

A comparison of three methods of data transfer for Center-to-Center communications

Luke Pond
UW ITS Research Group
June 3, 1999

In the publication of the [Traffic Management Data Dictionary](#) and the [Transit Communications Interface Profiles](#), the transportation industry has presented a list of requirements for sharing data among its management centers. These documents were both completed in 1998, and followed the recommendation of the NTCIP's Center-to-Center working group that the data be defined in the ASN.1 language. The Center-to-Center group had already completed the specification of a network protocol for exchanging encoded ASN.1 data, which is called [DATEX-ASN](#). No one has to date announced an implementation of the protocol, or a server which meets the requirements of TMDD or TCIP. This report examines the technology underlying the proposed DATEX-ASN protocol, and compares it to two other possible candidates for Center-to-Center communications: Self-Describing Data (SDD) and Extensible Markup Language (XML). [SDD](#) is a product of the ITS Research Group at the University of Washington, and was developed to solve the problem of distributing streams of real time data from traffic sensors and transit vehicles. [XML](#) is a widely-implemented new standard from the World Wide Web Consortium for structuring textual data with markup tags. It will be shown that, while the goal of standardizing the data format is achievable with any of these technologies, the choice of DATEX-ASN implies the highest cost of development, and will therefore result in the creation of a monolithic, proprietary system, unavailable to smaller transportation authorities and closed to third-party innovation.

Each of the systems contains three parts: a data definition language, a mechanism for encoding and decoding data, and a network delivery protocol. The characteristics of each of these sub-systems will be assessed to determine their contribution to center-to-center communications, and the qualities of the software that uses them.

1. Data definition language

1.1 Implications of defining a standard in ASN.1

The TMDD and TCIP standards describe the data to be shared among centers with specifications written in ASN.1. This language allows the definition of data types in terms of: 1) a collection of other data types, 2) base types such as integers or strings, or 3) a sequence (of unspecified length) of another data type. Other features that are used extensively are the enumerated type, which allows naming of particular integer values, and the bit string type, which allows multiple named values to be combined into a single field. The definition of types is straightforward and elegant, but unfortunately the language contains additional features having to do with creating values for those types, and the various standards written in ASN.1 have all yielded to the temptation of using them in one way or another. The standards diverge in the methods they use to associate globally-unique "object identifiers," and verbose descriptions, with the defined types. The divergence is made possible by another feature of ASN.1: the ability to extend the language using the "macro" facility. The TMDD standard does not do this at all, while the TCIP standard is written entirely in terms of the TCIP-CLASS macro. Furthermore, the DATEX-ASN recommendation contains ASN.1 scripts that define types with the DATA-ELEMENT macro, and specify type constraints with the VALUE-DOMAIN macro. The result of this divergence is that every standard looks different to both humans and computer programs: a program would need specific knowledge about a standard in order to understand its hierarchy of named types.

Another feature of ASN.1 is the concept of a "module," in which a script can import types from other scripts. All the standards make use of this as well, and the result is that dependency checking must be done

when parsing each script, frequently resulting in errors when the same version of the referenced script is not available.

There are few software development tools available for working with ASN.1. The TCIP and TMDD standards both mention that the ASN.1 compiler from [Open Systems Solutions](#), a commercial product, was used for their development. This compiler verifies the correctness of the script, and produces C language constructs capable of performing the encoding and decoding of data. The handful of ASN.1 tools from other suppliers do not support every feature of the language; typically providing support only for a particular application of ASN.1 such as network management. For example, IBM's [Snacc for Java](#) understands a limited set of macros defined by security-related protocols. Since the transportation applications define new macros, they cannot be parsed by these previously existing tools.

No ASN.1 compilers have been identified that are extensible, so using a standards document for some purpose other than encoding and decoding a data stream, such as presenting a user interface that allows exploration of the defined types, would not be a simple task.

1.2 SDD Schema language is a subset of SQL

SDD is closely tied to the database language SQL, in that they both use the same data definition language. The SDD Schema describes the data available in an SDD stream, and is a set of CREATE TABLE statements using the same syntax that would be used to create tables in a relational database. It is a straightforward task to transform an ASN.1 SEQUENCE type into a SQL CREATE TABLE statement, in which each member of the sequence becomes a column with a corresponding name and type in the table. *See Figure 1.* The SQL FOREIGN KEY construct can be used to represent defined types that are nested inside other defined types.

SQL does not have quite as many mechanisms for validating data as ASN.1. In ASN.1, it is possible to specify minimum and maximum values that an INTEGER type can hold, whereas SQL's representation of numbers allows the number of digits before and after the decimal place to be defined. However, SQL and ASN.1 can often provide the same level of validation. For example, the ASN.1 ENUMERATED type restricts a number to one of a set of named values. The equivalent SQL construct restricts an integer value to be an index to a table of names. Also, both ASN.1 and SQL require that character strings have a pre-defined maximum length.

