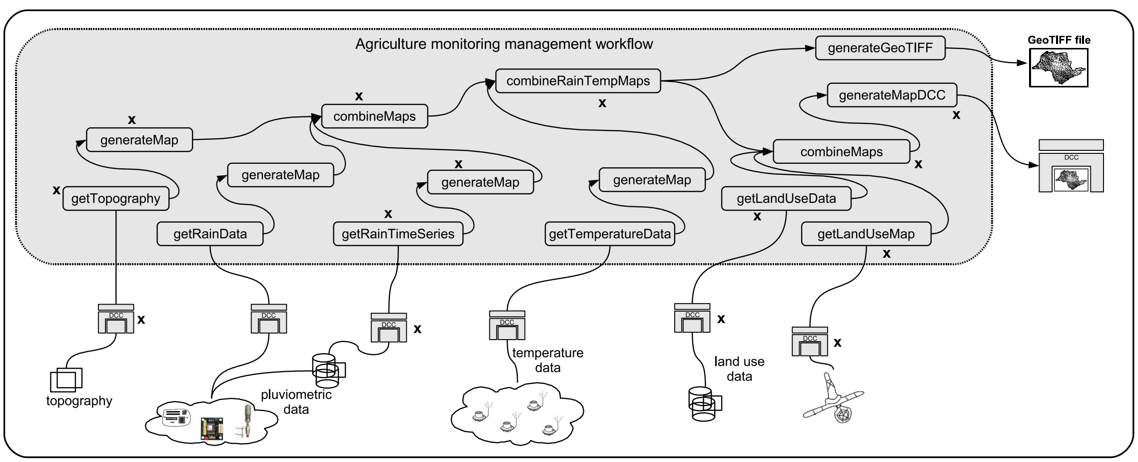


Figure 15: An example of a management workflow.

Figure 15 shows an example of a management workflow for agricultural monitoring (stage (ii) of our example). It is used to acquire data from distinct kinds of sensors and sensor-derived data files. It periodically produces derived data, here a file of a map in GeoTIFF. For instance, activity `getRainData` accesses a SensorDCC for real time measures, while `getRainTimeSeries` accesses a SensorDataDCC for historic rainfall data. Both generate maps that are combined to detect rainfall evolution patterns, taking into account a region's topography (map generated from topographic data).

Workflow activities access DCCs for data input, and can generate other DCCs. The main mechanism to determine which DCCs can be used for a given activity is based on a combination of ontology annotations and type matching (of DCC operation interfaces and metadata, and activity specification). This is supported by the DCC management infrastructure – see section 5. In the Figure 15, activities and DCCs marked with `x` delimit the subworkflow and the SensorDataDCC used for crop planning (stage (i) of our running example). We point out a few aspects that characterize our solution. First, crop monitoring can be configured as a workflow that *reuses* part of the planning workflow. Second, the monitoring workflow is only concerned with sending requests to DCCs, *regardless of the nature* of the data sources. Finally, the workflow *manages and publishes* sensor data, and publication results can themselves be encapsulated in a DCC.

Our workflows are specified favoring our reuse methodology [54]. The first step consists in the definition of the data and activity types. Next, activities are created from types: after choosing an activity type, one can choose the data that is going to be used by that activity, based on the data types of the activity's parameters. Activities can be created only at an abstract specification level – thus defining an *abstract workflow*. This workflow can be then customized to specific situations, and subsequently instantiated for execution. Management workflows can be reused and adapted to solve new data needs. To make a workflow executable, pieces of software must be associated to each of its activities. This is where the DCCs come into action. Workflow execution is carried out transparently by a WFMS, which uses DCC operations, coordinating the data flow.

A workflow can be used as a basic management unit. Examples of these units are: a process that issues a warning when a certain threshold is reached; a process to detect outliers in a data set within a time frame; a process that monitors data generation intervals, raising flags indicating correct, incorrect or interrupted data generation. From these basic units, more complex processes can be build.

### 4.2 Validation Workflows

We can use several validation techniques to make sure that data follow given quality or validity parameters. This evaluation can be specified using the workflow and DCC based infrastructure. An interesting aspect of these mechanisms is the possibility of offering data quality assurance to the applications.

We propose five methods for data validation, which can be used individually or in a sequential combination inside a workflow. The methods are: sampling, summarization, pre-processing, and pre- and post-condition verification. Sampling, summarization and pre-processing are basic features frequently used to extract information from data sets. In the validation context, the idea is to extract representative sets of values from sensor-produced data, perform additional processing and evaluate the results against some benchmark. Pre- and post-conditions consists in evaluating logical expressions using the sensing data as variables. This evaluation is to be performed right after the readings, before any modifications on the data (pre-conditions), or after all modifications on the data (post-conditions), right before its publication to the applications.

All the validation schemes can be stored and published along with the data, so that data production can be traced back to its source, so that all the manipulations are available to be analyzed. Pre- and post-conditions can be stored as text. Pre-processing, summarization and sampling can be stored by references to the respective DCCs used (including DCCs that encapsulate workflows).

