

# The Satellite Data Model

Kenneth Sundberg, Scott Cannon, Todd Hospodarsky  
Space Software Lab  
Dept. of Computer Science  
Utah State University  
Logan, UT 84322-4305

Don Fronterhouse  
Phillips Lab  
Air Force Research Laboratory

## Abstract

*The Satellite Data Model (SDM) is part of the Air Force Research Laboratory (AFRL) Responsive Space Testbed Initiative. It is a developing standard for rapid integration of hardware and software components in a satellite network.*

*The SDM is a middleware software system providing self-configuration and self-discovery capabilities in multi-processor networks of applications, sensors, and actuators. By allowing both software and hardware to self-discover and self-configure, the SDM supports rapid development and integration of satellite networks.*

*The SDM is intended to provide a completely generalized device-application interface whereby applications can be written independently of their data sources and messaging and without knowledge of the final data consumers. Similarly, sensors and actuators can be built without concern for specific applications or what processes they will interact with.*

## 1 The Responsive Satellite

The primary goal of the Responsive Space Testbed Initiative of the AFRL is to support the construction of a functioning satellite in a six day window from conception to launch. The Satellite Data Model is a critical component of this effort. To achieve these aims, a large consortium of application and device developers is currently working together to produce SDM compatible subsystems that can be quickly integrated for rapid-response tactical satellites.

In a six day window there is no opportunity for development or testing of software. To accomplish this six-day goal, the Satellite Data Model must support the development and testing of all software prior to the six-day integration with hardware and other software components. The Satellite Data Model software was written to perform the necessary auto-configuration and integration tasks for hardware and software components. The SDM also provides a generalized interface between components.

An SDM-compliant component in the Responsive Space Testbed consists of a plug-and-play communications interface and the ability to describe data products, data and message formats, and controls in an xTEDS document. The component must then have sufficient intelligence to respond to standardized messaging from the SDM as the SDM serves as a broker between data sources and destinations.

## 2 Overview

The SDM is a middleware software system providing self-configuration and self-discovery capabilities in multi-processor networks of applications, sensors, and actuators. By allowing both software and hardware to self-discover and self-configure, the SDM supports rapid development and integration of satellite networks.

For devices, the SDM is based upon plug-and-play hardware communications interfaces. In addition, the SDM supports allowing an intelligent device to describe its controls, data products, data formats to the network at the time of self-discovery in a standardized XML device document using xTEDS (eXtended Transducer Electronic DataSheet) XML format.

For applications, the SDM allows application software to identify sources of needed data by querying the device documents that are present in the network. Once an application has found an appropriate device, the application may subscribe to data and control data source devices according to their device documents without specific a priori knowledge of the particular device interfaces. Applications may also describe any data products and controls to the system with a similar xTEDS document.

The SDM serves as a data, device, and application broker, matching applications with data sources and destinations and providing standardized descriptions of data and message formats from xTEDS documents. In other words, the SDM allows the development of software applications without a *a priori* knowledge of specific device interfaces, data formats, and controls. An application is supported in

locating and using data from any device that describes its interface and controls in an xTEDS document.

Conversely the SDM allows devices to be built without a priori interface control standards associated with specific applications that will use them. It also provides auto-configuration capabilities for distributed concurrency.

## 2.1 Scenario

When a satellite is conceived, a variety of applications are chosen. Applications would typically include some standard items such as flight software and ground communications as well as mission-specific applications. Appropriate devices are then chosen to provide needed data and support to these applications; attitude sensing and control, ground communications, imaging and other sensors, etc.

When the SDM system boots, it discovers existing hardware and device components through the plug-and-play communications network. Each device is then queried for its xTEDS document for a description of capabilities. These descriptions are combined with information about the physical location and orientation of the associated device and stored by the SDM system.

The SDM system then starts scheduled applications. These applications query the SDM for providers of needed data and capabilities. The SDM matches the queries against the capability descriptions and returns the description of data messages, data formats, and device capabilities and controls to the application. An application then selects the most appropriate capability provider and registers with the SDM system as a consumer for that capability.

The SDM system then forwards the subscription information to the producer device or application – along with the address of the consumer. The producer and consumer subsequently cooperate and communicate in a point-to-point manner.

In the case of device failure, the application is notified of the failure and may query the SDM for an alternate provider. If such an alternate exists, the SDM will return that information to the application. The application then registers for the new provider and thus recovers from the fault.

## 2.2 SDM Components

The Data Manager, Task Manager, Process Manager, Sensor Manager components form the core of the Satellite Data Model. The other component types, devices, and applications are chosen to accomplish the tasks of a particular satellite and communicate through the core of the SDM system.