The SDD software distribution contains an implementation of SDD in the Java language, including a parser for the SDD Schema. The following data types from the SQL standard are not supported in SDD: VARCHAR, DATETIME, INTERVAL, and BIT. The precise subset of SQL accepted by the Schema parser is described in the SDD [report](#). The compactness of this language means that 1) Schema parsers for operating environments other than Java could be created without difficulty using parser generators, and 2) the usage of the parser is simplified to the point where it may be invoked automatically as a side-effect of every SDD transfer. The automatic, ubiquitous invocation of the parser is desirable from a software engineering standpoint because it allows the schema to undergo backwards-compatible changes without the need to change programs that use it. As well, it enables applications to manipulate the information contained in the Schema without any advance knowledge of its existence.

Because table-based data is available from so many sources, it is possible to write programs that generate new SDD schemas. Two approaches under investigation at UW involve the automatic generation of an SDD stream from either a set of relational database tables, in which case the data source is a database server, or a set of SNMP "conceptual tables," in which case the data source is a managed network device such as a router. Another common task, recording an incoming SDD stream in a database, is simplified by the fact that every stream already contains the SQL table definitions required by the database.

1.3 XML is evolving rapidly

Version 1.0 of XML specifies a way to validate XML documents called the Document Type Definition. This part of XML allows tags to be given names and to be placed within a hierarchical structure that identifies which tags must follow, or contain, other tags. The tags that will contain data are identified as

such, but no restrictions are placed on the data apart from its being parsable character data. In other words, the DTD does not define data types such as integer or date, features that are central to ASN.1 and SQL. Of course, integer or date values, or even byte strings, can be represented as strings, but an XML parser would not be able to detect the presence of illegal values. Work is underway to remedy this with an XML Schema and associated data types, and the draft [documents](#) present a system that has much more descriptive power than either ASN.1 or SQL.

Many XML parsers have been developed and made freely available, and can be used from languages such as Perl, Java, and C. The parsers support two modes of operation: first, a simple callback mode in which user code can be executed upon the discovery of particular tags or text within the tags, and second, a more traditional tree structure called the Document Object Model can be built by the parser and explored by programs that use it.

2. Data instance format

(Definition of instance: a set of data elements whose structure conforms to some schema)

2.1 ASN.1 offers multiple encoding options

Once a data description has been written in ASN.1, a compiler can be used to generate a programmable engine for encoding and decoding data instances. Going through this process is necessary because 1) there are multiple accepted formats for encoding the data, and 2) each encoding format embodies a complicated transformation from any type of data expressible in ASN.1 to a binary data stream. Every application of ASN.1 requires a new software toolkit to be created so that clients and servers for that application can be written. If a server wishes to provide a new type of data, all of its clients must be updated with new instructions for decoding that data. This one-to-one correspondence between clients and servers makes interoperation difficult, virtually ensuring that they will be produced in tandem by the same supplier. The data provided by a server is not accessible to anyone who does not have the corresponding client program.

2.2 SDD: rows and columns in plain text

SDD defines the Contents language as its default representation of a data instance. The Contents language simply states that one or more tables may be named, followed by rows of data belonging to that table, in which each data element is separated by a comma. The SDD Contents parser makes use of the specification produced by the SDD Schema parser to validate the data instance. Valid instances are then made available to an SDD client program to be displayed, analyzed, or stored. The client program often contains knowledge of the specific application of SDD it is using, but the knowledge is on a high level, such as “this stream contains a table named *Sensors*, which contains a column named *Sensor-Type*.” As long as these types of assertions about the stream remain true, the data stream can change in any way (for example, adding a column to a table, or deleting a deprecated table) and yet remain backwards-compatible with all of its existing client programs. Another type of SDD client program is generic, in that it is capable of performing its function with any SDD stream, without any knowledge of the application involved. One such program simply lists the contents of a table as its output, thereby providing interoperability with other programming languages that can read its output stream. Another example of a useful generic SDD client is a program that would place incoming data into corresponding tables in a database. The program would accept instructions telling it which parts of the stream to store, but would contain no application-specific code.

Transmission of all data in a legible ASCII text format is a very useful feature of SDD: not only is the data instance easy to create programmatically in an SDD server, it is also easy to manipulate on the client side and easy to debug. The comma-separated-value format works well with other programs: any spreadsheet, database, or GIS application knows how to import it. *See Figure 2 for an example.* With the assistance of the generic SDD client, the only knowledge required to get useable data into one of these programs is the name of the SDD server.