Validation can be also applied at query time, producing validated data in an on-demand fashion. This is specially useful when the answers to queries are streams of data. The mechanisms are the same for storing data, including the conditions and the DCCs.

### 4.3 Publication Workflows

Publication is the final aspect of data management in our work. Management workflows are geared towards supplying data for specific application needs (e.g., generating erosion maps or controlling the temperature in a factory environment). Publication workflows, on the other hand, are general purpose data providers.

Functionalities to publish sensor data include: (i) data fusion schemes, even exerting device reconfiguration control; (ii) data summarization and sampling, with configuration parameters; (iii) application of statistical analysis over the readings; (iv) data classification, filtering, clustering and many other mining related techniques. These functionalities are accessed and composed into a publication workflow via management operations offered by ManagementDCCs and individual operations on SensorDCCs;

# 5 Implementation Issues

This section presents implementation details and current implementation stage.

## 5.1 General Architecture

Figure 16 illustrates the main modules of the architecture. Scientists (the main users) interact with it via the GUI (Graphical User Interface) to design workflows and DCCs, and monitor workflow and/or DCC execution. The architecture relies on the following subsystems: (i) WOODSS [54] – a scientific workflow specification and documentation environment developed at the Laboratory of Information Systems, UNICAMP; (ii) ANIMA [80], an infrastructure developed to support DCC execution and management; (iii) a set of modules to design DCCs, eventually reusing and modifying DCCs already stored in a repository [79]; and (iv) a basic infrastructure needed to monitor and execute workflows, including the WFMS.

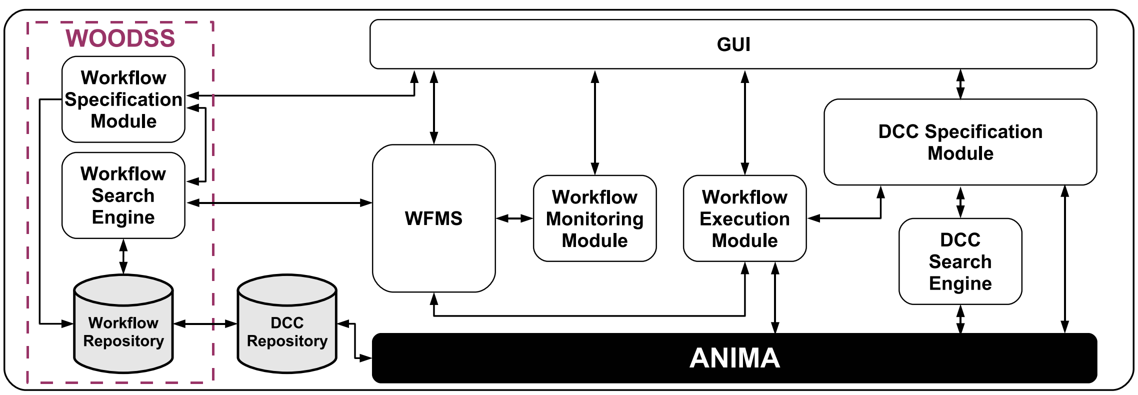


Figure 16: The main modules of the architecture.

In more detail, WOODSS allows users to specify scientific workflows, using our methodology, and to annotate both workflows and data manipulated by them. Workflow specifications (at all abstraction levels), their components, and annotations are stored in the Workflow Repository. Scientists can construct a workflow from scratch, or reuse and adapt stored workflows, which are retrieved with help from the Workflow Search Engine, based on workflow metadata – e.g., are there any workflows that create erosion maps? Repository records point to external elements they invoke (e.g., a Web Service) or manipulate (e.g., data sets). For more details, the reader is referred to [54].

DCC specification and management are handled by the modules on the right of the figure. Users interact with the GUI either to specify a new DCC from scratch, which is subsequently stored in the DCC Repository, or to construct a DCC reusing and adapting existing stored components. The goal of the DCC Search Engine is to help DCC design and composition, supporting the user in finding the most appropriate DCC to reuse for a given application purpose – e.g., is there any SensorDataDCC that encapsulates rainfall time series for São Paulo State? This engine's search mechanisms are based on a set of algorithms that combine type annotations, metadata and interface matching [78]. For more details, see [79].

To select a workflow for execution, a user (or an application) sends a request to the WFMS, which invokes the Workflow Search Engine to retrieve the desired workflow from the Workflow Repository. All data and DCCs needed to execute the workflow are all retrieved in this search step.

Subsystems (i) through (iii) are already implemented, and we do not intend to implement a workflow engine. Rather, we will use some available system. Meanwhile, subsystem (iv) has been replaced by a simple coordination mechanism, extracted from Anima's synchronization modules. This allows simulating reasonably complex workflows, including those with parallel branches, synchronization and loops. The GUI has distinct modules that support graphical workflow design, DCC specification and construction of composite DCCs [80]. Present implementation efforts are concerned with two issues: first, we are concentrating on the development of more DCCs for sensors. Next, we will support internal interactions among WOODSS, WFMS and DCC modules for their integrated execution.

