

A Self-Describing Data Transfer Model for ITS Applications

Daniel J. Dailey, *Senior Member, IEEE*, Stuart Maclean, Fritz W. Cathey, and 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.

Index Terms—ASN-1, intelligent transportation systems, self-describing data.

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]–[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 (SDD) 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 SDD transfer concept in the context of ITS applications; however, our approach is applicable to a variety of data types and sensors.

SDD 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 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, and the precision of its measurements. In addition, a description of the algorithm used to extract the desired information from the data is required.

Any successful methodology that provides SDD transfers must meet the following criteria:

Manuscript received February 14, 2000; revised November 1, 2002. This work was supported in part by a series of contracts with the Federal Highway Administration and the Washington State Department of Transportation. The Associate Editor for this paper was H. Takahashi.

The authors are with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: dailey@ee.washington.edu).

Digital Object Identifier 10.1109/TITS.2002.806805

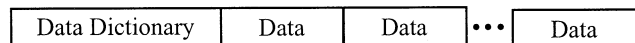


Fig. 1. Data model for SDD transfers.

- 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 the data type to be independent 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]–[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.

We present a new approach to solving the SDD 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 an SDD transfer, and the actual sensor data is serialized after the data dictionary as shown in Fig. 1. An SDD transfer is composed of one data dictionary and a continuous stream of sensor data. An SDD transfer ends and a new one begins 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).¹

¹See the ITS National Architecture study for more information on ISPs [10], [11]

A. Related Activities

We are aware of three other directions being pursued for communication of transportation related data. They are: 1) the National Transportation Communications for ITS Protocol (NTCIP) Corba and Datex activity [12]; 2) the Transportation Network Profile of the Spatial Data Transfer Standard (TNP) [13]; and 3) a new NTCIP Extensible Markup Language (XML) effort [14]. These standards were designed to meet a variety of needs and are designed to solve the problem of standardized data communication. The SDD paradigm presented here is related to aspects of each of these activities.

B. NTCIP

The first related activity is the National Transportation Communications for ITS Protocol (NTCIP). NTCIP is a family of communications protocols developed to perform two fundamentally different tasks [12].

The first task is communication between a management/control center and a sensor in the field. This portion of NTCIP extends ideas in the Simple Network Management Protocol (SNMP) [15]. It extends the management information base hierarchical name-space defined (using ASN.1 [16]) 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 Management Information Bases (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. Control or data exchange is effected by modification of the MIB. 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. The work presented here is not related to this aspect of the data communication. In the concept presented here, the management task and the data communication task are separate. The sensor is viewed as continuously producing a stream of data that needs to be communicated to external consumers, and the SDD concept presented here facilitates this activity. The management activity, such as starting or stopping the sensor, is assumed to be accomplished external to the data communication task.

The second task that NTCIP undertakes is the communication between management/control centers. This task is more directly related to the work presented here. The NTCIP efforts have taken two routes to accomplish this communications task. The first is an object oriented approach that uses a Common Object Request Broker Architecture (CORBA) [17] to provide multiple language support for creating applications that can send messages to a control center system to request data or perform control functions. The standardized message set defines the relationship between the data items that can be created [18]. The second approach is a modification of a European request/response messaging system named DATEX [19] that is used for transportation data. The developing standard provides the definition and the codification of the information to be exchanged,

analogous to the data dictionary, but using Electronic Data Interchange for Administration, Commerce and Transport (EDI-FACT) [19] for the messages.

The data transfer model presented here differs from the NTCIP efforts in that there is only one request at the beginning of the transfer that begins a continuous transfer. This design was driven by two issues: 1) server side security and 2) server side expense. These two issues are important because the agency that is serving the data may not be politically or fiscally associated with the agency making the data request. Security is an issue in any protocol that establishes a continued two-way communication between client and server. There are many examples of network attacks that compromise these types of servers [20]. In SDD there is no expectation of incoming communication from the clients and so there is no opportunity for the clients to attempt to send messages that can compromise the security of the server. Server side expense is an issue if the client can request some service in addition to the basic data flow. For example, if a client can request a specific sensor from an existing set of sensors, the server must perform a filtering function on the data stream and in addition must maintain state information on the client and the requested services. This can be costly for a large number of clients. The SDD data communications model described here does not propose to perform any actions on the data flow. In our model, the data flows through a series of components [21] and is transformed by the components so that the communication task and protocol are independent of the data flow. It is possible to create a component that would perform a request/response interaction with clients but that is outside the scope of defining an SDD flow.

