

# A Self Describing Data Transfer Methodology for ITS Applications

Submitted to IEEE TKDE 5/23/97

D.J. Dailey D. Meyers

## Abstract

The wide variety of remote sensors used in intelligent transportation systems (ITS) applications (loops, probe vehicles, radar, cameras, etc.) has created a need for general methods by which data can be shared among agencies and users who own disparate computer systems.

In this paper we present a methodology that demonstrates that it is possible to create, encode, and decode a self-describing data stream using: (1) existing data description language standards, (2) parsers to enforce language compliance, (3) a simple content language that flows out of the data description language, and (4) architecture neutral encoders and decoders based on ASN.1.

## Keywords

Self describing data, ASN.1, Intelligent transportation systems

## I. INTRODUCTION

The wide variety of remote sensors used in intelligent transportation systems (ITS) applications (loops, probe vehicles, radar, cameras etc.) has created a need for general methods by which data can be shared among agencies and users who own disparate computer systems. Such data sharing requires that the sender and recipient of the data agree on a method for transfer. To date, most systems constructed for this purpose either lack generality or are limited to data transfers of a specific type [1], [2], [3], or they are so general and complex as to be very difficult to implement [4]. The work presented in this paper is aimed at creating a general mechanism for *self-describing* data transfers of data streams that are produced by a set of remote sensors that change in number and type as a function of time. We present our self-describing data transfer concept in the context of Intelligent Transport Systems applications, however our approach is applicable to a variety of data types and sensors.

Self-describing data transfer requires information about the meaning of the data to be included as part of the transfer.[5] This *meta-data* must include all information needed to

This work was supported in part by a series of contracts with the Federal Highway Administration and the Washington State Department of Transportation

Author to which correspondence is to be sent, Box 352500, Department of Electrical Engineering, University of Washington, Seattle WA 98195. dailey@ee.washington.edu, (206) 543-2493

Department of Electrical Engineering, University of Washington, Seattle WA 98195

interpret the actual data stream. For example, the time-invariant properties of a remote sensor that might be relevant include its location, the units of measurement, the precision of its measurements, and so forth. In addition, a description of the algorithm used to extract the desired information from the data is required.

Any successful methodology that provides self-describing data transfers must meet the following criteria:

1. The transfer includes all information (*meta-data*) needed to interpret the data together with the data itself. If this requirement is met, the data transfer is *self-describing*.
2. The transfer method can be applied to a broad category of data types and procurement methods. This is a requirement for *data type independence* of the data transfer method.
3. The transfer method is applicable to a wide variety of computing environments. This is a requirement for *portability and general applicability* of the data transfer method.

Strictly, only the first requirement need be met to qualify the data transfer as self-describing, the effect of the other requirements is to enhance the generality of a transfer method. There are a variety of proposed data transfer mechanisms that transfer the data and meaning.[6], [7], [8], [9], [2], [3] Many of the data sharing methods used to date have involved the construction of custom software that “understands” the meaning of the data to be transferred for each class of data transfer.[3], [2] Such methods fail the second and third criteria listed above. They fail the second criterion because the transfer is specific to one type of data. They fail the third criterion unless the custom software is written in a very portable manner.

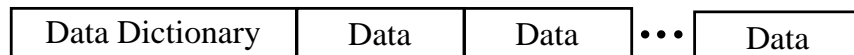


Fig. 1. Data model for self-describing data transfers.

We present a new approach to solving the self-describing data transfer problem. Our data transfer method serializes a data description in the form of a data dictionary with the actual data to be transferred. Our data description makes use of the power of database query languages to ease the task of constructing a *Data Dictionary* to contain the necessary meta-data. Database languages are well suited for the task at hand because they are designed for the description and categorization of data. This data dictionary is the initial part of a SDD transfer and the actual sensor data is serialized after the data dictionary as shown in Figure 1. A SDD transfer is composed of one data dictionary and a continuous stream

of sensor data. A SDD transfer ends when a new data dictionary is transferred. We are proposing the SDD transfer method presented here as a robust mechanism for distributing ITS data to Information Service Providers (ISP's<sup>1</sup>). We are aware of two other efforts to produce standards for communication of transportation related data. They are the National Transportation Communications for ITS Protocol (NTCIP) and the Transportation Network Profile of the Spatial Data Transfer Standard (TNP). These standards were designed to meet significantly different needs, though both are designed to solve the problem of standardized data communication. The SDD paradigm presented here is related to aspects of both NTCIP and TNP.

The first related standard is the National Transportation Communications for ITS Protocol (NTCIP). NTCIP is a family of communications protocols (A, B, C, E) developed for real time communication between a master controller and field devices such as traffic signal controllers, environmental sensor stations, dynamic message signs, highway advisory radio, closed circuit tv, and freeway ramp meters.[12] For example, the class B protocol is designed for direct communication between a master controller and one or more field devices connected by a communications cable using low-speed data transmission (on the order of 1200 baud). The target application is the control of traffic signal management systems. The standard covers both the rules of transmission and the format and meaning of a set of standardized messages to be transmitted.

