

GENERALIZING DISTRIBUTED SENSING NETWORKS *

Jens Küspert

Dirk Kutscher

J.Kuespert@medusa.uni-bremen.de D.Kutscher@medusa.uni-bremen.de

Center of Information Technologies
(TZI)
University of Bremen
Germany

ABSTRACT

Recent research in airborne oil spill remote sensing [FBFG94] leads towards modular systems that consist of several distinct sensors to combine the capabilities of the different sensor classes. The Medusa project [GHW96] is an example of a distributed system. It exhibits a distributed architecture to provide a maximum of flexibility, concurrency and safety and must clearly be rated as a classical distributed application from a computer science point of view. This article describes the "sensor description system" (SDS). SDS allows the developer of sensing systems to minimize the effort of integrating his particular subsystem into an existing application. By applying formal methods to the integration process a developer is able to describe the abstract properties of his sensing system like parameter values, generated data format, applicable methods on the data etc. and can thus rely on the SDS tools to produce the required software backends automatically: A graphical user interface for parameter control, an online visualization, data transfer facilities to a database and finally the evaluation and interpretation facility. This technique puts future sensing enterprises in a position where different classes of sensors can easily be combined almost off-the-shelf to build powerful systems in very short turnaround times.

1.0 INTRODUCTION

This section describes the kind of sensing systems that SDS is targeted at. It is motivated why generalizing these systems is feasible.

1.1 MULTI SENSING PLATFORMS

Many remote-sensing applications (like oil spill detecting) require the capability to control and to process input of a set of (different) sensors in order to overcome the shortcomings of specific sensors and combine certain sensing properties (for example to scan a large range of wavelengths). Distributing a sensing system is also a way to avoid "single point of failure" situations and to enable concurrency.

*Presented at the Second International Airborne Remote Sensing Conference and Exhibition, San Francisco, California, 24-27 June 1996

Beside the design and implementation of each sensing subsystem these *distributed* applications require solutions for other problems, mostly related to the integration of the subsystems: It must be possible to control the subsystems (to verify correct operation, to set parameters etc.) and data flow between sensing instances has to be managed. Finally suitable methods should be applied on the gathered data.

Finding solutions for these rather problem oriented issues implies the necessity to deal with other, rather "low-level" aspects in order to put the idea of *distributed* systems into practice:

- identifying the subsystem entities
- implementing communication relations between subsystems
- implementing data transport and visual representation of gathered data
- managing concurrency

1.2 AN EXEMPLARY SENSOR INTEGRATING SYSTEM

Medusa [GHW96] is an example of such a distributed system that deals with these issues. The project was initiated by the German Ministries of Transport and Defense. Taking the issues mentioned above into account the following design premises for this system have been demanded:

1. The whole system should consist of more than one computer to minimize failures. The components are connected via a standard local network, the communication is based on the TCP/IP family of transport protocols.
2. Every sensing subsystem should accumulate the real time data on its own (i.e. concurrently to other sensing subsystems) in order to relieve the integrating system from the real-time requirement.
3. The user interface should integrate the different sensing subsystems in such a way that the operator can see (and control) it as a whole. The distributed system should be consistently controllable like a single application.

Some features of Medusa are elaborated more precisely in the following in order to explain some terms and principles that are later referred to. Nevertheless most of the essential characteristics should be common to the class of distributed sensing networks in general.

1.2.1 Integration And Communication

The different sensing subsystems communicate with the COC[†] on three levels as shown in figure 1:

1. Controlling the sensing backend by changing parameters or influencing other operational functions.
2. Transmitting "online"-data (transient data with real-time characteristics) that is required for visualizing, for verifying the sensor's operation or for information interchange between the subsystems.[‡]
3. Transmitting the gathered data into a database (for further analysis and backup).

