Evaluation of the Serialisation and Deserialisation Performance of Table Driven XML

Alex Ng
alexng@ics.mq.edu.au
Department of Computing, Macquarie University, North Ryde, NSW 2109, Australia

Abstract

SOAP is a simple object access protocol that builds upon the versatile XML standard in providing the widely interoperable cross-platform system-to-system Web Services. However, the intrinsic character based XML standard coupled with finite network resources and the heterogeneous nature of a vast variety of devices involved in the process bring about unavoidable delays. Table Driven XML (TDXML) provides an efficient way of accelerating the bottlenecks of network verbosity and at the same time improves the inefficient serialisation/deserialisation process of converting machine object representations to/from XML representations. TDXML encodes the message data into tabular format, with each data and attribute element being assigned unique identifiers for improved serialisation and deserialisation efficiencies. Evaluation result shows that the proposed technique reduces the resultant message size by over 200% and improves the serialisation/deserialisation efficiency by 400% when compared to other standard SOAP implementations.

1. Introduction

The requirement for an increased security measures [10] for the Web Services protocol stacks; the negative impact of XML's verbosity and processing overhead, storage requirements, and bandwidth consumption [15]; have made the need for an optimised transfer mechanism for Web Services an eminent issue. There are a number of performance enhancement techniques being proposed [13]. A majority of the enhancement techniques have emphasised on compressing XML message size through different techniques, such as, software or hardware compression [6], using shorter XML tags [16], using binary metadata [18], and using binary XML encoding to replace the unparsed, text-based XML format [9, 13].

However, not many of the proposed techniques provide efficient ways to reduce the network verbosity while at the same time improve the inefficient serialisation and deserialisation processes in the SOAP protocol. The performance of the SOAP protocol is affected by numerous factors: the implementation platform, the choice of encoding style, and the complexity of the message structure. Chiu et al. [2] confirm that object serialisation and deserialisation are the bottlenecks in a SOAP transaction.

Table Driven XML (TDXML) is a proposal that does not just aim to reduce the network verbosity issue of XML but also improves the parsing and serialisation and deserialisation processes. There are four objectives of this work. Firstly, TDXML must be based on the XML technology because the ubiquitous use of XML is vital. Secondly, the verbose nature of XML based technologies has induced extra network bandwidth requirements for Web Services. It is necessary for TDXML to provide a compact message footprint that is more network bandwidth friendly than XML so that devices running on slow network can benefit from it. Thirdly, just reduce network bandwidth requirement is not enough. TDXML should also improve the parsing, serialisation and deserialisation efficiencies. Finally, often, users want to leave their existing non-XML formats to be treated as opaque sequences of octets by XML tools and infrastructure. Such an approach would allow widely used formats such as JPEG and WAV to coexist with XML. Therefore, TDXML must be able to encode opaque binary data.

The rest of this paper is organised as follows: Section 2 explains some of the design features in TDXML. Section 3 provides the results of an analysis of the performance of TDXML. Section 4 discusses related work and some conclusions are presented in Section 5.

2. Overview of TDXML

TDXML is built upon the XML standard. A TDXML document is embedded in an XML document like a SOAP message embeds in an XML document. The presence of a TDXML document is signified by a pair of <TDXML:Envelope> </TDXML:Envelope> tags (<TDXML:Env> </TDXML:Env> in short form). Instead of grouping the elements in an XML tree

Proceedings of the Advanced International Conference on Telecommunications and
International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)
0-7695-2522-9/06 $20.00 © 2006 IEEE

structure, the elements are grouped into columns and rows. This has the advantage of reducing the number of passes needed to parse a TDXML document down to one. A TDXML document is composed of two entities: *Data Schema* and *Data*. The TDXML *Data Schema* is required for both the sender and the recipient to properly interpret the data involved. It is required at service setup time; subsequent exchanges of messages require sending an XML namespace of the original schema only. When both parties have agreed on a set of schemas, special data handlers can be developed to perform the actual serialisation and deserialisation of the data.