NTCIP is basically an extension of the Simple Network Management Protocol (SNMP).[13] It extends the management information base hierarchical name-space defined (using ASN.1 [14]) for use with SNMP to include nodes devoted to the NTCIP. Groups of objects defined in this tree structured name-space are referred to as MIBs, and represent the set of objects (in the object oriented design sense) that are needed to effect the desired control or information transfer operations. It is intended for many types of transportation devices each with different database requirements. Control or data exchange is effected by modification of the objects in the MIB associated with the device(s) being controlled. This modification uses the get/set paradigm of the SNMP to change the values of objects in the MIB. The resulting action depends on the programming of the controlled devices.

NTCIP is designed to standardize the control of traffic management devices such as traffic

<sup>1</sup>See the ITS National Architecture study for more information on ISP's [10], [11]

light controllers. The method uses a data dictionary, the management information base (MIB), as a repository of information used to control the external devices. The response of a device to some change to its MIB is determined by software resident at the device. Under that paradigm, to request data from a sensor, a management application modifies the sensor's MIB in a way that allows the management application to obtain the sensor data.

NTCIP is a control protocol which uses a globally agreed upon set of objects. To be effective, the control/management software that uses the get/set operations needs to understand how the software within the devices reacts to changes in the MIB. So there must be an a priori agreement on software functions as well as object definitions.

From an NTCIP perspective SDD transfers can be cast as a compound object for information transfer without a priori information about the data to be shared. Thus SDD is targeted at information transfer and not control. The control function of NTCIP can be used to initiate or terminate SDD transfers. The data dictionary components and the actual data can be declared as objects and an ASN.1 compound object can be created. Changes in the MIB structure would then initiate or terminate a SDD transfer using the SMNP paradigms. The SDD transfer could be implemented either by viewing the MIB as a control that initiates an out of band data transfer as in [15] or the MIB could actually contain the components of SDD as objects. SDD transfers leverage NTCIP in that SDD has an agreed upon data description language that includes methods to describe and extract the elements of data from a data stream without the need for a great deal of a priori knowledge. For example, it is possible to create an application that would operate in a Java environment and that could obtain the methods from the data dictionary in the form of Java language elements and have those methods operate directly on the data stream, creating an automated transfer of data with only SQL and Java as the required a priori knowledge.

The second related standard is the Transportation Network Profile (TNP) of the Spatial Data Transfer Standard (FIPS 173) that is designed for use with geographic vector data that has network topology. The TNP allows transfers of spatial data that can be represented by vector objects which comprise a network or planar graph. The TNP data types are *nodes* and *links* between nodes each of which may have associated attributes. These associated attributes may be multi-valued, and therefore could be used to convey time-varying information associated with a node or a link.

The TNP provides a mechanism for defining an external data dictionary module that need not be included in each transfer, thereby reducing the amount of overhead that must be devoted to sending a dictionary module for multiple transfers that use the same dictionary.

Though it was not designed specifically for the purpose of transferring large amounts of time-varying data, the SDTS/TNP could be used to transfer time-varying data from a set of remote sensors by representing the time-invariant information about the sensors (location, sensor type, etc) as single valued attributes of the nodes that make up the network and the time-varying data would be represented as multi-valued attributes functionally dependent on time. If the time dependent attributes are isolated into a separate table from the time invariant information it would be possible to send the module containing time-invariant data first, and continue with the “open-ended” time-varying module until no more time-varying data was desired at the receiving end. Though possible, transfer of large amounts of time-varying data using the SDTS/TNP has two disadvantages: (1) the data stream must be interpreted before transfer and placed into the appropriate attribute tables, possibly at a significant cost in required bandwidth and (2) the overall design of SDTS/TNP does not really include the notion of a transfer of indeterminate length. Our method makes a clear distinction between time-invariant data (the data dictionary contents) and time-varying data (the actual data stream), and allows for a more efficient treatment of the actual data stream.

## II. DATA MODEL

In the work presented here the data to be transferred is modeled as having two components: (1) the data dictionary, and (2) the actual sensor data. These two components, transferred serially, effect a SDD transfer. The Data Dictionary component is central to our self-describing data transfer method. In our model the Data Dictionary is comprised of two parts: (1) Dictionary Schema and (2) Dictionary Contents. These two parts provide the necessary description of the data to make it useful to a client, and are described in the next sections.

### A. Dictionary Schema

The first part of our Data Dictionary is the *Dictionary Schema*. This is meta data that specifies the schema of the data description (e.g. a sensor has a name, position, and units

of measure and the position is specified in latitude and longitude). The dictionary schema is a provider-defined database schema written in a subset of Entry Level SQL-92. [16] We chose Entry Level SQL-92 because it is fully relational and the relational model can represent an arbitrary set of data.[17] The SDD model presented here allows for the construction of any schema that SQL allows, and this guarantees a powerful data description language. In the Dictionary Schema, the data provider should include sufficient information about the actual data to allow a recipient to interpret that data. Because the sufficiency of the data dictionary is dependent on its author, it is clear that the data dictionary concept *allows* for self-describing data transfer, but does not *ensure* that any given data transfer is in fact self-describing. It would seem difficult, if not impossible to make such an assurance in an automated system. It is, however, possible to automate the verification that the schema provided conforms to the data description language.