Indeed, since WOODSS and the DCC management and execution system were developed independently, their integration is not automatic – i.e., in order to construct a workflow that accesses SensorDCC and SensorDataDCC, the user first has to design these DCCs using the DCC Specification Module on the right, and then proceed to workflow specification using WOODSS. Present integration occurs only at the repository level (the Workflow Repository points at DCCs stored in the DCC Repository). Moreover, the DCC Search Engine is not yet integrated into the system. At present, in order to find appropriate DCCs to be invoked by a workflow, the scientist has to directly interact with the DCC Search Engine via the GUI, providing the desired parameters, and choose a DCC from the list returned by the engine.

## 5.2 Implementing DCCs for Sensors

This section gives an overview of DCC implementation aspects. The presentation concentrates on specific components, thereby exemplifying some of the problems encountered. Code was implemented in Java. Annotations follow the OWL vocabulary and refer mainly to three ontologies: NASA's SWEET [72], POESIA [30] and the DCC ontology [77–79].

### 5.2.1 SensorDataDCCs

This section describes three of the implemented SensorDataDCCs, namely, *TemperatureSensorDataDCC*, *RainfallMapSensorDataDCC* and *RainfallMapSetSensorDataDCC*.

The TemperatureSensorDataDCC encapsulates a temperature time series stored in a plain text file, whose records are pairs <temperature, timestamp> – both measures as defined by SWEET – captured by a sensor in a given geographic location (GPS coordinates). There is one TemperatureSensorDataDCC per sensor/location. Operations available include retrieval of one pair or a sequence thereof, and some summarization computations (e.g., average). For instance, `getTemp(Date begin, Date end)` returns the set of all temperature readings within the time frame from *begin* to *end*. DCC metadata contain location coordinates, the measurement unit (e.g., Celsius degrees), and information on originating sensor (e.g., type of sensor that captured the series).

The RainfallMapSensorDataDCC encapsulates one GeoTIFF file where each pixel corresponds to a given geographical coordinate and contains one rainfall measure for a particular month of a particular year. Its operations concern: getting the entire GeoTIFF file, getting values from individual pixels (given either geographical coordinates or pixel relative position in the image). Finally, the RainfallMapSetSensorDataDCC contains a set of GeoTIFF image files with rainfall measures. Operations can request data from one image in the set, or from a subset of the images (average or accumulated values). Metadata are of a similar nature to those provided for the temperature sensor. Figure 5, which also encapsulates a set of GeoTIFF image files, exemplifies some of the operations of this DCC.

We recall that SensorDataDCCs are passive components; thus, the implementation of these DCCs had to be followed by the implementation of its companion, which contains the code of all interface operations (see section 2.3). The main challenges faced here were determining the appropriate operations and their parameters, and the actual implementation of the operations. Moreover, these files are provided by a variety of institutions, and thus metadata values are not always available. However, since PassiveDCC contain only static data, we did not need to worry about sensor behavior. This problem appeared in the implementation of SensorDCCs.

### 5.2.2 SensorDCCs

We implemented the communication driver using the TinyOS [47, 88] interfaces. The sensors were programmed with TinyOS and software developed in the NesC [31] language, which is a C-derived component oriented programming language.

We implemented SensorDCCs for the MICAz mote [20] (MicazSensorDCC) and for the TelosB mote [21] (TelosbSensorDCC). These SensorDCCs have similar interfaces and were tested with temperature and light readings. The MICAz mote requires an auxiliary MTS300 sensorboard, whereas TelosB came with integrated sensors. The MicazSensorDCC and the TelosbSensorDCC are generating real-time data and making them available in four ways (operations implemented): (i) on-demand access (application receives the data as needed in real time); (ii) generating text file data outputs; (iii) generating new SensorDataDCCs (TemperatureSensorDataDCCs); and (iv) updating existing SensorDataDCCs.

Implementation was divided in two parts – creating communication drivers to access data through sensor-specific protocols, and creating the DCCs themselves. The first part required familiarity with the characteristics of each sensor, e.g., to interpret the data packets emitted. This kind of effort is needed for every new kind of sensor device encountered – not only in our approach, but for any development environment that wishes to support access to sensor data. The difference to other solutions appears in the second part. Whereas they require extensive communication-oriented coding to forward data delivered by the drivers, DCCs directly map driver operations to interface operations. Thus, coding effort is much smaller – indeed, the DCC approach not only supports homogeneous access from external applications, but also simplifies a programmer's work. Nevertheless, there were also challenges in developing these DCCs, mostly associated with semantic annotations (e.g., finding appropriate ontology terms).

### 5.3 Examples

This section presents two simple examples of the result of running our implementation with real data and sensors. Consider an application whose goal is to compare rainfall data from two different periods. In agriculture, one way to achieve this is to get maps for each period and subtract them. The result is also a map which must be interpreted by experts.