TDXML := f{DataSchema, Data} *(E1)*

```
<TDXML:Env>
<TDXML:DS>
person [0]::= SEQUENCE OF{
lang(0) [ATTRIBUTE]
      UTF8String  ("English")
firstname[0] UTF8String,
lastname[1] UTF8String }
</TDXML:DS>
<TDXML:CT>
[0]0(0)|English
[0]0[0]|Alan
[0]0[1]|Johnson
[0]1(0)|French
[0]1[0]|Chris
[0]1[1]|Michell
</TDXML:CT>
</TDXML:Env>
```

Figure 1: An example showing a block of TDXML data representing the details of two persons.

There are many encoding schemes available to construct a TDXML document. In this paper, only the *Aggregated Scheme* is presented. Inside a <TDXML:Envelope> block, the *Aggregated Scheme* uses two tables to provide different information: *Data Schema* and *Content Table*.

TDXML Aggregated Scheme := f{DataSchema, ContentTable} *(E2)*

The *Data Schema* provides important information for describing the structure and constraining the contents of a TDXML document. The *Content Table* contains the actual data and attribute representations of a structure instance described in a TDXML *Data Schema.*

2.1 TDXML Data Schema

A TDXML *Data Schema* is marked by the <TDXML:DataSchema> </TDXML:DataSchema> block (<TDXML:DS> </TDXML:DS> in short form). The *Data Schema* provides functions that are similar to XSD with some additional features built into it. As with XSD, TDXML uses a rich datatyping system to allow for detailed constraints on a document's logical structure, and the associated rules that are required by a robust validation framework. TDXML makes use of Abstract Syntax Notation One (ASN.1) [5] to describe the data schema contained in a TDXML *Data Schema* block. There are tools [11] available to convert the XML Schema of a document into ASN.1 notation. Using ASN.1 to describe TDXML *Data Schema* delivers the following advantages over XML Schema:

(1) The resultant schema definition is smaller;
(2) It is easier to add additional custom features to the schema;
(3) It is easy for machines as well as humans to understand the resultant schema representation; and
(4) It provides a clear separation of the information content of messages from the encoding and representation of those documents.

The example shown in Figure 1 defines the simple structure of a <person>, which contains an attribute <lang> that has a default value 'English'. Each <person> contains other details such as <firstname> and <lastname>. The element <person> is assigned tag [0], which is the root element. The attribute <lang> is assigned tag (0), which is the first attribute for the element <person>. The child elements of <person>, <firstname> and <lastname>, are assigned tags [0] and [1] respectively. The unique identification of <firstname> is [0][0], meaning the first child element of the root element; and the unique identification of <lastname> is [0][1], meaning the second child element of the root element. The unique identification of the attribute <lang> is [0](0), which means the first attribute of the root element.

2.2 TDXML Content Table

A TDXML *Content Table* is identified by the <TDXML:ContentTable> </TDXML:ContentTable> block (<TDXML:CT> </TDXML:CT> in short form). The TDXML *Content Table* contains unique identifiers for each XML element/attribute and their corresponding values in a table format. In the example shown in Figure 1, the first row has the entry `'[0]0(0)|English'` which identifies the attribute <lang> in the first occurrence of the <person> element. The second row's entry is `'[0]0[0]|Alan'` which identifies the child element <firstname> of the first occurrence of the <person> element. The value of the element is 'Alan'. The last row's entry is `'[0]1[1]|Michell'` which identifies the child element <lastname> of the second occurrence of the <person> element. The value of this <lastname> element is 'Michell'.

From the above example, we immediately can see some of the benefits that TDXML offers:

(1) The table format is easily understood by humans and machines;

(2) The simple tag numbering format enables data to be validated and retrieved with ease;
(3) The serialisation and deserialisation processes are straightforward;
(4) The resultant document format is able to provide direct-access and streaming capabilities; and
(5) Compatible with existing XML parsers.

3. Performance Evaluation

The tests reported here were based on a simple application with effectively no application logic but just performed serialisation and deserialisation of the data objects on a Microsoft Windows based platform. The test setup consisted of a multi-threaded test driver written in Microsoft's C# language using the .NET API. This test driver goes through the following stages to perform precise measurements:

1. **Initialisation Stage**: The specified test objects are constructed. All the threads are started and each goes into "Suspend" mode to wait for the "Run" signal issued by the test controller. The test controller ensures all test threads had gone through the initialisation tasks before firing the "Run" signal.
2. **Run Stage**: Once the "Run" signal is received, each test thread performs the task assigned and log the duration taken to perform the task in memory. Each test thread repeats the task until the test controller fires the "Stop" signal to the test thread.
3. **Finalisation Stage**: The test controller retrieves all the test results from each thread and consolidates the results into a report on screen.