2.3 Tagging data with XML

XML follows the familiar `<start-tag>data-element<end-tag>` paradigm. Since each data element is delimited by a tag that describes it, a data dictionary is not strictly necessary to understand the meaning of the data instance. If a dictionary is referenced, via the XML `<!DOCTYPE>` tag, it is used only to ensure that the data conforms to structural specifications.

Most database vendors have recently announced that XML will be a first-class interface to their systems. This means not only that there will be a simple procedure for storing XML objects in the database, but also that a SQL query against the database can return results in XML. Essentially XML is being promoted as the new database serialization format, the way to share data across organizations. This is extremely important news to the programmer who needs to provide structured data to a diverse set of clients. The next generation of database systems will have the capability to generate XML automatically to satisfy this need for interoperability, so if the data ever originates in (or is destined for) a database system, XML will be an obvious choice for its encoding format.

3. Network delivery protocol for real-time data streams

3.1 Delivering ASN.1 data instances

The NTCIP standardization effort has recognized that a “family of protocols” approach must be used to accommodate the diversity of networks and devices in the National ITS Architecture. When existing network protocols have not been adequate for the task at hand, new layers have been developed that provide the required functionality, but as an unfortunate side-effect segregate the devices that use them from the rest of the Internet. Very little effort has been taken to make the new protocols easy to understand or implement. This has a drastic effect on the quality and quantity of software tools available for working with these protocols.

For example, ITS field devices would seem to have the same requirements as any other faceless network server: the ability to notify a manager of exceptional conditions, the ability to receive commands, and the ability to respond to requests for internal state. The standard protocol for accomplishing these tasks is SNMP. When the engineers encountered the well-known performance limitations of SNMP, they defined STMP, the “Simple Transportation Management Protocol”, a set of semantic restrictions on the usage of SNMP for ITS field devices to obey. The side-effect of this choice was to eliminate the possibility of using existing network-management tools, forcing ITS managers to acquire or create specialized STMP tools of their own.

The same scenario is spelled out in the DATEX-ASN document. STMP would be an obviously poor choice for Center-to-Center communications, which requires a “publish and subscribe” mechanism rather than the connectionless polling required by STMP. The suggested new protocol, DATEX-ASN, allows (but does not require) the use of connection-based TCP rather than connectionless UDP as the underlying transport protocol. In order to make connectionless operation possible, in which a server response must fit within a single datagram, DATEX-ASN specifies that the server may respond to a request for data with a location from which a file may be downloaded using FTP. It is never made clear what need is served by accommodating UDP, and yet the protocol is made more complex for its sake. All protocol messages used to coordinate the transfer of data are specified in ASN.1; therefore DATEX-ASN inherits the limitations previously detailed.

The primary lesson to be learned from the rapid development and adoption of the World Wide Web is that application-level network protocols must avoid a proliferation of optional features. The HTTP protocol is a model of simplicity: a client makes a TCP connection to a server, sends it a simple-to-parse string containing a command along with a few parameters, and receives the response over the same connection. HTTP encourages innovation and adoption by making it easy to develop a minimally functional client or server, which will be able to communicate with any existing client or server. DATEX-ASN, on the other hand, makes two unreasonable assumptions about every potential implementation: first, that the necessary

ASN.1 tools will be available for the target platform or language, and second, that interoperability is unimportant (as evidenced by the choice of lower-level protocol layers available).

3.2 SDD and its predecessors

The extremely thin protocol layer used by SDD provides communication between programs written in C and Java and running on a variety of computing platforms. Even before SDD was invented to hide the encoding format of a stream from its consumers, the same mechanism was used to create streams of application-specific data structures. A client makes a TCP/IP connection to a server, and the server sends a series of ItsFrame structures over the connection as long as it remains alive. The ItsFrame header states the length of the frame and the type of data it contains, which may be one of Schema, Contents, or Data. One other important piece of information in the ItsFrame header is the “timeout” field, which states the number of seconds that a client should wait between subsequent frames before assuming that the connection has been broken.

The stated need for a “publish and subscribe” mechanism can be met by creating an SDD server for each available type of subscription. The TMDD standard is segmented into 17 “message sets,” *as shown in Figure 3*. As can be seen by the diversity of information available across these message sets, nearly every one would require the integration of different data sources, and they would certainly be implemented one piece at a time. A client’s interest in a particular subscription would be expressed by simply connecting to the corresponding server. The SDD software library provides the necessary starting point for quickly building a server for a new stream that would conform to stated specifications such as TMDD.