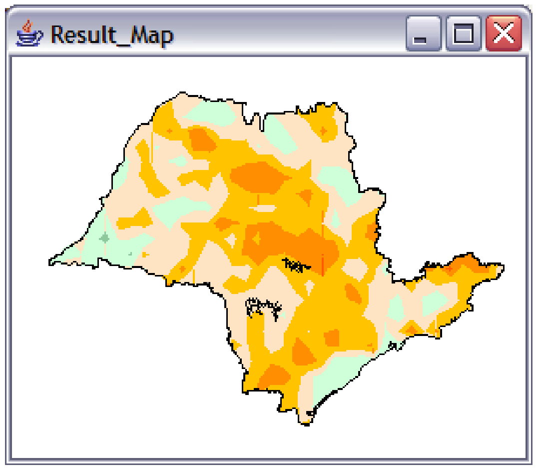


Figure 17: Screen capture of a combined rainfall map.

Figure 17 shows a screen capture of this application. This screen copy was generated by a workflow with three activities: two invocations of the GetMap operation on a RainFallMapSetSensorDataDCC containing data on São Paulo State (the DCC that encapsulates a set of rainfall maps), followed by a map subtraction operation implemented directly without accessing a DCC. This is typical in a planning scenario.

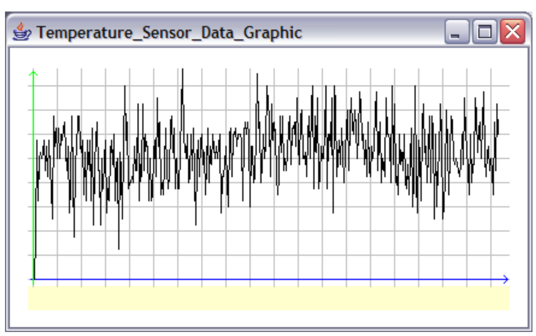


Figure 18: Screen capture of a temperature graphic.

Figure 18 shows the result of an application that contains a loop that polls a MicazSensorDCC, and plots a real time curve to be interpreted by experts. This is typical in a monitoring scenario.

# 6 Alternative Solutions

We have commented on related work throughout the report. There remains to compare our proposal with alternative approaches, in particular regarding two aspects. One of the aspects concerns the (general-purpose) data management solutions, for which we propose solving through scientific workflows. Another issue we emphasize is homogeneous access to resources, which we achieve by encapsulating the sensors within DCCs.

Scientific workflows are extensively used in eScience, e.g., orchestrating Web services, or specifying execution of experiments in a grid environment [112]. To the best of our knowledge, however, no proposals exist to use such workflows to manage access to heterogeneous sensing devices.

From the point of view of management solutions, alternatives include: (i) Specialized implementation, e.g., an entire software system for one specific application, with the same drawbacks of the specialized implementation for homogeneous data access; (ii) Other composition techniques, such as a publish-subscribe scheme [25, 103], which are not suited for process oriented executions.

From the point-of-view of homogeneously accessing the data, one approach alternative to ours is to directly use Web services [108] to publish data, and to have access to sensors. However, this has two drawbacks: (1) unlike PassiveDCCs, Web services do not actually encapsulate data, and thus are always associated with some specific implementation; (2) a Web environment is mandatory for Web services, while DCCs can also be used in a standard programming environment, regardless of the Web (and its overhead).

Other accessing solutions that could be considered are: (i) Specialized (and language specific) implementations; (ii) Software components and communication middlewares, such as CORBA [62], COM (DCOM and COM+) and .NET [56], EJB [86], and others; (iii) WSN middleware as defined by [37]. The first approach has the classic overhead of unnecessary repetition of work, hard maintenance, lack of standardization and interoperability. Finally, components and general middleware lack flexibility, semantic descriptions, and, more importantly, homogeneous treatment of data, devices and software.

Exploring further the WSN middlewares, four approaches are close to ours. *Global Sensor Network* (GSN) [2, 41], has similar goals, however, data management is restricted to homogeneously accessing the network using a declarative language, while our proposal considers including software and combination of software modules with workflow-based coordination. The *Sensor Network Services Platform* (SNSP) [82] proposal considers the possibility of including processing software through the concept of "auxiliary service". However, it is centered around a formal specification of levels and services, leaving aside implementation issues, both for data publishers and data consumers. *Hourglass* [40] concisely considers processing solutions, but requires the use of a specific definition language and uses low-level TCP socket communication schemes, while DCCs use Semantic Web standards for both specification and communication. *IrisNet* [32] limits the use of sensor data to a hierarchical XML database, using XPath queries. DCC interfaces, on the other hand, can support a wide range of access mechanisms.