Another difference between the NTCIP activity and the work presented here is the ability to represent arbitrary data sets and to use data sets that are not part of the established data dictionary standard. In our work, the data dictionary carries sufficient information to use the data without a priori information shared between the data server and client as is required by the NTCIP model.

C. TNP

The second related standard is the Transportation Network Profile (TNP) of the Spatial Data Transfer Standard (SDTS) that is designed for use with geographic vector data that has network topology [4]. 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 multivalued 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 multivalued 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.

D. XML

The third related activity is the use of XML [22] as an SDD language to support ITS. This is being considered in both the U.S. NTCIP forum and the European TRansport Intermodality Data sharing and Exchange NeTwork (TRIDENT) project [19]. XML is a markup language. An XML file is transported, using any of a variety of network models, between the client and the server and is then parsed when the closing token is received. The model for SDD data transfers presented here is a continuous stream model that cannot be duplicated using XML; however, it is possible to transport XML in the SDD framework described here. Further, the semantics of the data description inherent in SQL is not presently available in XML; however, at the time of writing there is an XML Schemas Working Group [23] considering if and how to include these semantics.

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 an SDD transfer. The Data Dictionary component is central to our SDD 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. Data Dictionary

1) *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). The dictionary schema is a provider-defined database schema written in a subset of Entry Level SQL-92 [24]. We chose Entry Level SQL-92 because it is fully relational, and the relational model can represent an arbitrary set of data [25]. The SDD model presented here allows for

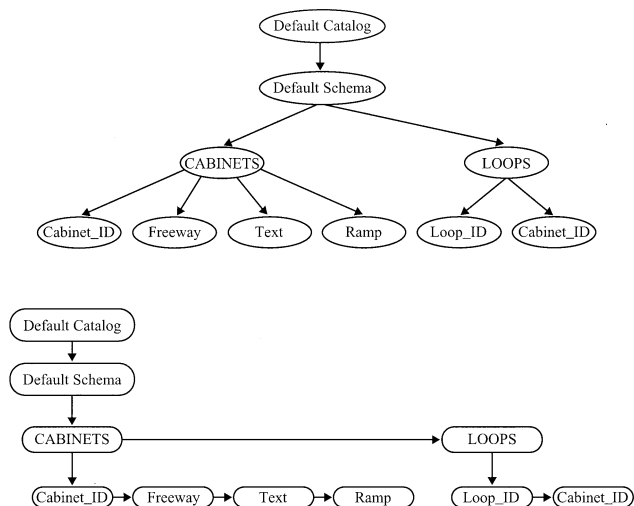


Fig. 2. Parse tree structure at top and its leftmost child, right sibling representation at bottom.

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 SDD 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 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 so that it is easy to learn.

The schema parser is used in two ways by our SDD transfer protocol. First, the sending application uses the parser to verify that a user-provided schema definition is valid. The second use 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 the 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. Fig. 2 shows an example parse tree and the list structure used to instantiate that tree.

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 to make it easier to extend to higher levels of SQL-92 conformance (which allows multiple schemas and catalogs).

2) *Dictionary Contents*: The second part of the data dictionary is the dictionary contents. The dictionary contents is the information about the actual sensors in this data transfer that can be used to construct a database describing either: 1) the slowly changing dynamic contents or 2) the unchanging static contents. This is the component that contains particular values describing attributes 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 enforces 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.

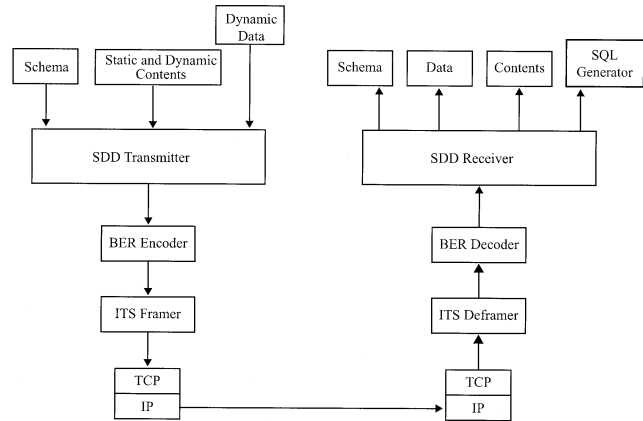


Fig. 3. An overview of the structure of our system for SDD transfers.