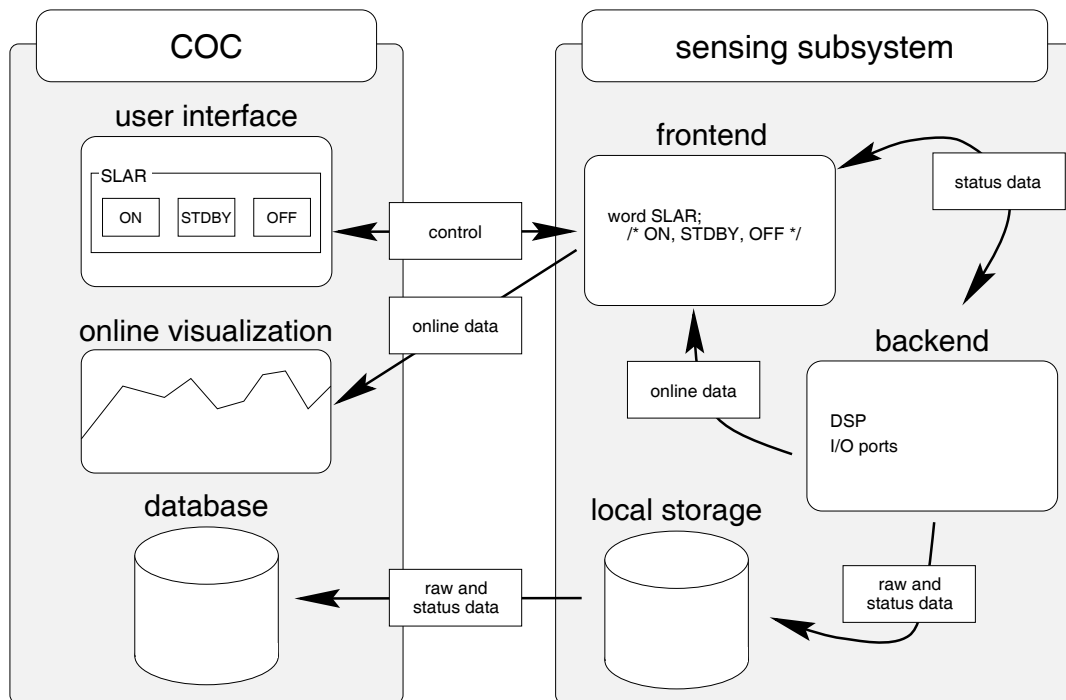


Figure 1: Communication

All forms of communication are replicated for all sensing subsystems and can thus be subject to standardization. There are some details that can be separated logically from the data gathering module on the sensing subsystem like mechanisms of inter-client communication and data-transport. However each subsystem has to provide a "communication module" that supports these services.

1.2.2 The Sensing Subsystems

Figure 1 shows that such a sensing subsystem consists of two components:

Backend

It is responsible for controlling the actual sensing hardware (e.g. by setting specific values at certain I/O ports) and gathering the actual data.

Frontend

This component communicates with the "outer world" to allow this subsystem to be controlled (e.g. by the COC). Further it is responsible for submitting the online-data that is to be visualized.

[†]Central Operating Console - the control and visualization instance

[‡]Medusa provides a "Navigation"-sensor that records position, speed etc. This information is needed by another sensor for data processing. The concept of transient data allows the efficient information interchange for this or similar purposes.

1.2.3 Open Interfaces

We have seen that open interfaces between subsystems are required to submit efficient and safe data communication and control.

The interfaces between sensing subsystems and COC are indeed well defined, but nevertheless this does not guarantee that a potential *new* sensor will correspond to this informal definition. Integrating a new sensor into an existing framework is therefore a time consuming and error prone task: This includes programming the sensing subsystem as well as reaching a state of interoperability that allows the integration. A formal method (like a type system of a programming language) would be required to assure that each sensing subsystem corresponds to the demanded conventions.

Due to the similar architecture of all sensing subsystems many functions and services are duplicated. There are common code modules (such as the frontend) that could be shared between sensing subsystems. It seems desirable to formalize this information and generate the appropriate code.

2.0 SDS: FORMAL SPECIFICATION OF SENSING NETWORKS

2.1 ABSTRACTION AND GENERALIZATION

When designing and integrating new sensing subsystems the developer doesn't want to care about shared memory addresses, TCP/IP connections, port numbers, control elements of user interfaces and things like that. Of course these details are important and must be considered carefully, but since they appear in every subsystem instance it makes more sense to implement these basics once. If it is possible to generalize the technical details of the subsystems, it must be possible to find an abstract way of describing the *crucial characteristics* that make up the nature of a sensing subsystem.

In the previous section (1.2) we showed the elementary characteristics that can be used to describe a sensing subsystem. The logically distinct function categories are control, visualization and data evaluation. We created a specification system that allows describing a sensor by defining its concrete nature expressed in terms of this system in a rather abstract way. Specifying the sensor controllable parameters for example does *not* require to talk of variable addresses, representation as GUI[§] elements and remote procedure calls but rather consists of naming the control parameters and annotating their respective properties like data type (numeric, boolean etc.), range of values and the functional context. The text entity containing this specification is called "the description file".