Other proposals act in more specialized branches. *TinyDB* [50] provides access to an entire network in a single entry point, which uses an SQL-like query system. It also aggregates data readings, decreasing the transmission costs, but at the cost of maintaining information on the network structure, compromising scalability. *Cougar* [110] works well on large sensor sets and makes data access easy with its query system; however, its focus is centered in efficient sensor programming, while our proposal is concerned with the management of data from the sensor outwards. *Impala* [48] is limited to a specific handheld hardware, but supports protocol and operation mode updates. *Maté* [46] addresses issues such as protocol updates and node heterogeneity (using a virtual machine approach), but lacks effective and easy communication with applications. *Magnet* [10] delivers a Java virtual machine on top of the network, facilitating the development of Java applications, but is unfit for nodes with limited capacity nodes. All these solutions can be encapsulated in our BridgeDCC.

# 7 Concluding Remarks

This report presented a framework to support the flexible management and publication of sensor-produced data. Part of this framework has been implemented and validated. It is based on two main levels: the use of DCCs to provide uniform access to sensor data, sensing devices and software to process the data; and the construction of scientific workflows to coordinate sensor data integration and publication.

At the bottom level, DCC interfaces and composition mechanisms offer a homogeneous view of data and sensing devices – applications and workflows do not need to directly concern themselves with whether they are handling static files, or real time data streams. Workflows provide customizable building blocks to access and invoke DCC operations. Sensor data management can thus be translated into workflow specification and execution, a task closer to the procedures of scientific communities that need to process sensor data. Moreover, thanks to the workflow repository, workflow pieces can be reused to support new kinds of data publication or integration needs – again facilitating usability. From an external viewpoint, applications just need to invoke appropriate workflows or workflow tasks to access the desired data, simplifying the problem of construction of special-purpose software for each new application need.

Ongoing work involves several issues. We are implementing the automatic translation of in-memory workflows to a publishable WS-BPEL specification. As mentioned in section 5, we are constructing more SensorDCCs. We are also working on the actual construction of BridgeDCCs and GroupDCCs, where several groups coexist and interact on top of the same sensor network. Finally, we will replace our synchronization-based execution module by a full-fledged workflow engine.

Another issue is to encapsulate workflows themselves in DCCs. This simplifies our framework from a conceptual point of view, since it can then be seen as a DCC-only solution – each management layer of figure 10 will only contain DCCs. On the other hand, this brings the added complexity of construction of ProcessDCCs to encapsulate DCC workflows.

The basic idea is to encapsulate all workflow building blocks within DCCs – thus, the construction of workflows is transformed into the problem of DCC composition. The same composition process applies here – workflow activities can be invoked as DCC operations, and they can harvest and process sensor data by coupling themselves to the appropriate SensorDCCs. Thus, the entire problem can be handled by using DCC composition mechanisms: for construction of workflows, and for having workflows access SensorDCCs.

## Acknowledgements

This work is supported by FAPESP (grant 2004/14052-3) and partially financed by a CNPq grant, the WebMaps II project and an HP Digital Publishing grant.

## References


[1] D. J. Abadi, S. Madden, and W. Lindner. REED: Robust, Efficient Filtering and Event Detection in Sensor Networks. In *Proceedings of the 31st VLDB Conference*, 2005.

[2] K. Aberer, M. Hauswirth, and A. Salehi. A Middleware for Fast and Flexible Sensor Network Deployment. In *Proceedings of the 32nd VLDB Conference (Demo Session)*, 2006.

[3] A. Ailamaki, C. Faloutsos, P. S. Fischbeck, M. J. Small, and J. VanBriesen. An environmental sensor network to determine drinking water quality and security. *SIGMOD Record*, 32(4):47–52, 2003.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[5] J. N. Al-Karaki and A. E. Kamal. A Taxonomy of Routing Techniques in Wireless Sensor Networks. In *Handobook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, pages 116–139. CRC Press, 2004.

[6] R. Allen. *Workflow Handbook 2001*, chapter Workflow: An Introduction. Workflow Management Coalition (WfMC), 2001. `http://www.wfmc.org/information/Workflow-An_Introduction.pdf` (as of November 2006).

[7] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[8] E. Armstrong, S. Bodoff, D. Carson, M. Fisher, D. Green, and K. Haase. *The Java Web Services Tutorial)*, chapter 1 – Introdution to Web Services. Addison Wesley, 2002. `http://java.sun.com/webservices/docs/1.0/tutorial` (as of May 2006).

[9] H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, E. F. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. B. Zdonik. Retrospective on Aurora. *VLDB Journal*, 13(4):370–383, 2004.

[10] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. G. Sirer. On the need for system-level support for ad hoc and sensor networks. *ACM SIGOPS Operating Systems Review*, 36(2):1–5, 2002.

[11] T. Berners-Lee. RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntax, August 1998. `http://www.ietf.org/rfc/rfc2396.txt`, (as of November 2006).

[12] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, May 2001.

[14] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping Wireless Sensor Network Applications with BTnodes. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 323–338, 2004.

[15] P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *Proc. of the 2nd Int. Conference on Mobile Data Management*, 2001.