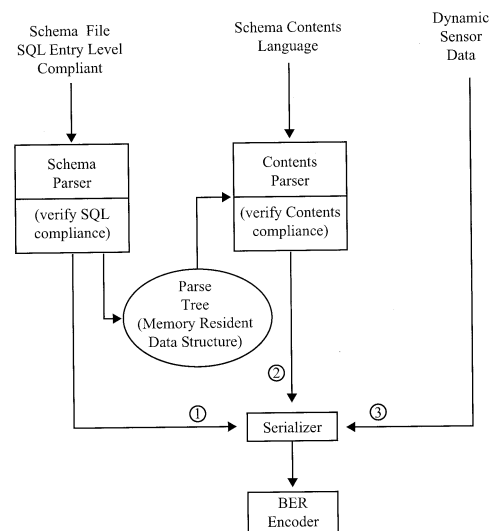


Fig. 4. SDD transmitter.

B. Data Transfer

The overall architecture of our system for SDD transfer is shown in Fig. 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 the most recently available set of data from the data source. The implementation of an SDD transfer takes the form of a transmitter and a receiver.

C. Transmitter

Fig. 4 shows the components necessary to construct and transmit a stream of SDD. 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 comply with the schema. If both the schema and the contents are in compliance, they are serialized in the order of 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 (BERs) 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 [21]. In our example 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, and requires no effort by the transfer recipient.

BER encoding of the SDD stream involves three types: data, schema, and contents. Each of these types is encoded according to the BER standard (ISO 8825-1) [26] with a type, length, and value.