As part of the SDD transfer method we have created a schema parser which is used to verify the data dictionary schema definition. The language accepted by the parser is a subset of entry level SQL-92 that allows definition of schemas, tables, etc. but does not include any of the query processing facilities of a complete database language. Our intent in defining the schema language is to provide sufficient power for the definition of a data dictionary while simultaneously making the language simple enough that it is easy to learn.

The schema parser is used in two ways by our self-describing data transfer protocol. First, the sending application uses the parser to verify that a user-provided schema definition is valid. The second use of the parser is the construction of a parse tree that is subsequently used to verify that the dictionary contents are compatible with the defined dictionary schema. The receiving application uses the parser again to verify that the received dictionary schema is a valid one, and constructs a parse tree that is used to create and verify dictionary contents file.

The parse tree is a memory-resident data structure that is constructed from the schema definition. The tree is implemented as a “leftmost child, right sibling” data structure in which the descendants of a node are represented as a linked list of nodes ordered from leftmost child to rightmost child. A node has two associated linked lists: a siblings list and a descendants list. Figure 2 shows an example parse tree and the list structure used to instantiate that tree.

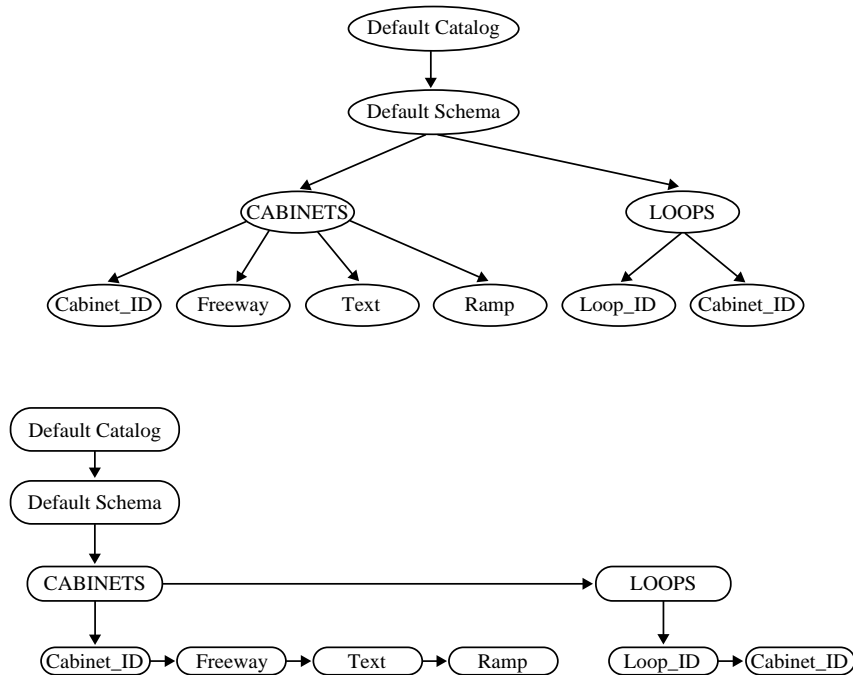


Fig. 2. Parse tree structure at top and its leftmost child, right sibling representation at bottom.

The parse tree is used during schema verification to facilitate certain semantic checks that must be performed. For example, no two tables may have the same name within the same schema, and no two columns within the same table may have the same name. When a name is encountered, it is inserted into the parse tree, and checked for uniqueness at that time. An additional check that must be performed is to ensure that the columns named in a foreign key reference are compatible between the referencing and referenced tables. The parse tree contains sufficient information to make such checks, and is organized so that the checks can be performed efficiently.

The structure of the schema language is such that the maximum depth of the parse tree is four. Further, at each level, the nodes are of a specific type. The four types, in order of increasing depth in the parse tree are catalog, schema, table, and column. For entry level SQL-92, the catalog and schema nodes are not really needed, since only one of each can exist. A single schema node could be used as the root of the parse tree. We chose the 4-level tree implementation make it easier to extend to higher levels of SQL-92 conformance (which allows multiple schemas and catalogs).

### B. Dictionary Contents

The second part of the data dictionary is the *Dictionary Contents*. The dictionary contents is the information about the actual data stream that can be used to construct a database describing the static information about the sensors for this data transfer. This is the component that contains particular values for the description of each sensor (e.g. sensor one is at 122.23 deg longitude, 47.21 deg latitude and measures rainfall in inches). We define a *Contents Language* and associated parser to facilitate verification that the schema contents are compatible with the schema into which they are to be placed. A contents parser recognizes the language used to describe the contents of the data dictionary. The language is designed to allow specification of the table and columns into which a set of data tuples are to be inserted.