### 2.2.1 Data Manager

The Data Manager is a central repository for descriptions of the capabilities of devices and applications, including data formats and control messages. These descriptions are formed as XML-based documents called xTEDS (further described in section 2.3). The Data Manager is also at the heart of the publish/subscribe communication system.

Applications and devices register their capabilities, in the form of an xTEDS document, with the Data Manager. Applications that need a data source or some other service, query the Data Manager for an appropriate provider. It is the Data Manager's responsibility to run the query against its xTEDS library and return the appropriate results. The results of this query consist of a provider id, which is used to subscribe, and a description of the messages and data formats to be used in communication with the provider.

The Data Manager is also responsible for knowing the physical locations of every component in the system. When a match is made between a data subscriber and a data publisher, the Data Manager informs the publisher of the subscriber's address and request. After that, all communication is point-to-point.

Finally, the Data Manager also maintains a list of capabilities of inactive applications. If a request for data cannot be matched by any currently available process or device but could be satisfied by an application that is not currently executing, the Data Manager requests that the application be started by the Task Manager.

### 2.2.2 Task Manager

The Task Manager is responsible for scheduling and distributing the running of tasks within the system.

The Task Manager manages the aggregation of compute nodes available in the system. It does this with the aid of the Process Managers which are each responsible for one node. The Task Manager also maintains the load balance between available nodes. It is the Task Manager's responsibility to change system operational modes, for example, by shutting down all noncritical applications for a power-saving mode.

The Task Manager is also a data publisher, making information about running tasks available in the system.

The Task Manager maintains a library of SDM applications as executable files. Whenever there is a request made for a particular application to be run, the Task manager assigns the task to a Process Manager. This decision of which Process Manager to send the task to is made using information about the available resources, the current processing load of each compute node, and the expected requirements of the assigned task. If necessary, the Task Manager sends the executable code to the Process Manager.

### 2.2.3 Process Manager

There is one Process Manager residing on every node capable of general computation in the system. The Process Managers cooperate heavily with the Task Manager to perform their responsibilities. The Process Manager informs the Task Manager of its existence and the capabilities of the node it represents.

When there is an application or task to be run by the SDM, it is assigned to a Process Manager by the Task Manager. In order to avoid unnecessary transmissions, the Process Manager first checks its local filesystem for the executable code for the task. If the file is not found, the Process Manager requests the file from the Task Manager which responds with the necessary code.

The Process Manager also informs the other components of the SDM system in the event of the abnormal termination of any task. The Process Manager is responsible not only for running requested tasks but for terminating them if requested to do so by another component of the SDM system. This lends the Process Manager a key role in switching modes. The Task Manager determines what processes need to be terminated, but it is the Process Manager that terminates the task and informs the other components that this has occurred.

The final responsibility of the Process Manager is to assign port numbers to applications that it manages. As an application must be written to run on any machine and with any other processes, there are no preassigned ports. Instead, an application makes a request for the use of a port number, and the Process Manager returns an available number to the requesting process.

### 2.2.4 Sensor Manager

There is one Sensor Manager on every node with a connection to any hardware device. The Sensor Manager is responsible for device discovery. It also is responsible for requesting the xTEDS of new devices and forwarding them on to the Data Manager.

The Sensor Manager acts as a portal to the SDM for simple devices. A device does not need to conform to the full complexity of the SDM to be used, it only needs to conform to a very simple interface to the Sensor Manager. This is especially useful when integrating legacy components.

A device with its own connection to the SDM network and able to cope with the full complexity of the SDM can have a direct connection outside of a Sensor Manager's control.

### 2.2.5 Devices

Devices, both sensors and actuators, form a very important class of SDM components. They all have an xTEDS

which describes their function. They rely on the Sensor Manager to handle most communication details. They primarily listen for any commands that they respond to and push data when appropriate. It is the Sensor Manager that reads this data and forwards it on to all interested subscribers.

SDM devices are plug-and-play. That is, any SDM device can be attached to any Sensor Manager and will function properly. This capability is critical in realizing the six-day satellite objective of the Responsive Space Testbed Initiative.

Cotterell's eBlocks [1] provide similar functionality in terms of using off-the-shelf components. eBlocks, however, have a vastly different target audience, users with no technical background, and are limited to Boolean input and output. This is unnecessarily restrictive for building a satellite. The Satellite Data Model components express their inputs and outputs in an eXtended Transducer Electronic Data Sheet (xTEDS). Also these components are not limited to pre-built hardware components, but they can also include pre-written software packages.

### 2.2.6 Applications

The final class of SDM components are applications. An application may be a data producer, in which case it also has an xTEDS which it registers with the Data Manager. It may also be a data consumer that makes requests of the rest of the system.