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
```

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 an SDD transfer facility.

The serializer in Fig. 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.

D. 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 Fig. 5, is parallel to that of the transmitter in Fig. 4. The BER decoder takes an SDD data stream as an input, BER decodes the data stream and provides a decoded serial data stream that has the structure described in Fig. 7. The BER Type Demultiplexer receives a serial data stream from the BER Decoder and 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 which verifies contents against the schema. The outputs of these two parsers are the verified schema and contents used to describe the dynamic sensor data.

In our implementation, we have added an SQL generator as shown at the top of Fig. 3. The SQL generator creates a series of SQL INSERT commands from the 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,

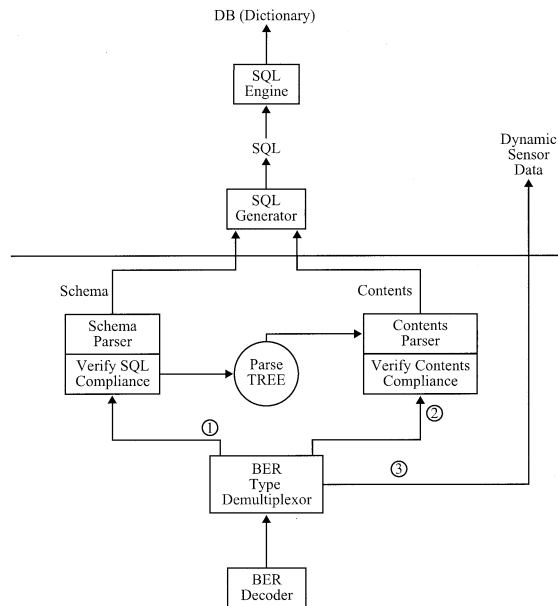


Fig. 5. The detailed structure of the SDD receiver application.

so that the result of an SDD transfer is a data base which an engineer can interact with and a binary stream of dynamic sensor data.

Fig. 6 presents a complete transmitter/receiver pair that implements an SDD transfer. The practicality of this paradigm is demonstrated in the next section using an ITS example.

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 these TMS traffic data are the focus for this ITS application of SDD. The data from the TMS consists of three parts: 1) sensor data, which are in binary form 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 come from inductance loop detectors placed at ≈ 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 slowly varying component of the data dictionary contents.

Fig. 8 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

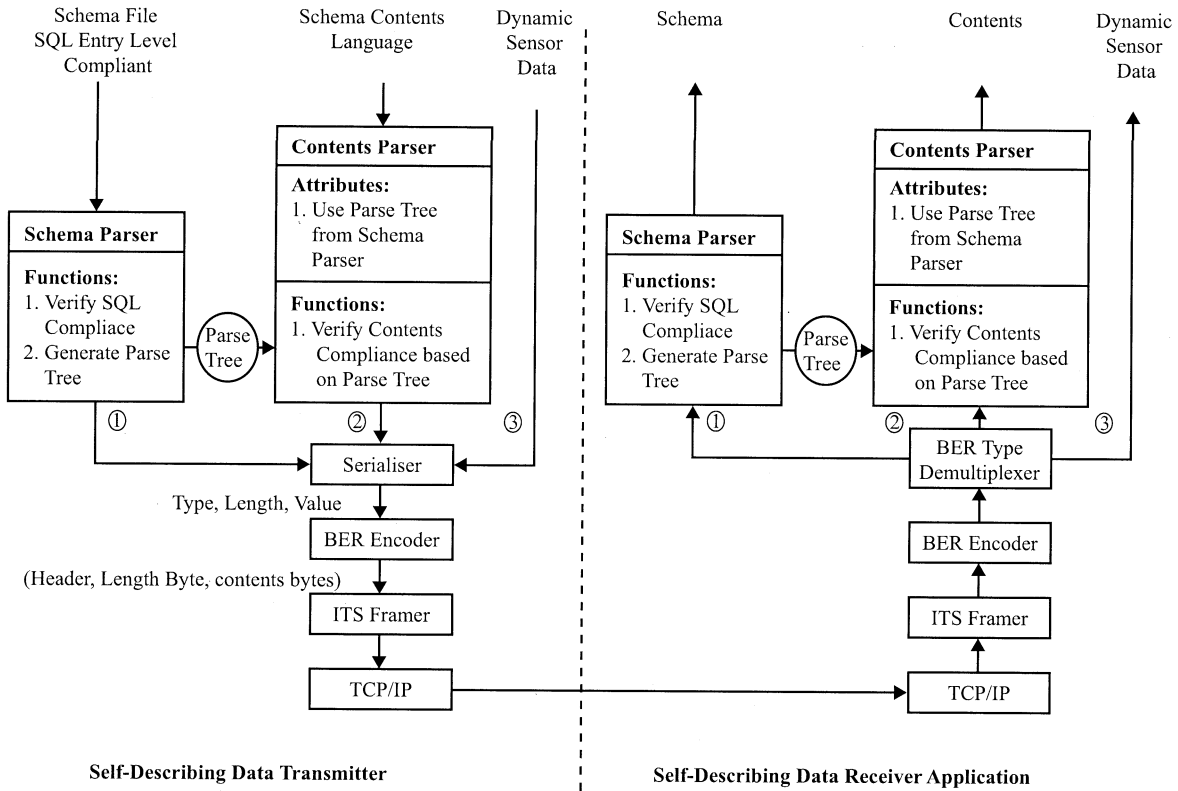


Fig. 6. Complete transmitter and receiver pair.

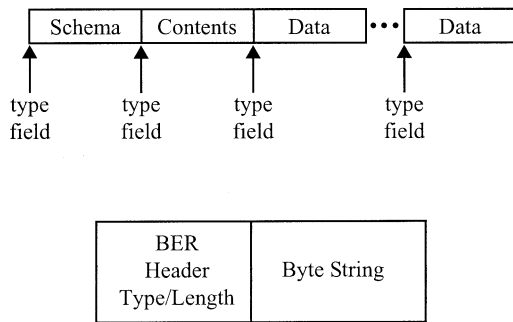


Fig. 7. Structure of our serialized data stream for SDD transfers.

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 shown at the top of Fig. 8. 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 tms2SDD 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 meaningless

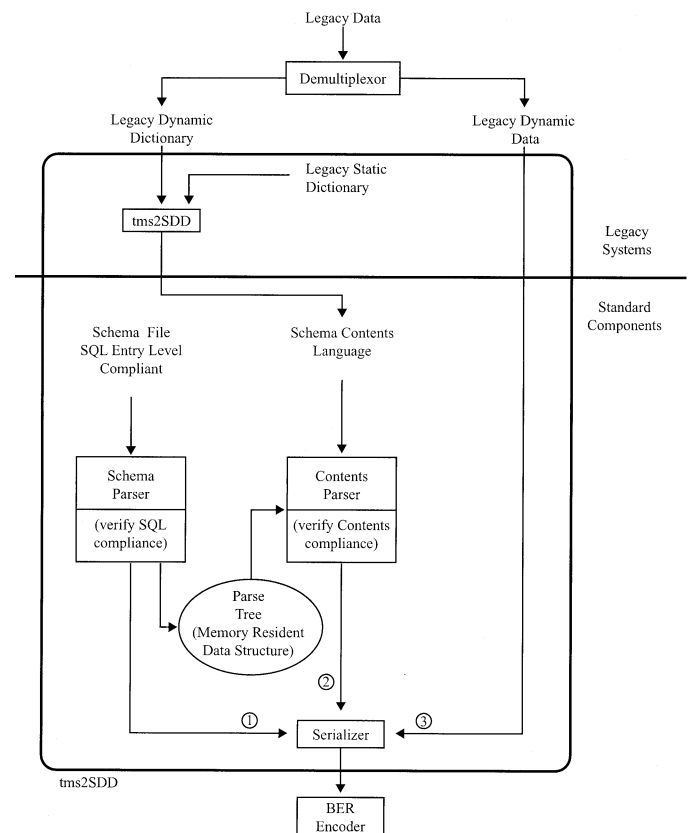


Fig. 8. Overview of the structure of our example for SDD transfer.

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 are transmitted as they become available (in our application, a new block of sensor data arrives every 20 s).

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 a variety of data types including inductance loop data, transit vehicle data, parking data, arterial data, and incident data. Further, an object oriented toolkit that implements the SDD protocol in a series of components [21] is freely available to make building ITS applications both easier and consistent across regional traffic management schemes. (See <http://www.its.washington.edu/bbone>)

IV. CONCLUSIONS