Describing a sensor this way can be compared to writing a program in a programming language: There are certain syntactical (and semantical) rules that have to be adhered by the programmer in order to make the description document compliant. Only documents structured consistently are treated as valid specification instances. For example coding a sensor's control parameters also requires to specify its name and its properties in order to keep the document consistent. Most common programming languages [Wat90] are universal, i.e. not application specific. Since describing a sensor's properties is a very special application it would not be suitable to use a generic programming language for this task. Because of the high amount of formalization that can be applied we have chosen an approach that allows the specification to be written almost as easy as completing a fill-in form: The logical constraints of (for example) describing a control parameter can be used to facilitate the editing and validating process by tools as application aware editors etc.

[§] graphical user interface

The information coded in the description file is used to generate the appropriate code segments for interprocess communication, generating user interfaces, storage access and hardware controlling (see figure 2).

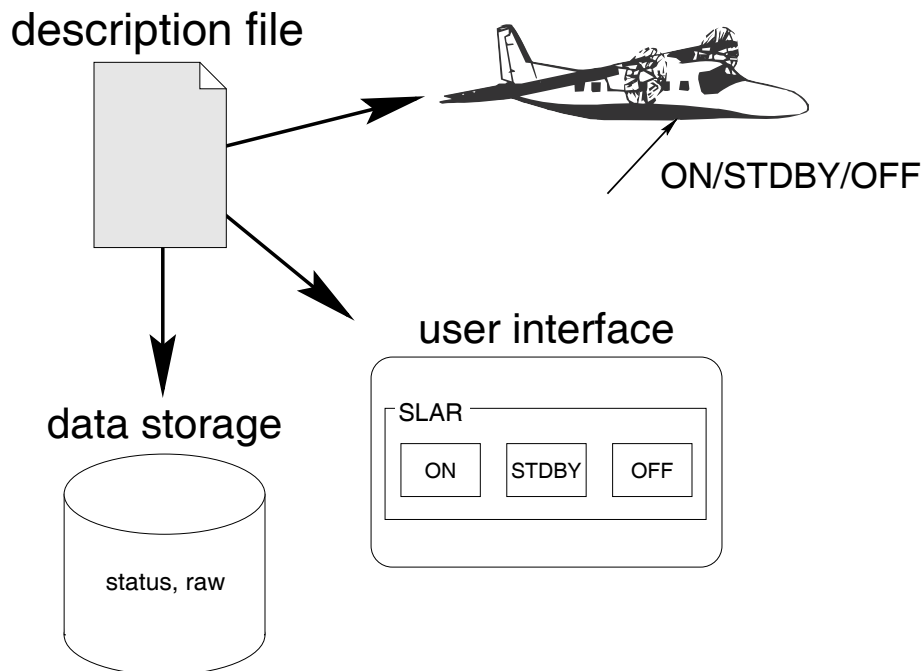


Figure 2: Schematic view of a sensing subsystem

2.2 TECHNIQUES

As noted above description files are logically structured and the specification of a sensor is formalized to a certain degree. There are many ways to implement a specification system that meets the required demands, there are specifications languages such as the "Z" notation [BN92] that are targeted at complex systems to provide a possibility to verify and proof their correctness.

Since describing sensors mainly results in parametrizing software modules that are already constructed there is no need to complicate the process of specifying a sensor with tools used for design verification etc. What is required is a notation that makes allowance for expressing the logical structure of these description documents.

SDS utilizes ISO 8879:1986 (SGML)[Go190] to provide the syntactical framework for the required logical structure. The logical structure of a SDS document is defined in the SDS-DTD[¶], which is an integral part of SDS and describes the class of valid SDS documents. It and can be used by any SGML-Editor to provide context sensitive editing help, validating services etc. It is even possible to use a simple Text-Editor, but this is error prone and thus deprecated. It is possible to generate different output-formats from the SGML-document for different purposes, an overview of the target formats supported by SDS so far is given in the next section.

[¶]A "document type definition" (DTD) is an abstraction, the description of a class of documents that an application designer is using SGML to represent. [Go190, p. 302]

2.3 INTEGRATION OF DESCRIPTION AND DOCUMENTATION

SDS is not only targeted at describing sensor parameters and data formats. It is possible to include parts of the sensor documentation into the file entity in order to describe the sensor formally as well as informally. Developing and integrating a sensor thus becomes an act of *literate programming* [Knu84] where formal specification is embedded into informal documentation (or vice versa).