A configuration file was used to control the setup of the test environment such as: (1) the number of client threads to be used; (2) the duration of the test run; (3) the task to be performed (i.e. serialisation or deserialisation); (4) the implementation platform to be tested; and (5) the message type to be used.

3.1 Test Scenario and Message

Five test messages were used in this study. The messages are of varying length and complexity. The first test message (*short*) represents conventional text based messages which contains a text message of 50byte length. The second message (*simple*) contains a single customer's account record and uses string, Boolean and datetime data items. The third message (*medium*) consists of twenty customer account records, representing a batch inquiry and subsequent update transaction. The fourth message (*complex*) consists of one customer record and 50 product details, representing an invoice or a customer statement. The fifth message (*simpleJpeg*) added a 14 Kbyte binary JPEG image, representing requests with a photograph of a customer or product appended to a *simple* message. This will provide some insights how TDXML handles messages which are predominantly containing binary opaque data.

In order to evaluate how well TDXML compares to other commercially available encoding mechanisms, the following three encoding mechanisms available in the Microsoft .NET platform were used:

- **SOAP**: The .NET **SoapFormatter** Class serialises and deserialises an object, or an entire graph of connected objects, in SOAP format to support remote procedure calls. This mechanism is the target reference for TDXML to compare with. The default *Document/Literal* encoding is used in the tests.
- **XML**: ASP.NET uses the **XmlSerialiser** class to encode XML Web service messages. Therefore, it is appropriate to include this mechanism in the test scenarios to verify that TDXML is able to deliver performance that at least rivals or outperforms the **XMLSerialiser** class in .NET.
- **BINARY**: The .NET **BinaryFormatter** Class serialises and deserialises an object, or an entire graph of connected objects, in binary format. **BinaryFormatter** enables native .NET to .NET application to perform remote procedure calls using .NET's internal serialisation. This is the ultimate performance target for TDXML because, in theory, it should be the fastest when compared to other mechanisms under test.

The following performance measurements were taken during the tests and used to evaluate performance, resource usage and scalability: (1) serialisation time; (2) deserialisation time; and (3) message footprint.

3.2 Performance Result Analysis

A Dell computer was used to perform the tests. The hardware and software configuration of the system was:

- Dual 2.8GHz Intel Pentium4 processor
- 512Mbytes of memory
- Intel PRO/1000MT network card
- Microsoft Windows XP Professional SP2
- Microsoft .NET Framework 1.1

3.2.1 Message Size Analysis

Files were used to capture the resultant message for all test cases. The results are shown in. Table 1. TDXML consistently outperforms SOAP and XML encoding mechanism for all test cases. For *short* and *simple* message types, it was observed that TDXML produced the smallest message sizes amongst all other mechanisms. In the *medium* and *complex* test cases, it was found that the BINARY encoding mechanism

yielded smaller message sizes than TDXML by over 30%. This was due to the *medium* and *complex* test messages containing a large number of repeated data structures (20 in *medium* and 50 in *complex*). Just as SOAP and XML, TDXML suffers from a flaw, that is, having to attach a tag for every piece of serialised data and attribute. One of the future enhancement tasks for TDXML is to incorporate new measures to allow better grouping of tags and data.

Table 1: Serialised message sizes produced by different encoding mechanisms

	TDXML	SOAP	XML	Binary
Short	121	675	204	122
		458%	69%	1%
Simple	355	1647	751	512
		364%	112%	44%
Medium	4407	14082	8796	3245
		220%	100%	-26%
Complex	11795	33073	25276	5911
		180%	114%	-50%
SimJpeg	20170	21735	20616	15310
		8%	2%	-24%
JpegMIME	15176	N/A	N/A	N/A
		43%	36%	1%

For the *simpleJpeg* test scenario, both SOAP and XML serialisations generate the binary portion using Base64 encoding. Two versions of TDXML serialiser were programmed for the *simpleJpeg* message type: one using Base64 encoding and another using the MIME multipart attachment to encapsulate the JPEG file with binary attachment. When using Base64 encoding to serialise the *simpleJpeg* test message, the result showed little gain in using TDXML. It was because the 14Kbyte JPEG file had dominated the test message and Base64 encoding had increased the message size by 33%. However, when using the MIME multi-part attachment mechanism to serialise the *simpleJpeg* message in TDXML, the resultant message was reduced to a size that is even smaller than the BINARY mechanism (15176 bytes versus 15310 bytes). This shows that the use of MIME in conjunction with TDXML is a competitive alternative for handling messages containing large portion of binary data.