In this paper, we present a methodology that provides a framework to create, encode, and decode an SDD stream. Clients of these data streams can interpret the detailed information in the data transfer with limited *a priori* information. We demonstrate that an SDD transfer can 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 ability to build applications that support ITS deployment across regions with disparate management data and practices.

REFERENCES

- [1] S. R. Hall, "The star file: A new format for electronic data transfer and archiving," *J. Chem. Inform. Comput. Sci.*, vol. 31, no. 2, pp. 326–333, 1991.
- [2] G. Wideman, "Streamlining experiment data manipulation with psychology experiment data interchange format (pxdif)," *Behav. Res. Methods, Instrum., Comput.*, vol. 23, no. 2, pp. 288–291, 1991.
- [3] F. H. Allen, J. M. Barnard, A. P. F. Cook, and S. R. Hall, "The molecular information file (mif): Core specifications of a new standard format for chemical data," *J. Chem. Inform. Comput. Sci.*, vol. 35, no. 3, pp. 412–427, 1995.
- [4] Computer Products Office 5285 Port Royal Road NTIS, Spatial data transfer standard: Fips 173-1, National Technical Information Service (NTIS), Springfield, VA, June 1994.
- [5] L. Mark and N. Roussopoulos, "Information interchange between self-describing databases," *Inform. Syst.*, vol. 15, no. 4, pp. 393–400, 1990.
- [6] S. R. Hall, F. H. Allen, and I. D. Brown, "The crystallographic information file (cif): a new standard archive file for crystallography," *Acta Cryst.*, vol. A47, pp. 655–685, 1991.
- [7] S. R. Hall and A. P. F. Cook, "Star dictionary definition language: Initial specification," *J. Chem. Inform. Comput. Sci.*, vol. 35, no. 5, pp. 819–825, 1995.

- [8] J. A. Johnson and F. C. Billingsley, "A standard method for creating self-defining data structures for information archive and transfer," in *Dig. Papers. 10th IEEE Symp. on Mass Storage Systems. Crisis in Mass Storage*, May 1990, pp. 26–32.
- [9] —, "A standard method for creating self-defining data structures for information archive and transfer," in *Dig. Papers. 10th IEEE Symp. on Mass Storage Systems. Crisis in Mass Storage*, K. D. Friedman and B. T. Olear, Eds., May 1990, pp. 26–32.
- [10] Rockwell International Joint Architecture Team, Loral Federal Systems, "Theory of operations," *ITS Architecture*, Oct. 1995.
- [11] —, "Physical architecture," *ITS Architecture*, October 1995.
- [12] *The Ntcp Guide*, 1999.
- [13] , John A. Volpe National Transportation Systems Center, Cambridge, MA, Spatial Data Transfer Standard Transportation Network Profile, U.S. Dep. Transportation, Bureau of Transportation Statistics, 1996.
- [14] *Ntcpnews*, 2002.
- [15] W. Stallings, *SNMP SNMP-2 and RMON: Practical Network Management*, 2nd ed. Reading, MA: Addison-Wesley, 1996.
- [16] ISO, ISO/IEC 8824-1 Information Technology—Abstract syntax notation one (ASN.1): Specification of basic notation, ISO/IEC Copyright Office, Switzerland, 1995.
- [17] J. Siegel, *CORBA—Fundamentals and Programming*. New York: Wiley, 1996.
- [18] *Tmdd and ms/etmcc Guide*, 2000.