## 2.3 xTEDS

To support the completely generalized interface of the SDM, a rich XML-based capability description language, eXtended Transducer Electronic DataSheets (xTEDS), an extension of IEEE 1451 TEDS [3, 4], was developed. The xTEDS of an SDM component describes its input (command messages), output (data messages), and the variables that the messages are composed of.

The major differences between a TEDS and an xTEDS are in the format. xTEDS are XML documents, whereas TEDS are encoded in a binary format described by IEEE 1451.2. Each field in the xTEDS has an optional text description which aids in documentation. Another major difference is that the xTEDS is annotated with physical location information by the SDM system when a new device is discovered. xTEDS are also intended as an extensible descriptive language of capabilities.

Many attributes of the variables and messages, such as precision, can be specified. The xTEDS serves as a self-documenting mechanism for hardware and software within the SDM system. xTEDS also facilitate plug-and-play functionality. When a new component is detected, the first action of the SDM is to request the xTEDS of the

new component. The xTEDS informs the SDM system on all further interactions with the component.

Legacy equipment can be easily supported by the SDM system by adding an appropriate wrapper.

### 3 Self-Configuration

The SDM system starts out as a publish/subscribe system, in order to more easily support self-configuration. One of the great strengths of publish/subscribe as a communication model is the uncoupling of producers and consumers. This uncoupling makes it possible for control programs, sensors, and actuators to locate and identify each other with no *a priori* knowledge of a specific address.

This generic discovery process is further aided by the xTEDS of the SDM components. With xTEDS clearly describing the *capabilities* of a device, discovery queries can search for any device that is capable of some desired task. All that an SDM application programmer needs to know to write a program is what that program will require for data inputs.

The programmer does not need to know the format of the data messages that will be passed to acquire this data. Rather, the SDM provides flexibility to handle and adapt to whatever format data is presented in. Furthermore, the programmer does not need to know the name, location, or any other specifics about the data provider.

For example, if a programmer were to write a thermostat control program, they could write the program to use a generic temperature sensor, and a generic heater actuator. Thus, when the actual thermostat is built, the builder grabs any heater and any thermometer along with my pre-written control program and connects them together. The SDM system matches the program's need for a generic temperature sensor with the specific thermometer available to the system, and likewise with the heater. If, at some point, one of the hardware components fails any component with the necessary capabilities can replace it. Even if the components expect to communicate with vastly different messages, this replacement can occur because message passing is managed dynamically by the SDM system.

#### 3.1 Initialization

Initially, only the Data Manager and Task Managers exist at a known location. The Task Manager periodically sends a ready message to the location of the Data Manager. When the Data Manager comes on-line (if not already), it responds to the Task Manager. The Task Manager then becomes active.

Process Managers, one on each node, begin sending ready messages to the location of the Task Manager.

When the Task Manager becomes active, it inspects the ready message to extract the address of each Process Manager, and responds. The Process Manager knows that both the Data Manager and Task Manager are active. It sends a message containing its capabilities to the Task Manager and awaits a request to execute a task. It is critical the the Data Manager be active before running any tasks so that if any of those tasks should register an xTEDS, that the information will not be lost.

Sensor Managers follow a similar process. When they begin, they periodically send ready messages to the Data Manager. The Data Manager notes the address of the ready Sensor Manager and then responds. After the Data Manager responds, the Sensor Manager begins the device discovery process. As in the case of the Process Manager, the Sensor Manager does not become active until the Data Manager is available, thus, there is an available repository for the xTEDS that the Sensor Manager will request from discovered devices.

Figure 1 graphically describes this process. Note that only a partial ordering is imposed on the SDM system. Also note that this is an ongoing process. A new Sensor Manager or Process Manager can be added at any time to the SDM system, and its resources will begin to be utilized appropriately.

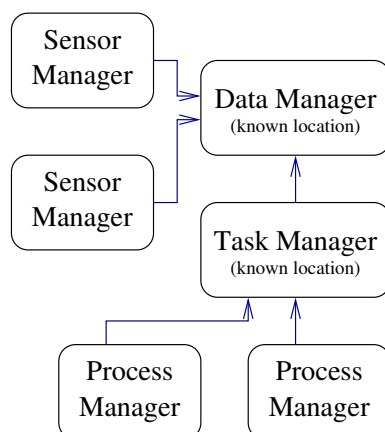


Figure 1: Initializing the SDM system.

#### 3.2 Generic Marshalling and Unmarshalling

In their xTEDS, SDM devices and applications define the format of their input and output messages. This allows any device to be made compatible with the SDM, but also places a burden on applications which must be able to interpret any such message using the description in the xTEDS.