[16] BPMI.org. Business Process Management Initiative. `http://www.bpmi.org` (as of November 2006).

[17] BPMI.org. Business Process Modeling Language 1.0. `http://www.bpmi.org/bpml-spec.htm` (as of Novemeber 2006).

[18] M. C. R. Cavalcanti, M. L. Q. Mattoso, and M. L. M. Campos. *Scientific Resources Management: Towards An In Silico Laboratory.* PhD thesis, Coppe – UFRJ, Rio de Janeiro–RJ, 2003.

[19] M. C. R. Cavalcanti, M. L. Q. Mattoso, M. L. M. Campos, F. Llirbat, and E. Simon. Sharing scientific models in environmental applications. In *Proc. of ACM Symposium on Applied Computing*, 2002.

[20] Crossbow Technology Inc. MicaZ Mote. `http://www.xbow.com/Products/productsdetails.aspx?sid=101` (as of November 2006).

[21] Crossbow Technology Inc. TelosB Mote. `http://www.xbow.com/Products/productsdetails.aspx?sid=126` (as of November 2006).

[22] D. Culler, D. Estrin, and M. Srivastava. Overview of Sensor Networks. *Computer*, 37(8):41–49, 2004.

[23] N. Cullot, C. Parent, S. Spaccapietra, and C. Vangenot. Ontologies: A contribution to the DL/DB debate. In *Proceedings of the 1st International Workshop on the Semantic Web and Databases, 29th International Conf. on Very Large Data Bases*, pages 109–129, 2003.

[24] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-based approximate querying in sensor networks. *VLDB Journal*, 14(4):417–443, 2005.

[25] P. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[26] European Semantic Systems Initiative (ESSI Cluster). Web Service Modeling Language (WSML). `http://www.wsmo.org/wsml/` (as of Novemeber 2006).

[27] European Semantic Systems Initiative (ESSI Cluster). Web Service Modeling Ontology (WSMO). `http://www.wsmo.org/` (as of Novemeber 2006).

[28] European Semantic Systems Initiative (ESSI Cluster). Web Service Modelling eXecution environment (WSMX) – WSMO reference implementation. `http://www.wsmx.org/` (as of Novemeber 2006).

[29] R. Fileto. *The POESIA Approach for the Integration of Data and Services in the Semantic Web*. PhD thesis, IC–UNICAMP, Campinas–SP, Brazil, 2003.

[30] R. Fileto, L. Liu, C. Pu, E. D. Assad, and C. B. Medeiros. POESIA: An Ontological Workflow Approach for Composing Web Services in Agriculture. *The VLDB Journal*, 12(4):352–367, 2003.

[31] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003.

[32] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), 2003.

[33] J. I. Goodman, A. I. Reuther, and D. R. Matinez. *Handobook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter Next Generation Technologies to Enable Sensor Networks. CRC Press, 2004.

[34] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[35] M. Gruninger and J. Lee. Special issue on ontologies applications and design, 2002.

[36] N. Guarino. Formal ontology and information systems. In *Proceedings of the Internatinal Confonference on Formal Ontologies in Information Systems (FOIS)*, pages 3–15, 1998.

[37] S. Hadim and N. Mohamed. Middleware Challenges and Approaches for Wireless Sensor Networks. *IEEE Distributed Systems Online*, 7(3), 2006. art. no. 0603-o3001.

[38] D. L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, January 1997.

[39] J. M. Hellerstein, W. Hong, and S. Madden. The Sensor Spectrum: Technology, Trends and Requirements. *SIGMOD Record*, 32(4):22–27, 2003.

[40] J. Shneidman and P. Pietzuch and J. Ledlie and M. Roussopoulos and M. Seltzer and M. Welsh. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Technical report, Harvard University, 2004. TR-21-04.

[41] K. Aberer and M. Hauswirth and A. Salehi. The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical report, Ecole Polytechnique Fédérale de Lausanne, 2006. LSIR-REPORT-2006-006.

[42] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Mobile Networking for “Smart Dust”. In *Int. Conf. on Mobile Computing and Networking*, 1999.

[43] V. Kashyap and A. Sheth. Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies. In M. P. Papazoglou and G. Schlageter, editors, *Cooperative Information Systems*, pages 139–178. Academic Press, 1998.

[44] D. Kaster, C. B. Medeiros, and H. Rocha. Supporting Modeling and Problem Solving from Precedent Experiences: The Role of Workflows and Case-Based Reasoning. *Environmental Modeling and Software*, 20(6):689–704, 2005.

[45] I. Lazaridis, Q. Han, X. Yu, and S. Mehrotra. QUASAR: Quality Aware Sensing Architecture. *SIGMOD Record*, 33(1):26–31, 2004.

[46] P. Levis and D. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. In *Proc. 10th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, pages 85–95, 2002.

[47] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *Ambient Intelligence*, chapter TinyOS: An Operating System for Wireless Sensor Networks. Springer Verlag, 2004.

[48] T. Liu and M. Martonosi. Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. In *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP 03)*, pages 107–118, 2003.

