Specification and Role of the Sensor Manager In the Science/Autonomy System

Lenny J. Delligatti   Dimi S. Apostolopoulos

CMU-RI-TR-99-08

Submitted in partial fulfillment of
the requirements for the
CIT Honors Research program

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 1998

This research was supported in part by NASA grant NAG5-7707. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of NASA, Carnegie Mellon University, or the U.S. Government.

Abstract

The NOMAD project focuses on producing a rover capable of autonomously searching for and classifying Antarctic meteorites. The rover is equipped with a sensor array and a network of software modules, the Science/Autonomy system, which work in conjunction to enable the rover in its aforementioned task. The sensor manager is a communications interface between the sensor array and the higher-level decision making modules of the Science/Autonomy system. The use of the sensor manager in the system simplifies the system's communications web while making the details of sensor operation and control transparent to the higher level modules. The current implementation of the sensor manager uses a sensor class to give the sensor hardware a representation in software, which changes to reflect the current status of the sensor as it runs through a series of operations. NDDS is used for communication with the sensor manager. Future implementations will give the sensor manager a decision making capability with respect to which sensor to use under a given set of circumstances.

1 Overview of the (SAS) Science/Autonomy System

The SAS hardware is comprised of the sensor array and the mobile platform on which they are mounted. The SAS software relevant to sensor control and operation is comprised of the mission planner, database, sensor manager, and the sensor device drivers.

1.1 Platform and Sensors

The SAS sensor array contains a high-resolution color camera and a spectrometer. Ultimately, this array will be expanded to include a magnetometer and metal detector. The system, also, uses a panoramic camera for environment imaging and eventually for landmark-based navigation, however, this sensor is not under the control of the sensor manager. All sensors are mounted on the NOMAD mobile platform.

1.2. Software Modules

The relationships of the software modules relevant to sensor control and operation are shown in Figure 1.

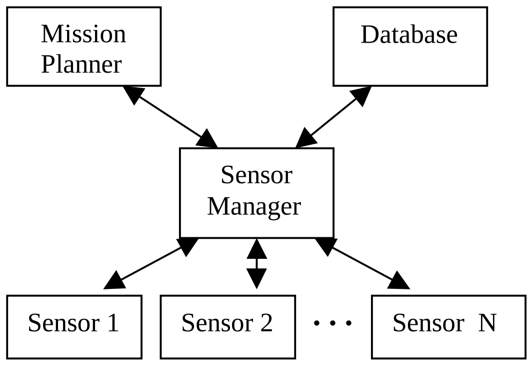


Figure 1

The mission planner is responsible for making decisions with respect to whether NOMAD will continue to follow a search pattern or deviate to investigate a target of interest.

The database contains records for all targets encountered. These records include the target's GPS coordinates, class probability statistics, as well as the filenames for sensor data.

The sensor device drivers, ultimately, will be responsible for actuating the sensor's deployment mechanism as well as interfacing with the hardware to acquire data and save this data to disk.

The sensor manager is primarily responsible for making the details of individual sensor control and operation transparent to the mission planner when it sends the command to deploy a sensor or use it to acquire data from a target.

2 Sensor Manager

2.1 Role of the Sensor Manager

The sensor manager provides a communications interface between the sensor device drivers and the higher level modules of the science/autonomy system. Every sensor has unique requirements and procedures for deployment and data acquisition. The sensor manager's primary job is to hide the details of a sensor's operation under an umbrella of a simple functional interface.

A secondary function of the sensor manager is to simplify the communications web in the SAS. The system without the sensor manager would appear as shown in Figure 2. The mission planner would have to communicate with each sensor individually to acquire data, cost estimates, or to deploy the sensor. The sensors, in turn, would then have to communicate with the database on an individual basis in order to get target data such as GPS position which it will need to deploy, as well as to save the data file names after data acquisition. Use of the sensor manager allows each module in the system to establish fewer NDDS ports required for communication (see section 3.1).

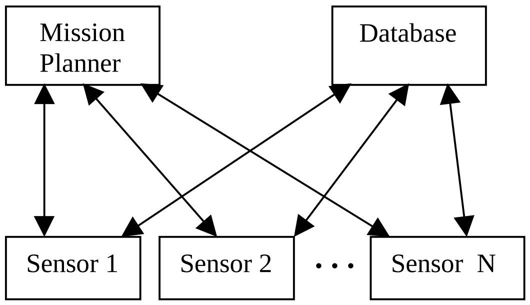


Figure 2

The benefit of using the sensor manager, for purposes of simplifying the communications web, increases in direct proportion to the size of the sensor array mounted on the rover.