The contents language is fairly simple. A *data dictionary file* consists of a series of *table entries*. Each table entry is of the following form:

```
TABLE <table name>
COLUMN (<column name 1>, ... , <column name n>)
<data for column name 1>, ... , <data for column name n> ;
... one tuple for each row to be inserted into the table
<data for column name 1>, ... , <data for column name n> ;
```

The parser ensures this format, and ensures that the type of data supplied in each tuple matches the data type declared for its corresponding column.

On the sending end of a data transfer, the contents parser is used to verify that the contents file supplied by a data provider is compatible with the dictionary schema in use. On the receiving end, the contents parser is used to verify that the data in the contents file are compatible with the schema during SQL command generation.

### C. Data Transfer

The overall architecture of our system for self-describing data transfer is shown in Figure 3. This structure is independent of the actual data stream involved. The data transfer is a serialized stream divided into frames. The first frame of a transfer contains the Dictionary Schema. The second frame contains the Dictionary Contents, subsequent frames contain

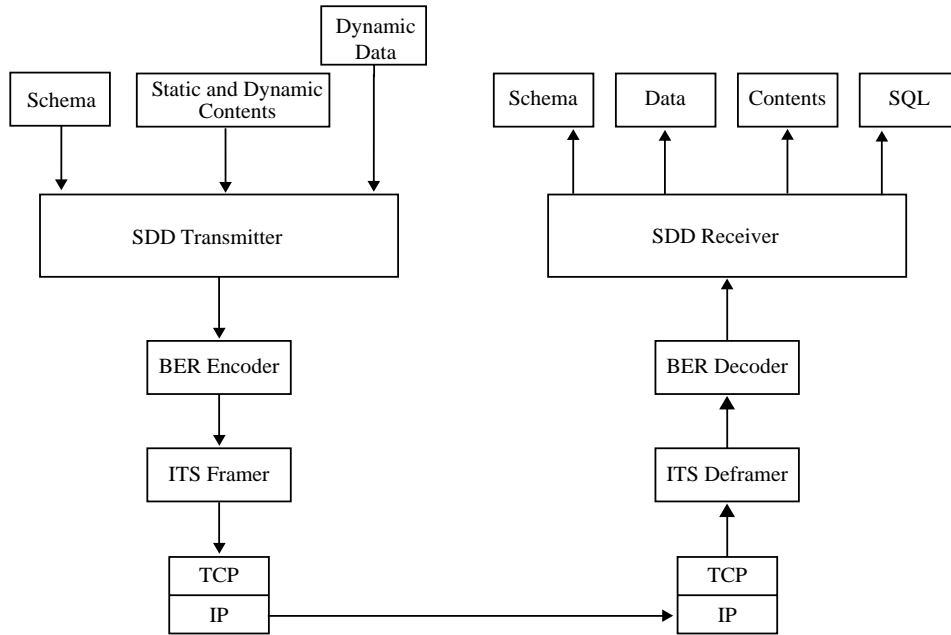


Fig. 3. An overview of the structure of our system for self-describing data transfers.

the most recently available set of data from the data source. The implementation of a self-describing data transfer takes the form of a transmitter and a receiver.

#### D. Transmitter

Figure 4 shows the components necessary to construct and transmit a stream of self describing data. The SDD transmitter verifies the compliance of the schema using the schema parser, which creates a parse tree as a byproduct of the compliance check. This parse tree is then used by the contents parser to verify that the contents complies with the schema. If both the schema the contents are in compliance they are serialized in the order schema, contents, and the actual sensor data. Transfer of the dictionary and data between computer systems is accomplished by encoding the data according to the Basic Encoding Rules (BER) defined for the ASN.1 standard. As a block of data is received, it is encoded and sent to all clients using the redistributor methodology detailed in [15]. In our application, the data is a stream of bytes, and the information is extracted using algorithms specified in the Data Dictionary. This mechanism allows for a very general transfer that is easy to encode, but which requires programming effort to be devoted to data extraction by the transfer recipient.

BER encoding of the self-describing data stream involves three types: data, schema, and contents. Each of these types is encoded according to the BER standard (ISO 8825-1)[18], with a type, length, and value.