[49] H. Luo, J. Luo, and Y. Liu. Energy efficient routing with adaptive data fusion in sensor networks. In *DIALM-POMC '05: Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 80–88, 2005.

[50] S. R. Madden, M. J. Franklin, and J. M. Hellerstein. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[51] M. Mani. Understanding the Semantics of Sensor Data. *SIGMOD Record*, 32(4):28–34, 2003.

[52] F. Manola and E. Miller. RDF Primer – W3C Recommendation 10 February 2004, 2004. `http://www.w3.org/TR/rdf-primer/` (as of November 2006)).

[53] K. Martinez, J. Hart, and R. Ong. Environmental Sensor Networks. *Computer*, 37(8):50–56, 2004.

[54] C. B. Medeiros, J. Perez-Alcazar, L. Digiampietri, G. Z. Pastorello Jr, A. Santanchè, R. S. Torres, E. Madeira, and E. Bacarin. WOODSS and the Web: Annotating and Reusing Scientific Workflows. *SIGMOD Record*, 34(3):18–23, 2005.

[55] D. Menasce and V. Almeida. *Capacity Planning for Web Services*. Prentice Hall, 2001.

[56] Microsoft. COM: Component Object Model Technologies. `http://www.microsoft.com/com/` (as of March 2007).

[57] R. Müller and G. Alonso. Efficient Sharing of Sensor Networks. In *Proceedings of the 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2006.

[58] E. F. Nakamura, F. G. Nakamura, C. M. S. Figueiredo, and A. A. F. Loureiro. Using Information Fusion to Assist Data Dissemination in Wireless Sensor Networks. *Telecommunication Systems*, 30(237-254):1–3, 2005.

[59] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[60] Oasis-Open. Organization for the Advancement of Structured Information Standards. `www.oasis-open.org` (as of November 2006).

[61] OASIS Web Services Business Process Execution Language Technical Committee. WSBPEL Specification. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel` (as of march 2007).

[62] Object Management Group (OMG). Common Object Request Broker Architecture (CORBA). `http://www.corba.org/` (as of March 2007).

[63] OWL-S (The DAML Program). OWL-based Web Service Ontology. `http://www.daml.org/services/owl-s/` (as of November 2006).

[64] S. Papavassiliou and J. Zhu. *Handobook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter Architecture and Modeling of Dynamic Wireless Sensor Networks, pages 297–312. CRC Press, 2004.

[65] G. Z. Pastorello Jr. Publication and Integration of Scientific Workflows on the Web. Master's thesis, IC–UNICAMP, Campinas–SP, Brazil, 2005. In Portuguese.

[66] G. Z. Pastorello Jr, C. B. Medeiros, S. M. Resende, and H. A. Rocha. Interoperability for GIS Document Management in Environmental Planning. *Journal on Data Semantics*, 3(LNCS 3534):100–124, 2005.

[67] C. Plesums. *Workflow Handbook 2002*, chapter An Introduction to Workflow. Workflow Management Coalition (WfMC), 2002. `http://www.wfmc.org/information/introduction_to_workflow02.pdf` (as of November 2006).

[68] PostgreSQL Global Development Group. PostgreSQL Object/Relational Database Management System. `http://www.postgresql.org/` (as of November 2006).

[69] G. Pottie and W. Kaiser. Wireless Integrated Netork Sensors (WINS): Principles and Approach. *Communications of the ACM*, 43(5):43–50, 2000.

[70] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2), 2002.

[71] U. Ramachandran, R. Kumar, M. Wolenetz, B. Cooper, B. Agarwalla, J. Shin, P. Hutto, and A. Paul. Dynamic Data Fusion for Future Sensor Networks. *ACM Transactions on Sensor Networks*, 2(3):404–443, August 2006.

[72] R. Raskin and M. Pan. Semantic Web for Earth and Environmental Terminology (SWEET). In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, 2003.

[73] L. B. Ruiz, T. R. M. Braga, F. A. Silva, H. P. Assuncao, J. M. S. Nogueira, and A. A. F. Loureiro. On the Design of a Self-Managed Wireless Sensor Network. *IEEE Communications Magazine*, 43(95-102):8, 2005.

[74] L. B. Ruiz, J. M. S. Nogueira, and A. A. F. Loureiro. MANNA: A Management Architecture for Wireless Sensor Networks. *IEEE Communications Magazine*, 41(2):116–125, 2003.

[75] L. B. Ruiz, J. M. S. Nogueira, and A. A. F. Loureiro. *Handobook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter Sensor Network Management. CRC Press, 2004.

[76] A. Santanchè. *Fluid Web and Digital Content Components: from a document-centric perspective to a content-centric perspective.* PhD thesis, IC–UNICAMP, Campinas–SP, Brazil, 2006.

[77] A. Santanchè and C. B. Medeiros. Managing Dynamic Repositories for Digital Content Components. In *Current Trends in Database Technology – EDBT 2004 Workshops*, volume LNCS 3268, pages 66–77, 2004.