In its first implementation, the sensor manager has no decision making capability. The decision of which sensor to use for data acquisition of a particular target will be made by the mission planner and passed to the sensor manager which will process the message and take the necessary steps to see that the command is executed. These steps include everything from sending a request to the database for target data needed by the sensors, to sending messages to the sensors for calibration, diagnostics, and data acquisition. The sensor manager, in this implementation, effectively acts as the central communications hub of the SAS for purposes of sensor control. See section 4.1 for a discussion of future implementations of the sensor manager in which it will have increased responsibility for making decisions about sensor usage.

2.2. Architecture

Each sensor is represented in software by a sensor object contained within the sensor manager. The sensor manager maintains an array of these objects which together represent the sensor array mounted on the NOMAD platform (refer to Figure 3).

In the current implementation, the array of sensor objects in the sensor manager acts as a status board. The sensor class contains fields that keep track of the current state of a sensor as it passes through a series of operations (see section 3.2.1 for a listing of these fields). As new commands are received for the sensor, the sensor manager first checks the corresponding sensor object to determine the current state of the hardware to determine an appropriate sequence of steps to ensure that all commands are executed reliably, or failing this, ensuring that an appropriate status message is returned to the mission planner.

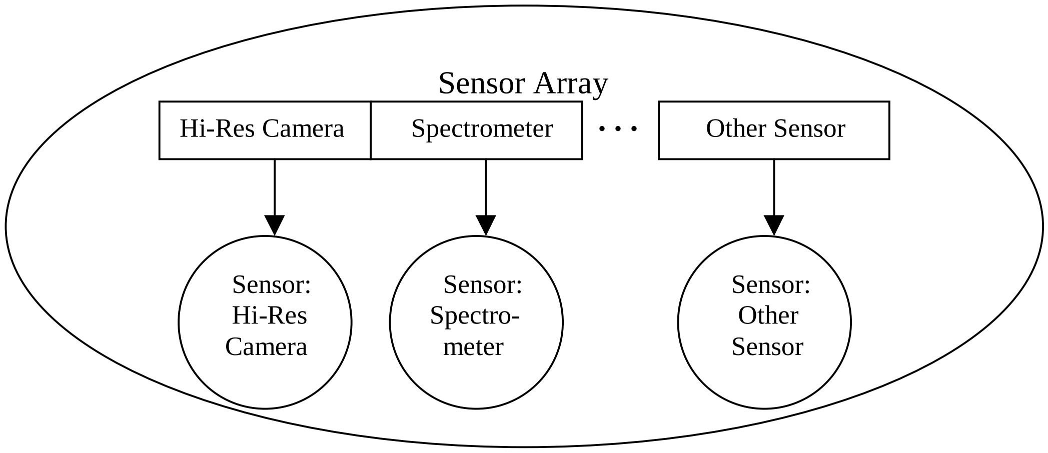


Figure 3: Sensor Manager Module

3 Implementation

3.1 Network Data Delivery System (NDDS)

3.1.1 Consumers and Producers

NDDS is the communications package used to transmit data between the software modules in the SAS. Currently, NOMAD has one computer devoted to the SAS software. In the future, these modules may be run on different host computers. Each computer in the network, however, must be running the NDDS application to establish it as a node in the link.

Each application must then create NDDS consumers and producers to establish channels of communication with other modules. The NDDS consumers are ports dedicated solely to the reception of particular messages which are registered with them. Messages of other types are ignored. Producers, like the consumers, are dedicated to particular messages, which they sample and send out across the network. Refer to Figure 4 for an illustration of the SAS modules with consumer and producer ports.