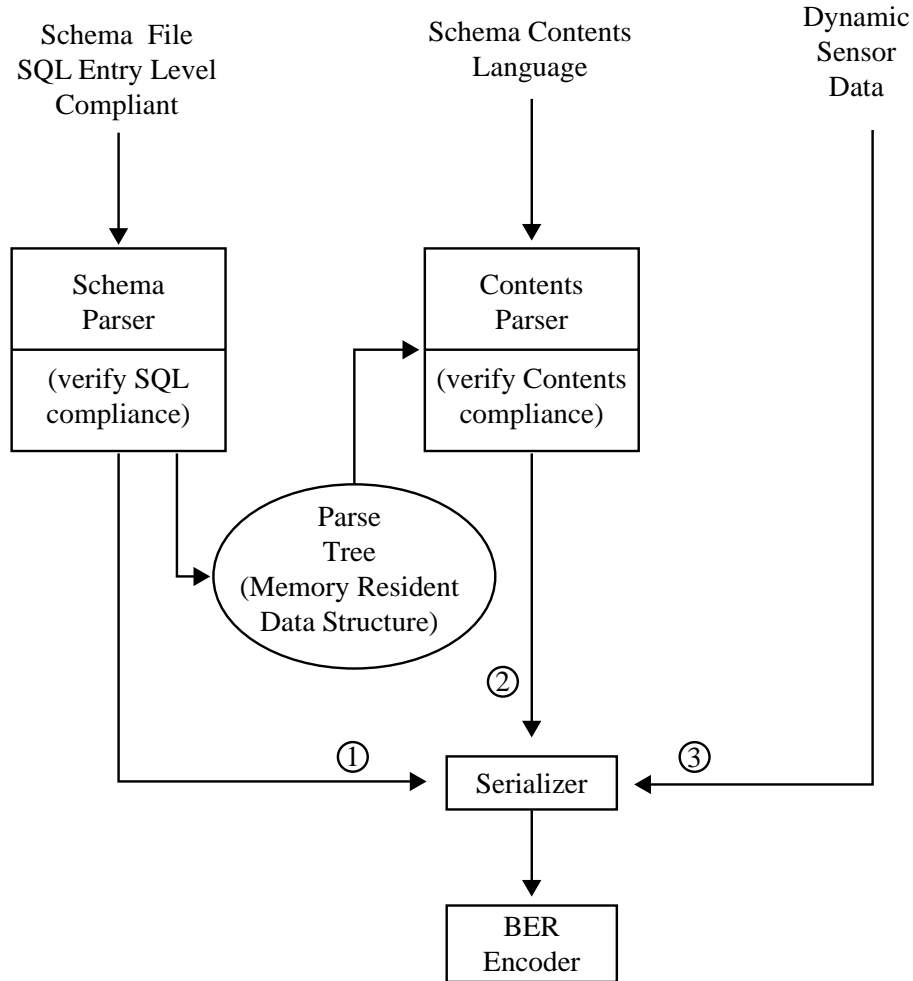


Fig. 4. Self-describing data transmitter.

We use the ASN.1 “Application” class with the “primitive” encoding, using tag numbers 1, 2, and 3 to denote schema, contents and data, respectively. Our identifiers are therefore encodable in one octet. We encode the “schema” and “contents” types as IA5 strings, and the “data” type is unchanged during encoding. An ASN.1 type declaration of our types is:

```

DictionarySchema ::= [APPLICATION 1] IA5String
DictionaryContents ::= [APPLICATION 2] IA5String
Data ::= [APPLICATION 3] OCTET STRING