3.2.2 Serialisation and Deserialisation Analysis

Specialised schema specific serialisation and deserialisation handlers were developed to make use of the direct access feature in TDXML. The .NET C# *delegate* objects were used to define reference types that can be used to encapsulate different schema specific serialisation/deserialisation methods. Each *delegate* object was associated with a specific TDXML schema specific index and stored in a hash table ready to be called during the serialisation and deserialisation processes. The serialisation and deserialisation results for all test scenarios are given in Figure 2.

TDXML outperforms SOAP and XML in both serialisation and deserialisation performance for all test message types. However, when TDXML is compared to the BINARY encoding mechanism, the change in size, complexity, and data content in a message affects the performance landscape. TDXML is the leader for the *short* (0.0426ms to perform a serialisation and deserialisation) and *simple* (0.132ms to perform a serialisation and deserialisation) message types. TDXML outperforms: BINARY encoding by over 120% in *short* message type and 20% in *simple* message type. TDXML outperforms SOAP encoding by over 500%; and XML by over 200%. The evaluation result confirms that the deserialisation process in SOAP and XML took longer processing time than the serialisation process. It is important to note that TDXML was able to reduce the deserialisation time that was much less than the serialisation process for the *short* and *simple* test scenarios.

When the test message size and complexity increased, the performance of TDXML became slower than BINARY encoding (10% difference in the *medium* message type and 90% difference in the *complex* message type). For the *medium* message type, TDXML required 1.089ms (versus 0.917ms for BINARY) to perform one serialisation and deserialisation, and 3.852ms (versus 1.983ms for BINARY) for the *complex* message type. TDXML leads the SOAP encoding by over 500% and XML encoding by 100% for the *complex* message type. It was also observed that TDXML's deserialisation process for the *medium* and *complex* message types took longer time than the corresponding serialisation processes. This differs from the case shown in the *short* and *simple* message types. This indicates that for the *medium* and *complex* message types, TDXML needs to incorporate further enhancement features to achieve even better results. The reason for the BINARY encoding mechanism being more efficient than TDXML for the *medium* and *complex* message scenarios is attributed to the fact that the resultant TDXML encoded message sizes are 50% to 100% larger than the equivalent BINARY encoding and thus, TDXML requires more time to write (in serialisation) and read (in deserialisation) the messages.

Proceedings of the Advanced International Conference on Telecommunications and
International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)
0-7695-2522-9/06 $20.00 © 2006 **IEEE**