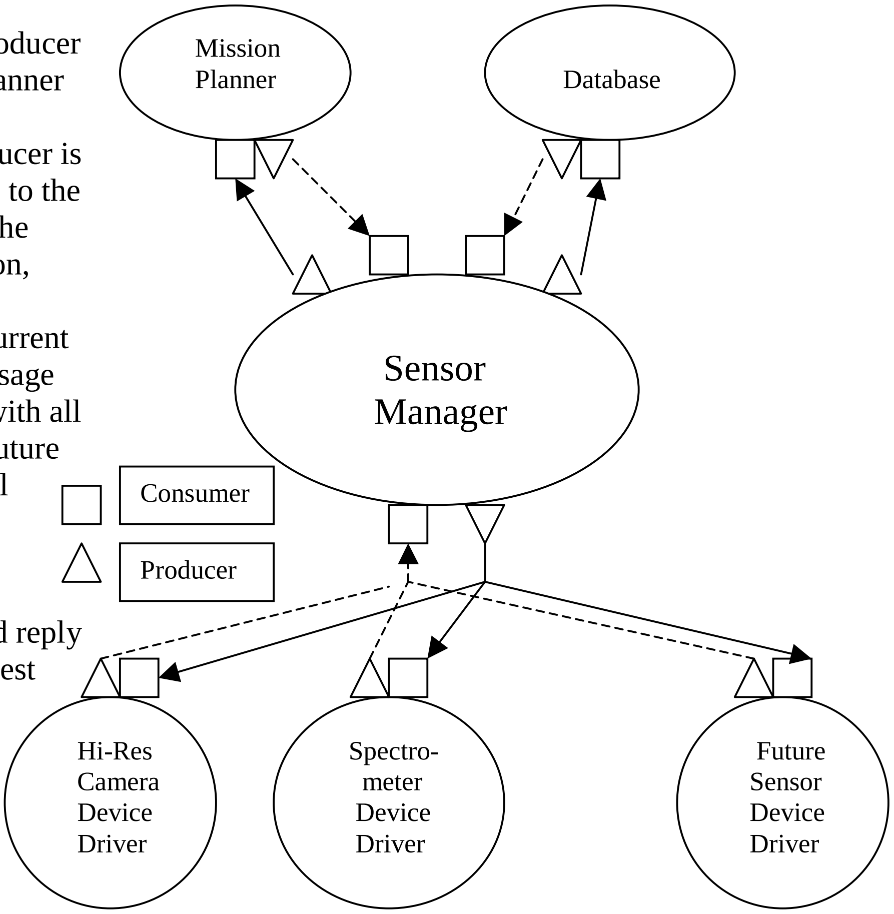
The sensor manager has 3 NDDS producers: a database request producer, a sensor manager status producer, and a "to sensor" message producer. The database request producer sends messages to the database to perform "get" and "put" functions to retrieve target data for sensor deployment and to store data file names after data acquisition, respectively.

The sensor manager status producer sends messages to the mission planner to report on the results of sensor operations. The "to sensor" producer is responsible for sending messages to the sensor device drivers to execute the various functions (i.e. initialization, calibration, deployment, data acquisition, shutdown). In the current implementation, one generic message type is used for communication with all of the sensor device drivers. In future implementations, each sensor will have its own dedicated message and producer.

Additionally, the sensor manager has 3 consumers: a field reply consumer, a sensor manager request consumer, and a "from sensor" message consumer. The field reply consumer receives messages from the database, which return the requested data. The sensor manager request consumer receives messages from the mission planner which requests a cost estimate, deployment, or data acquisition. The "from sensor" message consumer receives messages from the sensors which either report on operation status or request data from the database.

3.1.2 Callback Functions

When a consumer receives a new message, NDDS calls a handler routine (callback function) which either calls a function in the sensor manager or pa ı˝ D0 Tc489.6›7.93e


Mission Planner
Database
Sensor Manager
Consumer
Producer
Hi-Res Camera Device Driver
Spectro-meter Device Driver
Future Sensor Device Driver

3.2 Language and Data Structures

The sensor manager was developed with the Microsoft Visual C++ Developer Studio 5.0 using the Microsoft Foundation Class (MFC) package.

3.2.1 Sensor Class

```
class Sensor {

//static sensor parameters
float       Workspace_parms;   // *NOTE* struct to be defined later
CString     SensorName;
int         NumPowerModes;
float       *ModePowerConsumption;
float       RateOfDeployment;  //(arc sec./sec) *to be refined later
float       *HeatOutput;  //(W) heat produced by unit
float       SensorLifetime;  //finite num. uses? time period?
Float       DAQtime;       //time period to acquire data sample
float       restInterval;  //time period between successive DAQ's
float       storageSpace;  //memory (kBytes) required per data sample
float       Risk;    //risk to sensor/system associated with use?
float       PercentCPUcycles;   //CPU usage by sensor
float       MinSensorOpTemp;  //min. operating temp. of sensor (C)
float       MaxSensorOpTemp;  //max. operating temp. of sensor (C)

//dynamic sensor variables:
int         CurrentPowerMode;
int         SensorState; //see sensor state enums. in file smEnums.h
float       RemainingLifetime;
float       CurrentSensorTemp; //in degrees Celsius

    //class methods:
    Sensor (CString filename);
    ~Sensor (void);
};//end of Sensor class declaration
```

This excerpt of code shows the sensor class declaration. The static sensor parameters are taken from a data file associated with the given sensor. These fields are set when the sensor object is created.

The dynamic sensor variables reflect the current state of the sensor. These fields (most importantly the sensor state field) serve as the sensor's status board which is checked prior to the execution of a new command.

3.2.2 Sensor Manager Class

The sensor manager class contains an array of pointers to sensor objects (refer to Figure 3). The array is indexed with a unique sensor ID sent in a message from another module.