3.3 XML can be streamed too

Today’s typical use of XML involves automatically-generated XML documents, available through a web server, that can be downloaded and parsed by scripts or web browsers. For example, a frequently updated web site can publish an outline in XML format, which other web sites can consume in order to link to the new articles. To provide the real-time, event driven data streaming needed by the transportation centers, a web server could be extended with a request handler for XML streams. This extension would push message sets, represented in XML, to the client. The request could originate from a web browser or a custom application using the callbacks provided by an XML parser. The implied application-level network protocol is HTTP. This protocol provides a useful extensibility feature: both the request and response headers can contain application-specific parameter fields, which could be used by the client to ask the server to perform filtering, specify a duration for the connection, or specify a frequency of delivery. As well, the Uniform Resource Locator (URL) used to make the connection could have a close correspondence to the hierarchical categorization of the message set: for example, <http://www.traffic.org/tmdd/Network-state/Current-parking-state>.

Thinking outside the box

This report has attempted to motivate the examination of alternatives to ASN.1 as the data description language for Center-to-Center communications. Although ASN.1 has a track record of success in vertical applications, it is not well known or well supported by the software industry as a whole. The process of working with ASN.1, in which the relevant specifications must be accompanied by application-specific program code in order to access an encoded data instance, seems limited and old-fashioned when compared to the hundreds of interoperating implementations of web browsers and web servers made possible by HTTP. Taking advantage of the tremendous momentum behind the development of databases and tools that manipulate SQL or XML could greatly expedite the creation of the National ITS Architecture, and this can be only be done by moving the valuable data dictionaries away from ASN.1 and into a representation that allows more flexibility.

ASN.1	<pre> Current-link-state ::= SEQUENCE { link-id-number IA5String (SIZE(1..32)), link-delay INTEGER (0..12000), link-travel-time INTEGER (0..10800), link-volume INTEGER (1..100000), link-speed INTEGER (0..300), link-density INTEGER (0..2000), link-occupancy-percent INTEGER (0..100) } </pre>
SDD	<pre> CREATE TABLE Current-link-state { link-id-number CHAR(32) NOT NULL, section-id-number CHAR(32) NOT NULL, link-delay INTEGER NOT NULL, link-travel-time INTEGER NOT NULL, link-volume INTEGER NOT NULL, link-speed INTEGER NOT NULL, link-density INTEGER NOT NULL, link-occupancy-percent INTEGER NOT NULL, FOREIGN KEY (link-id-number, section-id-number) REFERENCES Link-identity } </pre>
XML DTD	<pre> <!ELEMENT Current-link-state (link-id-number, link-delay, link-travel-time, link-volume, link-speed, link-density, link-occupancy-percent)> <!ELEMENT link-delay (#PCDATA)> <!ELEMENT link-travel-time (#PCDATA)> <!ELEMENT link-volume (#PCDATA)> <!ELEMENT link-speed (#PCDATA)> <!ELEMENT link-density (#PCDATA)> <!ELEMENT link-occupancy-percent (#PCDATA)> </pre>

Fig.1: ASN.1 excerpt from Traffic Management Data Dictionary translated to SDD and XML

ASN.1	(non-legible binary data)
SDD	<pre> TABLE Current-link-state COLUMNS (link-id-number, section-id-number, link-delay, link-travel-time, link-volume, link-speed, link-density, link-occupancy-percent) `ES-055D:_MN_Stn',1,0,5,40,39.7,148,12.33 </pre>
XML	<pre> <Current-link-state> <link-id-number>ES-055D:_MN_Stn</link-id-number> <section-id-number>1</section-id-number> <link-delay>0</link-delay> <link-travel-time>5</link-travel-time> <link-volume>40</link-volume> <link-speed>39.7</link-speed> <link-density>148</link-density> <link-occupancy-percent>12.33</link-occupancy-percent> </Current-link-state> </pre>

Fig. 2: Appearance of the encoding of a sample data instance

Message Group	Message Set Category
1 - Roadway-Network	1.1 - Roadway-Network Description 1.2 - Roadway-Network Update
2 - Network-State	2.1 - Current-Network-State 2.2 - Predicted-Network-State 2.3 - Roadway-Network-Environment 2.4 - Current-Priority-Routes 2.5 - Current-Parking-State
3 - Network-Events	3.1 - Current-Network-Incidents 3.2 - Planned-Roadway-Events 3.3 - Event-Defined-Response 3.4 - Network-Incident-Update 3.5 - Roadway-Event-Update
4 - Traffic-Request	4.1 - Traffic-Status-Request 4.2 - Traffic-Control-Request 4.3 - Control-Response
5 - Traffic-Device-Status	5.1 - Field-Device-Status 5.2 - Surface-Street-Device-Status
6 - Traffic-Control	6.1 - Field-Device-Control 6.2 - Surface-Street-Control

Fig. 3: Message Groups and sets for External TMC Communication (from TMDD standard)