```

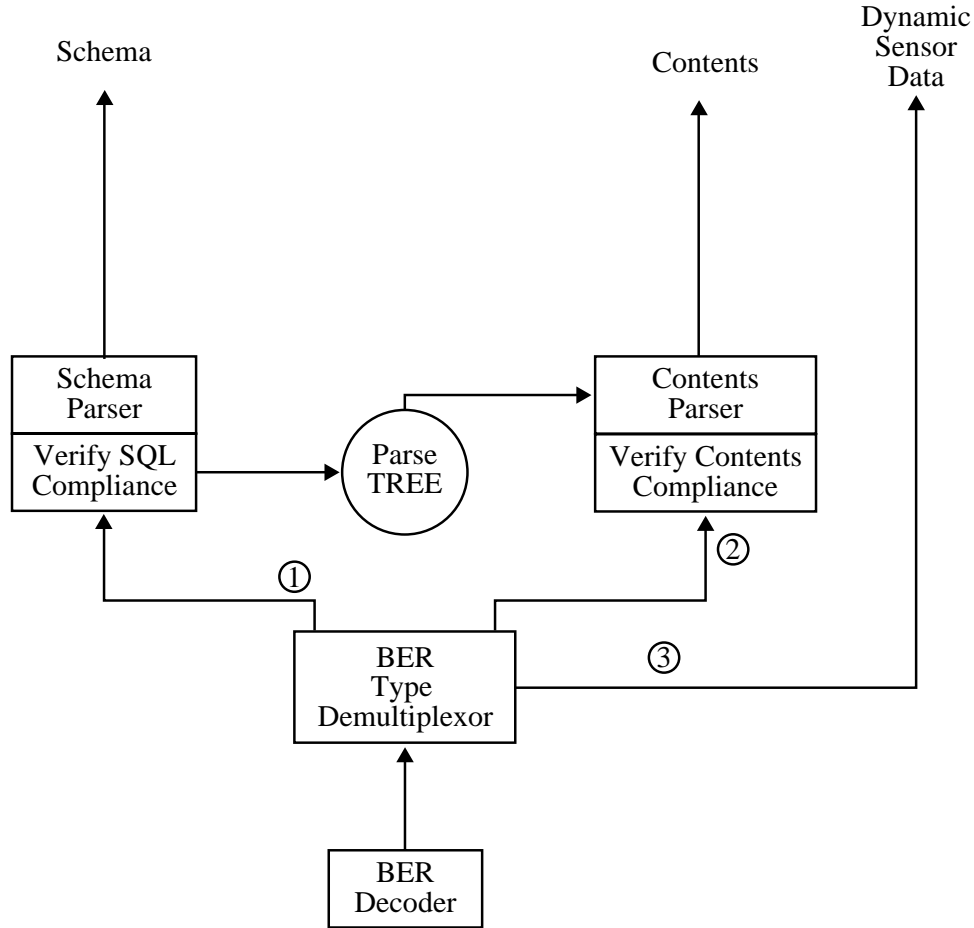


Fig. 5. The detailed structure of the SDD Receiver application.

The tag number needs to be unique across the family of protocols using these three data types. For example, if used with the NTCIP family of protocols the NTCIP standards document would need to dedicate three types to a SDD transfer facility.

The serializer in figure 4 sends a type, length, and value to the BER Encoder. The BER Encoder encodes these values as the appropriate header, length bytes and contents bytes according to the ASN.1 definition above.

### E. SDD Receiver

The SDD Receiver is a client application that converts from the data transfer stream format back to three data sets: schema, contents, and data. The structure of the receiver, as shown in Figure 5, is parallel to that of the transmitter in Figure 4. The BER decoder takes an SDD data stream as an input, BER decodes the data stream, and provides a

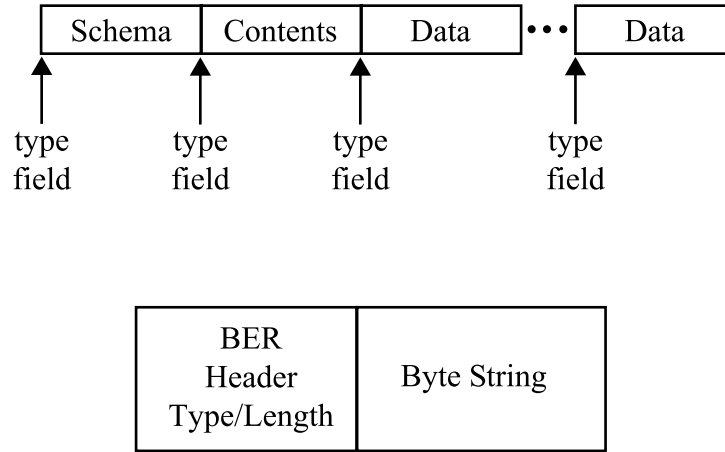


Fig. 6. Structure of our serialized data stream for self-describing data transfers.

decoded serial data stream that has the structure described in Figure 6. The BER Type Demultiplexor receives a serial data stream from the BER Decoder and it uses the BER type field to distribute the schema and contents to the appropriate parser. The parser components of the receiver are identical to those of the transmitter. The schema component is sent to a schema parser that verifies SQL-92 compliance and creates a parse tree for use by the contents parser. The contents are sent to the contents parser which verifies the contents against the schema. The outputs of these two parsers are the verified schema and contents used to describe the dynamic sensor data. With the arrival of the data dictionary schema, data dictionary contents, and the sensor data a self-describing data transfer is complete.

In our implementation we have added an SQL generator as shown at the top of Figure 5. The SQL Generator creates a series of SQL INSERT commands from the dictionary schema and dictionary contents. Each data tuple in the contents file will be represented by an INSERT statement in the SQL output file. The commands, when used as input to an SQL database engine, will create and fill a database that instantiates the data dictionary. We include this step as a practical matter for the engineering users of the SDD paradigm, so that the result of a SDD transfer is a data base with which an engineer can interact and a binary stream of dynamic sensor data. The practicality of this paradigm is demonstrated using an ITS example in the next section.

### III. ITS EXAMPLE

The Washington State Department of Transportation (WSDOT) operates a Traffic Management System (TMS) to collect data from traffic sensors in the central Puget Sound region, and this TMS traffic data is the focus for this ITS application of SDD. The data from the TMS consists of three parts, there are (1) sensor data which is binary data representing measures of traffic conditions (e.g. speed, flow, occupancy), (2) information about the location and type of each sensor (e.g. mile marker, latitude, longitude, calibration), and (3) lists of presently available sensors. The sensor data comes from inductance loop detectors placed at  $\approx 3000$  locations in the Seattle metropolitan area, and the list of sensors/detectors presently available changes slowly with time (e.g. every few hours). This dynamic list of available sensors makes up the dynamic component of the data dictionary contents. The name, location, measurement type, etc. make up the static component of the data dictionary contents.