[78] A. Santanchè and C. B. Medeiros. Self Describing Components: Searching for Digital Artifacts on the Web. In *Proc. of XX Brazilian Symposium on Databases*, pages 10–24, 2005.

[79] A. Santanchè and C. B. Medeiros. A Component Model and an Infrastructure for the Fluid Web. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):324–341, 2007.

[80] A. Santanchè, C. B. Medeiros, and G. Z. Pastorello Jr. User-centered Multimedia Building Blocks. *Multimedia Systems Journal*, 12(4):403–421, 2007.

[81] L. Seffino, C. B. Medeiros, J. Rocha, and B. Yi. WOODSS - A Spatial Decision Support System based on Workflows. *Decision Support Systems*, 27(1–2):125–123, 1999.

[82] M. Sgroi, A. Wolisz, A. Sangiovanni-Vincentelli, and J. M. Rabaey. *Ambient Intelligence*, chapter A service-based universal application interface for ad hoc wireless sensor and actuator networks. Springer Verlag, 2005.

[83] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB Journal*, 13(4):384–403, 2004.

[84] I. G. Siqueira, L. B. Ruiz, A. A. F. Loureiro, and J. M. Nogueira. Coverage Area Management for Wireless Sensor Networks. *International Journal of Network Management*, 2006. (published online: `doi.wiley.com/10.1002/nem.604`).

[85] S. Slijepcevicand and M. Potkonjak. Power efficient organization of wireless sensor networks. In *Proc. of IEEE Int. Conf. on Communications*, 2001.

[86] Sun Microsystems. Enterprise JavaBeans (EJB). `http://java.sun.com/products/ejb/` (as of March 2007).

[87] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat Monitoring with Sensor Networks. *Communications of the ACM*, 47(6):34–40, 2004.

[88] TinyOS. An open-source operating system for wireless embedded sensor networks, 2007. `http://tinyos.net/` (as of March 2007)).

[89] UDDI Consortium. UDDI Executive White Paper, November 2001. `http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf` (as of November 2006).

[90] M. Uschold and M. Gruninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[91] W. M. P. van der Aalst and A. H. M. ter Hofstede. Workflow patterns: On the expressive power of (petri-net-based) workflow languages. In *Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, 2002.

[92] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Advanced workflow patterns. In *Proc. 7th International Conference on Cooperative Information Systems (CoopIS 2000)*, 2000.

[93] W3C. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/soap/` (as of November 2006).

[94] W3C. Web Service Semantics. `http://www.w3.org/Submission/WSDL-S/` (as of November 2006).

[95] W3C. Web Services Activity. `http://www.w3.org/2002/ws/` (as of November 2006).

[96] W3C. Web Services Choreography Description Language 1.0. `http://www.w3.org/TR/ws-cdl-10/` (as of November 2006).

[97] W3C. Web Services Description Language (WSDL) – Version 2.0 Part 1: Core Language. `http://www.w3.org/TR/wsdl20/` (as of November 2006).

[98] W3C. World Wide Web Consortium. `http://www.w3.org/` (as of November 2006).

[99] W3C. Web Services Description Requirements, October 2002. `http://www.w3.org/TR/ws-desc-reqs/` (as of November 2006).

[100] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt andG. Schuster, H. Neumann, and S. Hübner. Ontology-based Integration of Information – A Survey of Existing Approaches. In *Proc. of the IJCAI-01*, pages 108–117, 2001.

[101] J. Wainer, M. Weske, G. Vossen, and C. B. Medeiros. Scientific Workflow Systems. In *Proc. of the NSF Workshop on Workflow and Process Automation Information Systems*, 1996.

[102] Q. Wang, H. Hassaneim, and K. Xu. *Handobook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter A Practical Perspective on Wireless Sensor Networks, pages 166–193. CRC Press, 2004.

[103] S. Wang, E. Rundensteiner, S. Ganguly, and S. Bhatnagar. StateSlice: New Paradigm of Multiquery Optimization of Windowbased Stream Queries. In *Proceedings of the 32nd VLDB Conference*, 2006.

[104] Webopedia: Online Computer Dictionary for Internet Terms. Definition of Web Service. `http://www.webopedia.com/TERM/W/Web_services.html` (as of November 2006).

[105] WfMC. Workflow Management Coalition. `http://www.wfmc.org` (as of November 2006).

[106] WfMC – Workflow Management Coalition. The Workflow Reference Model. Technical report, Workflow Management Coalition, 1995. TC-1003.

[107] WfMC – Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language (XPDL). Technical report, Workflow Management Coalition, 2002. TC-1025.

[108] World Wide Web Consortium (W3C). Web Services Activity. `http://www.w3.org/2002/ws/` (as of March 2007).

[109] Workflow Patterns. `http://www.workflowpatterns.com/` (as of November 2006).

[110] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3), 2002.

[111] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3), 2002.

[112] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49, 2005.