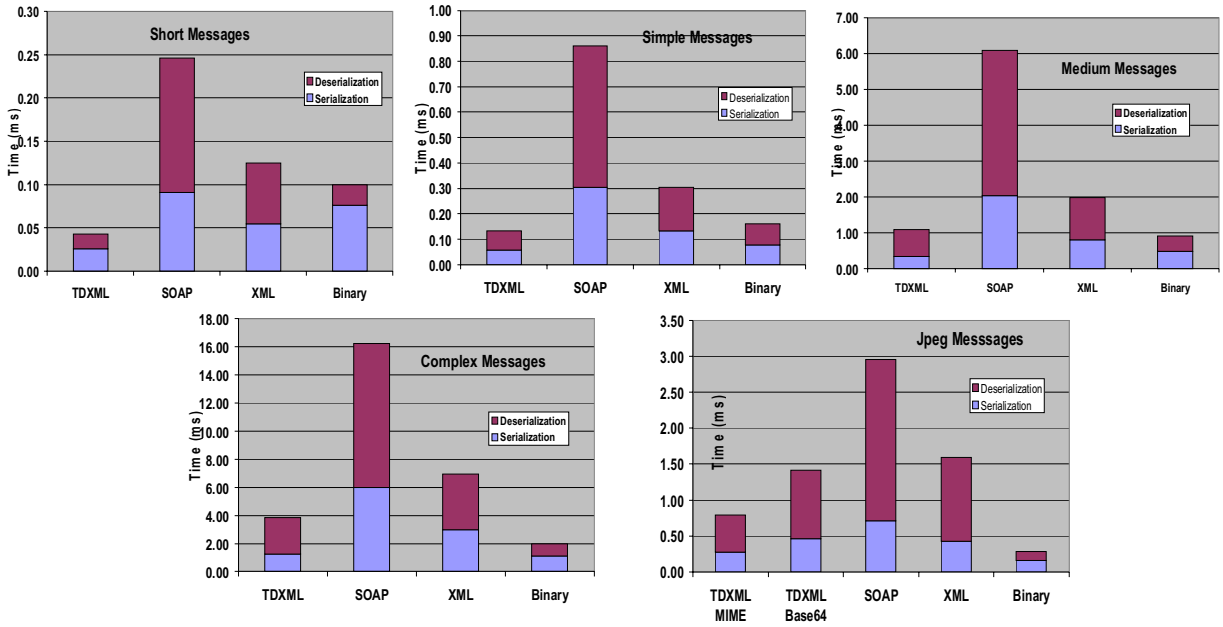

Figure 2: Average serialisation and deserialisation time for different message types using different encoding mechanisms

This also indicates that the copying of data to and from the memory buffers becomes the bottleneck in TDXML when dealing with large message size.

The *simpleJpeg* scenario was designed to study the impact of encoding binary opaque data using TDXML. Evaluation result shows that both the MIME and Base64 versions of TDXML perform faster than SOAP and XML but slower than BINARY encoding. Using MIME attachment (0.786ms) in TDXML was able to yield smaller resultant message and better performance than using Base64 (1.41ms). This shows that when a message is dominated by binary data, BINARY encoding yields better performance than TDXML. However, there is still a 100% gap between TDXML-MIME and BINARY encoding. This indicates that TDXML needs further enhancement in the aspect of improving the speed of copying data in and out of memory buffers.

4. Related Work

Some of the popular proposals for optimising the performance of Web Services are: (a) **Software compression** – There are many compression techniques being proposed, such as Millau [7], Gzip, and XMill [1, 6]. Although these compression algorithms all produce high compression ratio, there are concerns that the extra processing time required for compressing and decompressing data streams may outweigh the benefit of reduced network transit time [15]. (b) **Hardware accelerator** – Many XML hardware appliances are available in the market. The ServerIronGT E-series Web Appliance manufactured by Foundry Networks (http://www.foundrynet.com) is an example that supports XML tag switching and compression. (c) **Binary metadata** – The Portable Binary I/O metadata (PBIO) technique of creating efficient wire formats using Natural Data Representation (NDR) [18] is an example of using binary metadata on the wire, which is maintained by the sender, then decoded by the receiver into its desired form. (d) **Using shorter tags** – An example is the Cross Format Schema Protocol (XFSP) proposed by Serin [16] where elements and attributes are replaced via a tokenisation scheme which preserves valid XML document structure. (e) **Binary XML**- The examples are Sun's Fast Web Services [13] and Fast InfoSet [14] proposals. However, there are concerns [12] about issues of multiple binary representations (big or little endian) and interoperability with existing standards (eg. Infoset , and XML compression). (f) **Caching** – An example is Devaram's proposal [4] of using parameterised caching on the client side. (g) **Pull Parsing** (XPP) [17] and **schema-specific parsers** [3], are techniques that improve the parsing efficiency of an XML parser. They are starting to gain some attention in the Web Services community. (h) **SOAP Message Transmission Optimization Mechanism** (MTOM) – It is one of the recent W3C standards to meet the demand for integrating opaque data with XML and addresses the requirement of compatibility

with the XML Infoset. The MTOM proposal is able to selectively encode portions of a SOAP message using XML-binary Optimised Packaging (XOP) [8] to efficiently serialise XML Infosets containing binary data.