Figure 7 provides details about our “tms2sdd” application, which converts legacy TMS data to our self-describing transfer format. The application can be divided into a “legacy” component and a “standard” component. The legacy component is dependent on the specific data source, the standard component does not change from one source to another. As in the generic transmitter case the self-describing dictionary contents file is verified against the dictionary schema by the standard component of tms2sdd. The schema is verified by the Schema Parser, which constructs a memory-resident parse tree that is used by the Contents Parser during the process of parsing and verifying the dictionary contents file. If verification of both files succeeds, they are transmitted to the BER Encoder.

Operationally, when the TMS application is started a block of meta-data is retrieved from the TMS as an ITS Frame (as shown at the top of Figure 7). The meta-data block received from the TMS Data Stream represents that part of the dictionary contents that is subject to relatively frequent change. To construct a complete dictionary contents file, the tms2ddc function combines the contents of a Static Data File(s) with the information in the meta-data block to construct a complete Dictionary Contents File. The Data Dictionary is then transmitted in two parts: the Dictionary Schema, and the Dictionary Contents.

In this example the data dictionary embodies the notion of state, in that the sensor data stream has an unambiguous interpretation once the data dictionary is present and is

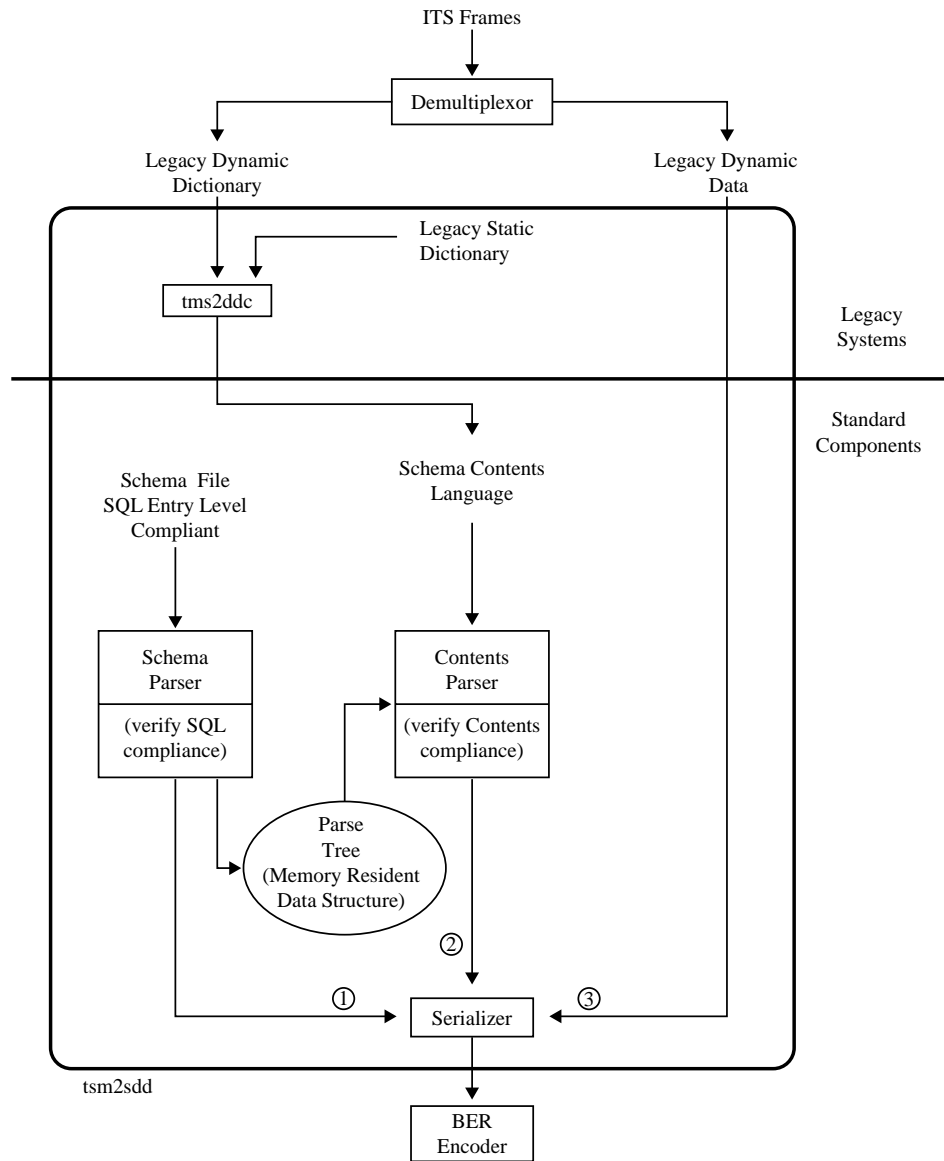


Fig. 7. An overview of the structure of our example application for self-describing data transfers.

meaningless otherwise. Whenever a block of meta-data is received from the TMS Data Source indicating that a change in the number, type, or availability of loop detectors has taken place a new data dictionary is created. The transmission of this new Data Dictionary signals the end of one SDD transfer and the beginning of another (or in other words, a change of state). Transfer of the Data Dictionary occurs when a client first connects, and subsequently whenever a block of meta-data is received in the TMS data stream. Once the Data Dictionary has been sent to a client, the actual data is transmitted as it becomes

available (in our application, a new block of sensor data arrives every twenty seconds).