Integrating the documentation into the specification document is straightforward: The documentation itself inhibits a logical structure in order to avoid layout-details complicate the documentation process.

By structuring documentation text on a higher level of abstraction it becomes possible to generate a range of output formats: PostScript for printed output, HTML for online documentation, RTF and more (see figure 3).

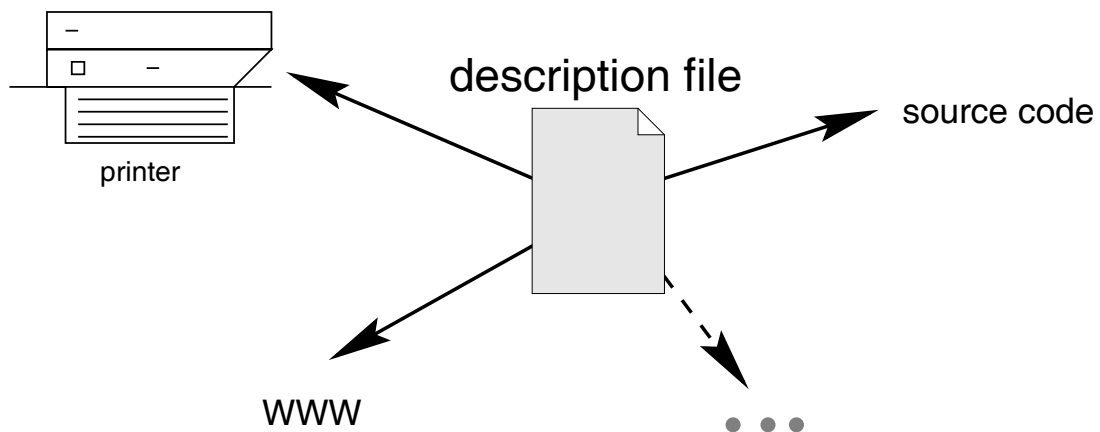


Figure 3: Possible output formats

The advantages of formal sensor specification are evident: It is not only easier to construct and integrate a particular sensor in an sensing network because of the possibility to reuse many software components, the sensor designer also benefits from the abstract specification, because he gains a clearly structured view on how the sensor interface can be described and can also integrate the documentation of his subsystem into the specification for later formatting.

2.4 IMPLEMENTATION

The components of a sensing subsystem consist of two different kinds of modules:

1. Commonly used code such as managing TCP/IP connections, that is used in all sensing subsystems.
2. Code that is specific for each sensing subsystem, but related in structure and purpose (such as communicating with the COC). This code would be manually written in traditional systems, but in SDS this is merely a new output format like the others described in section 2.3.

The Medusa project focuses on sensors that are designed for the real time operating system OS-9, so SDS concentrates on this platform. But both forms of code are designed to be easily portable to other platforms. In effect there is an abstract interface to all operating system specific functions. The generated code relies on that interface so code generation and sensor description would be unaffected by supporting other platforms.

```

<selection vals='ON, STDBY, OFF'>
  <name>Power</name>
  <docu>
    <p>
      The SLAR has three distinct power states (...)
    </p>
  </docu>
</selection>

```

Figure 4: SGML description of a control field

Power

- Type: SELECTION
- Values: ON, STDBY, OFF
- Description:

The SLAR has three distinct power states (...)

Figure 5: A possible HTML rendering

2.5 EXAMPLES

The complete description of a sensing subsystem would be beyond the scope of this article, so we will focus on a specific function of one sensor. To describe the power control facility of the sensor SLAR the description shown in figure 4 might be used. Exemplary for all possible documentation output formats a rendering of the HTML output is shown in figure 5. Generated C++ source code for the frontend is finally shown in figure 6.

3.0 FUTURE WORK

The work on SDS only begins to show its impact on how we look at a sensing system. We have begun to see sensors not only as a hardware entity, but also as a logical data-source. It seems attractive to expand this view of the system to other aspects of Medusa. This foosts on the observation that any sensing system is concerned with the following actions:

```

SM_Selection* s=new SM_Selection;
s->path("SLAR.Control.Power");
s->longDescTemplate("radio(\"Power\",%s,[\"ON\", \"STDBY\", \"OFF\"])\");
s->mem(global, offsetof(SYS_DMOD, System.Parameter.ControlPower));

```

Figure 6: Generated C++ code for the frontend

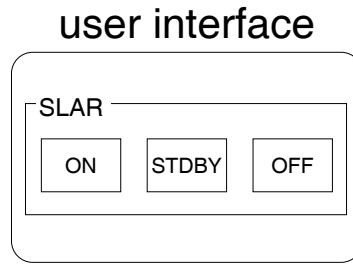


Figure 7: Generated Graphical User Interface

Data collection and sensor controlling

SDS generalizes these topics as described above.