5. Conclusion

This paper has presented a Table Driven XML encoding mechanism which optimises the performance of Web Services by using table structures to reduce the overall message size, parsing, serialisation and deserialisation overheads. The serialisation and deserialisation behaviours of TDXML were analysed by comparing it to a set of commercially available encoding mechanisms. The evaluation results show that:

(1) TDXML yields on average 100% smaller message footprint than SOAP and XML for a variety of message types;
(2) The serialisation speed of TDXML is at similar level to the BINARY encoding (for messages less than 500byte); and
(3) TDXML reduces the overall parsing and serialisation time by over 500% when compared to SOAP and over 200% when compared to XML.

In conclusion, TDXML is an efficient encoding mechanism which is able to improve the parsing, serialisation and deserialisation performance of Web Services. However, when comparing TDXML to a BINARY encoding mechanism, there are certain areas that need to be addressed in TDXML's design: (1) TDXML is weak in handling large volume of in-transit data (revealed from the *simpleJpeg* message test case); and (2) TDXML needs to improve the process in handling repetitive data structures (revealed in the *complex* message test case).

References

[1] Cai, M., Ghanderizadeh, S., Schmidt, R., et al. A Comparison of Alternative Encoding Mechanisms for Web Services. In Proceedings of the DEXA2002. 2002

[2] Chiu, K., Govindaraju, M., and Bramley, R. Investigating the Limits of SOAP Performance for Scientific Computing. In Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02). Edinburgh, Scotland,p.246-254:IEEE, 2002

[3] Chiu, K. and Lu, W., A Compiler-Based Approach to Schema-Specific Parsers for XML, Tech Report, No. 592, Indiana University, Feb 2004

[4] Devaram, K. and Andresen, D., SOAP Optimization via parameterized client-side caching, Department of Computing and Information Sciences, , Kansas State University, 2003

[5] Dubuisson, O., ASN.1 - Communication between heterogeneous systems. Elsevier-Morgan Kaufmann, 2000

[6] Ghandeharizadeh, S., Papadopoulos, C., Cai, M., et al., Performance of Networked XML-Driven Cooperative Applications.

[7] Girardot, M. and Sundaresan, N. Millau: an encoding format for efficient representation and exchange of XML over the Web, February 15, 2001 (on-line) Accessed 11 September 2003 http://www9.org/w9cdrom/154/154.html

[8] Gudgin, M., Mendelsohn, N., Nottingham, M., et al. XML-binary Optimized Packaging W3C Recommendation 25 January 2005, (on-line) Accessed 15 February 2005 http://www.w3.org/TR/xop10/

[9] Martin, B. and Jano, B., WAP Binary XML Content Format W3C NOTE, W3C, 24 June 1999

[10] Nadalin, A., Kaler, C., Hallam-Baker, P., et al., Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS, March 2004

[11] OSSNokalva OSS Nokalva Web Site, (on-line) Accessed 20 July 2004 http://www.oss.com/

[12] Pal, S., Marsh, J., and Layman, A. A Case against Standardizing Binary Representation of XML. In Proceedings of the Workshop on Binary Interchange of XML Information Item Sets. 2003

[13] Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., et al. Fast Web Services, August 2003 (on-line) Accessed 27 August 2003 http://developer.java.sun.com/developer/technicalArticles/WebServices/fastWS/index.html

[14] Sandoz, P., Triglia, A., and Pericas-Geertsen, S. Fast Infoset, June 2004 (on-line) Accessed 15 June 2004 http://java.sun.com/developer/technicalArticles/xml/fastinfoset/

[15] Schmelzer, R. Will binary XML solve XML performance woes?, 22 Nov 2004 (on-line) Accessed 24 November 2004 http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci1027726,00.html

[16] Serin, E., Design and test of the cross-format schema protocol (XFSP) for networked virtual environments. Naval Postgraduate School: Montery, California. p. 133, 2003.

[17] Slominski, A. Home page of XML Pull Parser (XPP), (on-line) Accessed 15 July 2003 http://www.extreme.indiana.edu/xgws/xsoap/xpp/

[18] Widener, P., Eisenhauer, G., Schwan, K., et al. Open Metadata Formats: Efficient XML-Based Communication for High Performance Computing. In Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing-10 (HPDC-10). San Francisco, 2001