The SDD methodology described here is being deployed for ITS traveler information applications in the Seattle metropolitan area. Information service providers can connect to the regional ITS backbone to obtain

#### IV. CONCLUSIONS

In this paper we present a methodology that provides a framework to create, encode, and decode a self-describing data stream. Clients of these data streams can interpret the detailed information in the data transfer with limited a priori information. We demonstrate that a self describing data transfer be completed using only:

1. existing data description language standards,
2. parsers to enforce language compliance,
3. a simple content language that flows out of the data description language,
4. architecture neutral encoders and decoders based on ASN.1.

We demonstrate the use of the SDD paradigm with data from a legacy traffic management center. This SDD paradigm has the potential to enhance the NTCIP family of protocols under development to support ITS deployment.

#### REFERENCES

- [1] Sydney R. Hall, "The star file: A new format for electronic data transfer and archiving," *J. Chem. Inf. Comput. Sci.*, vol. 31, no. 2, pp. 326–333, 1991.
- [2] Graham Wideman, "Streamlining experiment data manipulation with psychology experiment data interchange format (pxdif)," *Behavior Research Methods, Instruments, and Computers*, vol. 23, no. 2, pp. 288–291, 1991.
- [3] Frank H. Allen, John M. Barnard, Anthony P. F. Cook, and Sydney R. Hall, "The molecular information file (mif): Core specifications of a new standard format for chemical data," *J. Chem. Inf. Comput. Sci.*, vol. 35, no. 3, pp. 412–427, 1995.
- [4] NTIS, "Spatial data transfer standard: Fips 173-1," National Technical Information Service (NTIS) Computer Products Office 5285 Port Royal Road Springfield, VA 22161 703-487-4650 Specify FIPSPUB 173-1, parts 1 through 4 when ordering., June 1994.
- [5] Leo Mark and Nick Roussopoulos, "Information interchange between self-describing databases," *Information Systems*, vol. 15, no. 4, pp. 393–400, 1990.
- [6] Sydney R. Hall, Frank H. Allen, and I. David Brown, "The crystallographic information file (cif): a new standard archive file for crystallography," *Acta Cryst.*, vol. A47, pp. 655–685, 1991.
- [7] Sydney R. Hall and Anthony P. F. Cook, "Star dictionary definition language: Initial specification," *J. Chem. Inf. Comput. Sci.*, vol. 35, no. 5, pp. 819–825, 1995.
- [8] John A. Johnson and Frederic C. Billingsley, "A standard method for creating self-defining data structures for information archive and transfer," in *Digest of Papers. Tenth IEEE Symposium on Mass Storage Systems. Crisis in Mass Storage*, Washington, DC, USA., May 1990, IEEE, pp. 26–32, IEEE Comput. Soc. Press.
- [9] J.A. Johnson and F.C. Billingsley, "A standard method for creating self-defining data structures for information archive and transfer.," in *Digest of Papers. Tenth IEEE Symposium on Mass Storage Systems. Crisis in Mass Storage (Cat. No. 90CH2844-9)*., K.D. Friedman and B.T. Olear, Eds., Monterey, CA, USA, May 1990, pp. 26–32, IEEE Comput. Soc. Press, Washington, DC, USA.
- [10] Rockwell International Joint Architecture Team, Loral Federal Systems, "Theory of operations," *ITS Architecture*, October 1995.

- [11] Rockwell International Joint Architecture Team, Loral Federal Systems, "Physical architecture," *ITS Architecture*, October 1995.
- [12] NTCIP Steering Group, "National transportation communications for its protocol (ntcip) a family of protocols," 1996.
- [13] William Stallings, *SNMP SNMP-2 and RMON: Practical Network Management, 2nd Ed.*, Addison-Wesley Publishing Company, 1996.
- [14] ISO, "Iso/iec 8824-1 information technology – abstract syntax notation one (asn.1): Specification of basic notation.," ISO/IEC Copyright Office, Case postale 56, CH-1211 Geneva 20, Switzerland, 1995.
- [15] D.J. Dailey, M.P.Haselkorn, and D. Meyers, "A structured approach to developing real-time, distributed network applications for its deployment," *ITS Journal*, vol. 3, no. 3, 1996.
- [16] ANSI, "American national standard database language sql, ansi x3.135-1992," American National Standards Institute, 1992.
- [17] E.F. Codd, "A relational model of data for large shared data banks," *Communciations of the ACMe*, vol. 13, no. 6, June 1970.
- [18] ISO, "Iso/iec 8825-1 information technology – asn.1 encoding rules: Specification of basic encoding rules (ber), canonical encoding rules (cer) and distinguished encoding rules (der)," ISO/IEC Copyright Office, Case postale 56, CH-1211 Geneva 20, Switzerland, 1995.