Data interpretation and visualization

It is necessary to perform certain interpreting steps on the collected sensor data. This can be generic actions, such as marking "oil-spills" or extracting a specific range of data. But it is also possible to perform sensor dependant actions, such as classifying an oil-spill (a functionality provided by the sensor LFS [HR90]).

Therefore it seems a good idea, to describe these algorithms on a higher level, the generic methods will impersonate the expertise of the end-user, whereas the specific methods will expand the abstract sensor-description provided by SDS

An advanced concept would be the notion of "virtual sensors": A SDS-description file might not describe a hardware sensing subsystem at all. It is possible to describe another form of data-input (not only hardware components as in the current system), e.g. the image data collected by the LFS. Applying the method "classify oil-spill and filter heavy-fuel oil occurrences" on this data stream could result in a virtual sensor for this specific kind of oil. Other possible scenarios might be:

- Filtering oil-spills that are significant for legal prosecution
- Providing support for oil-spill combating on the sea surface

4.0 CONCLUSIONS

The work on Medusa has shown that generalizing distributed sensing networks is feasible even for a relatively small number of sensing systems (five systems at this moment). Generalization and abstraction allow to automate the process of sensor integration to a great extent. From a system design point of view a sensor is no longer a unique subsystem requiring manual integration. It can rather be treated as a "sensing-server" with an open, well-defined interface:

1. It needs to be controlled.
2. It gathers data and thus contributes to the perception of the whole system.

These characteristics are defined in each sensing subsystem's description file. The respective concrete software moduls can be generated (or at least parametrized) accordingly.

The formal specification of sensor is an appropriate method to assure the safety and usability of the sensing system.

Yet there is still some work to do: It would be desirable to define a sensor not only as a data source but also to formalize all possible aspects of data processing, for example evaluation methods. This would lead to an even more object oriented [Mey88] way of viewing sensing networks: There are services (data gathering and data evaluation), objects (the gathered data) and methods that are applied by the sensing instances on their objects. The focus would no longer be on individual sensor but more emphasis would be put on data streams and their properties.

5.0 ACKNOWLEDGEMENTS

The work presented here has been done in the development of Medusa and has been initiated by Dr. Theo Hengstermann of optimare.

The authors would like to thank Holger Dürer for his help in the preparation of this paper.

References

- [BN92] S. M. Brien and J. E. Nicholls. Z base standard. Technical report, OUCL, Oxford, Nov 1992. Accepted for standardization under ISO/IEC JTC1/SC22.
- [Com91] Douglas E. Comer. *Principles, Protocols, and Architecture*, volume I of *Internetworking with TCP/IP*. Prentice-Hall International Inc., 2nd edition, 1991.
- [FBFG94] M.F. Fingas, C.E. Brown, M. Fruhwirth, and L. Gamble. Assessment of sensors for oil spill remote sensing. In *Proceedings of the First International Airborne Remote Sensing Conference and Exhibition*, volume 1, pages 1–12. Environmental Research Institute of Michigan, 1994.
- [GHW96] Konrad Grüner, Theo Hengstermann, and Manfred B. Wischnewsky. Recent results obtained from a 2nd generation surveillance system for identification of maritime pollution. In *Proceedings of the Second International Airborne Remote Sensing Conference and Exhibition*. Environmental Research Institute of Michigan, 1996.
- [Gol90] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.
- [HR90] T. Hengstermann and R. Reuter. Lidar fluoresensing of mineral oil spills on the sea surface. *Applied Optics*, 29:3218–3227, 1990.
- [Knu84] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):pp. 97–111, 1984.
- [Mey88] Bertrand Meyer. *Object-oriented Software Construction*. International series in computer science. Prentice Hall, 1988.
- [VK93] Paulo Verissimo and Herrmann Kopetz. Design of distributed real-time systems. In Sape Mullender, editor, *Distributed Systems*, chapter 19, pages 511–528. Addison-Wesley, 2nd edition, 1993.
- [Wat90] David A. Watt. *Programming Language Concepts and Paradigms*. International series in computer science. Prentice Hall, 1990.