This class contains two methods in addition to the constructor and destructor: a Sensor Object Create and a Sensor Object Destroy method (see section 3.3.3 for a more in-depth discussion of these functions).

3.3 Functional Interfaces

The following enumeration lists the possible values that can be placed in the function field of a message to the sensor manager from another module (i.e. mission planner, database, sensor device drivers). The following sections discuss the above set of functions in relation to the modules in which they are implemented. Since none of the above functions are implemented in the mission planner, it's functional interface is not of relevant importance.

```
typedef enum {
            //initiated by sensor device drivers
            REGISTER_SENSOR_FUNCTION,
            UNREGISTER_SENSOR_FUNCTION,
            SAVE_DATA_FUNCTION,
            GET_DATA_FUNCTION,
            SENSOR_REPORTING_FUNCTION,

            //initiated by mission planner
            ACQUIRE_DATA_FUNCTION,
            DEPLOY_SENSOR_FUNCTION,
            STARTUP_SENSOR_FUNCTION,
            SHUTDOWN_SENSOR_FUNCTION,
            SENSOR_DIAGNOSTIC_FUNCTION,
            COST_ESTIMATE_FUNCTION,

            //initiated by database
            RETURN_DATA_FUNCTION
} SM_FUNCTION;
```

3.3.1 Sensor Device Drivers

```
typedef enum {
		READY_STATE,
		BUSY_STATE,
		WAITING_STATE,
		RESTING_STATE,
		MALFUNCTIONING_STATE
} SM_SENSOR_STATE;
```

diagnostic returns a sensor malfunctioning status report.

The sensors, additionally, have an off-line state, however, this is determined by checking if the corresponding sensor object pointer equals NULL. The off-line state is equivalent to the sensor object being uninstantiated.

3.5 Sensor Operation Status

The following enumeration lists the possible values that can be returned in the status field of a sensor manager status message to the mission planner:

```
typedef enum {
            SENSOR_OFF_LINE_STATUS,
            SENSOR_READY_STATUS,
            SENSOR_BUSY_STATUS,
            SENSOR_WAITING_STATUS,
            SENSOR_RESTING_STATUS,
            SENSOR_MALFUNCTION_STATUS,
            SUCCESSFUL_STARTUP_STATUS,
            STARTUP_FAILURE_STATUS,
            SUCCESSFUL_DEPLOYMENT_STATUS,
            DEPLOYMENT_FAILURE_STATUS,
            SUCCESSFUL_SHUTDOWN_STATUS,
            SUCCESSFUL_DAQ_STATUS,
            DAQ_FAILURE_STATUS,
            OUT_OF_WORKSPACE_STATUS
} SM_SENSOR_STATUS;
```

The first six directly correspond to the sensor state field of the sensor object or lack of an existing object entirely (e.g. off-line).

The next seven correspond to the success or failure of the various sensor operations, which can be initiated by the mission planner.

The final status, out of workspace, indicates that a data acquisition failure occurred due to the indicated target being out of range of the chosen sensor's deployment mechanism. This status is reserved for future implementations.

4 Future Work

4.1 Role of the Sensor Manager

Future implementations of the sensor manager will have a decision-making capability regarding which sensor to use for data acquisition under a particular set of circumstances. This decision will be based on a cost estimate function, which takes into account factors such as time to deploy a sensor and acquire data from a target and the power consumed by the sensor in performing these operations. It will also take into account environmental factors such as accessibility and size of the target, as well as statistical data such as the estimated info. gain expected from a given sensor.

4.2 Implementation

The next line implementation of the sensor manager will use an array of sensor threads, which will be responsible for processing messages sent to the corresponding sensor and for ensuring that the appropriate steps are taken to execute the commands reliably.

In the current implementation, processing and distribution for all messages is performed within the NDDS callback routines of the sensor manager. The disadvantage of this design is that these callback routines run within the context of the sensor manager application. NDDS does not spawn individual threads to handle the processing of each message. Thus, it is possible for incoming messages to be blocked while one is currently being processed.

In the next implementation, all incoming messages will initially be received by one of the sensor manager's consumers, but will immediately be inserted into a queue devoted specifically to a particular sensor's messages. This frees up the sensor manager's main thread to receive new messages while providing the individual sensor threads in the array the flexibility of processing the messages in the queues in a non-time-critical manner.

5 Conclusion

The use of the sensor manager in the science/autonomy system serves several purposes. The primary purpose is to hide the low-level details of a sensor's operation from the higher level modules. This provides the mission planner a simple functional interface to communicate with the sensors to perform the critical tasks of sensor deployment and data acquisition.

This design has the added benefit of simplifying the communications pathways between the modules in the SAS. In its role as a communications hub, the sensor manager reduces the number of NDDS ports that would be required for each module to independently establish communications with one another.