- [19] ERTICO and ITS Europe, Trident, 2000.
- [20] D. V. Klein, "Defending against the wily surfer-web based attacks and defenses," in *Proc. IEE Workshop on Intrusion Detection and Network Monitoring ID'99*. Santa Clara, CA, Apr. 1999, pp. 81–92.
- [21] D. J. Dailey, M. P. Haselkorn, and D. Meyers, "A structured approach to developing real-time, distributed network applications for its deployment," *ITS J.*, vol. 3, no. 3, 1996.
- [22] *Learning XML: Guide to Creating Self-Describing Data*, O'Reilly, Sebastopol, CA, 2001.
- [23] *XML Schema: The W3C's Object-Oriented Descriptions for XML*, O'Reilly, Sebastopol, CA, 2002.
- [24] ANSI, American National Standard Database Language sql, ansi x3.135-1992, Amer.Nat. Standards Inst., 1992.
- [25] E. F. Codd, "A relational model of data for large shared data banks," in *Commun. ACM*, June 1970, vol. 13.
- [26] 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, Switzerland, 1995.



Daniel J. Dailey (M'91–SM'01) is a research track Associate Professor in the Department of Electrical Engineering at the University of Washington, Seattle. He also holds Adjunct appointments in Civil and Environmental Engineering and Technical Communication in the College of Engineering where he serves as the Director of the Intelligent Transportation Systems program. He has published over 100 technical papers and reports on a variety of topics including: GIS, GPS, distributed computing, modeling of stochastic processes, computer vision, and ITS systems

as well as distance learning. He is coauthor of a book titled *Wireless Communication for Intelligent Transportation System* (Norwood, MA: Artech House, Oct. 1995).

Dr. Dailey is the current President of the IEEE Intelligent Transportation Systems (ITS) Council.



Stuart Maclean received the B.Sc. degree in mathematics in 1988, the M.Sc. degree in operational research in 1990, and the Ph.D. degree in computer science in 1996, from the University of Southampton, U.K.

Since joining the Intelligent Transportation Systems program at the University of Washington, Seattle, in 1998, his research has focused on traveler information systems, algorithms for vehicle tracking, and software engineering in the ITS domain.



Fritz W. Cathey received the B.A. degree in mathematics from the University of Utah in 1973 and the Ph.D. degree in mathematics from the University of Washington in 1979.

He was a university mathematics professor for seven years during which time he performed research and published in the area of topological shape theory. In 1986 he joined the Boeing Company and worked for many years in the area of Sensor Data Fusion. Prior to joining the Intelligent Transportation Systems research group at the University

of Washington, Seattle, in 2000, he held the rank of Boeing Associate Technical Fellow. His current research involves ITS algorithm and software development, sensor modeling, and Kalman filtering with applications toward travel-time estimation using mass transit vehicles as traffic probe sensors.

D. Meyers, photograph and biography not available at time of publication.