The SDM solves this difficult problem of completely generic message passing with a combination of three tools. The first is the xTEDS document which clearly describes the expected message formats. Second is a Common Data Dictionary an ontology that gives concrete meaning to field descriptions. Third is a helper class in the SDM API that allows a programmer to refer to variables with their ontological meanings, such as setting or reading a temperature value. This class uses the information in the xTEDS message description, and the Common Data Dictionary to match the desired ontological value with a physical location and format within a message. It dynamically performs the necessary marshalling or unmarshalling operations to deliver the value in the format expected by the application.

Since this is a dynamic and generic process, one application is compatible with every device capable of generating the desired data, regardless of the format of that data.

### 3.3 Data Flow

Figure 2 shows the typical flow of data through the SDM system. The first step is the acquisition of xTEDS. If the data provider is an application, it is responsible for informing the Data Manager about its xTEDS. If, however, the data provider is a sensor, the process has a few more steps. First, the Sensor Manager detects that a new device has been connected. It next requests that the device run any initialization routines. Finally, the Sensor Manager requests the xTEDS from the device and forwards it on to the Data Manager.

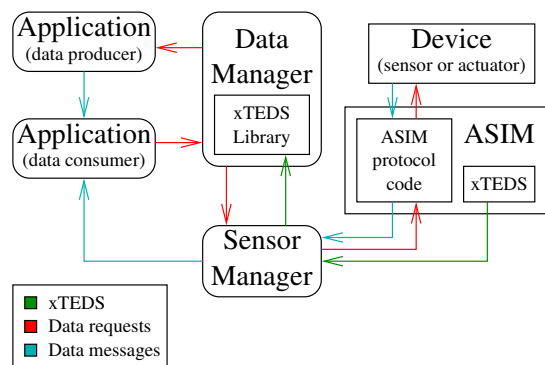


Figure 2: Flow of data within the SDM system.

When an application desires to consume data, it makes a request of the Data Manager. The Data Manager searches its xTEDS library for appropriate data sources and returns the results of its search. The application next selects which data source it wishes to use. The Data Manager forwards the request to the appropriate Sensor Manager or data producing application. The Sensor Manager

makes a request of the device to produce the desired data, and it forwards the data on directly to the consuming application.

### 3.4 Control Loops

One of the major advantages of a publish/subscribe system is the uncoupling in space between the publisher and subscriber [2]. While this is a key advantage during the discovery and self-configuration stages, it is unacceptable for the tight controls needed to fly a spacecraft.

Two couplings must be re-established. First, the control loop may need to know exactly which device it is communicating with and where that device resides. If an application is going to fire a thruster, it needs to know which way the thruster is pointed. This is solved by modifications to the xTEDS document of a device. The xTEDS on the device is location neutral so that the device can function wherever it is placed. Once the device is discovered by the Sensor Manager the Sensor Manager notes the physical location where the device was discovered. The notation is then made into an additional entry in the xTEDS of the device.

This means that an application can query for a device based not only on its capabilities but also its location.

The second coupling to be reestablished is coupling in time. There is some delay involved with all requests being routed through the Data Manager. For many applications, this is an acceptable trade-off for not having to be concerned with locations. In situations where this is not acceptable, an alternative method is available. As shown in Figure 3, an application may query the Data Manager for a component and request the location of that component. Once the application knows the address of the proper Sensor Manager (or some other application), a direct channel of communication can be opened, thereby decreasing the response time of the control loop.

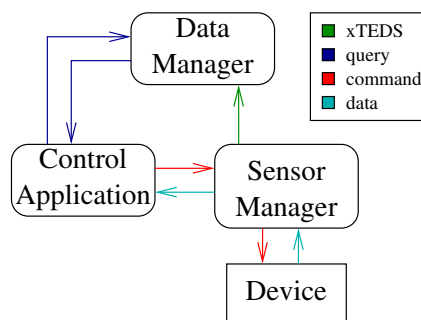


Figure 3: Forming a control loop

In these ways, the SDM system can transition gracefully from an uncoupled publish/subscribe system to a collection of tightly coupled control loops. As the orig-

inal publish/subscribe system remains intact if less used, it is always possible to transition in the other direction, as well, if needed.

## References

- [1] Susan Cotterell, Frank Vahid, Walid Najjar, and Harry Hsieh. First results with eblocks: Embedded systems building blocks. *CODES+ISSS*, October 2003.
- [2] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.
- [3] Kang Lee. A synopsis of the IEEE P1451 - Standards for Smart Transducer Communication. National Institute of Standards and Technology, 1999.
- [4] Kang Lee and Richard D. Schneeman. Distributed measurement and control based on the IEEE 1451 Smart Transducer Interface Standards. *IEEE Transactions on Instrumentation and Measurement*, 49(3):621–627, June 2000.