GJXDM
Performance Project

FINAL REPORT

XML Committee Members	Jim Threatte Tim Wilson Ellen Perry Joe Mierwa Paul Embly Ken Gill John Wandelt
GWU Test Committee Members	Sergey Kanareykin Hamzah Zeen-Aldin Tim Tickel Ammar Qusaibaty Dannee Zeedick Dr. Newton Howard
IJIS Institute Staff	Paul Wormeli David Siecker Samantha Styles

Table of Contents

Table of Figures

Table of Tables

U.S. Department of Justice
Office of Justice Programs
810 Seventh Street, NW.
Washington, DC 20531

Alberto R. Gonzales
Attorney General

Robert D. McCallum, Jr.
Associate Attorney General

Regina B. Schofield
Assistant Attorney General

Domingo S. Herraiz
Director, Bureau of Justice Assistance

Office of Justice Programs
World Wide Web Home Page
www.ojp.usdoj.gov

Bureau of Justice Assistance
World Wide Web Home Page
www.ojp.usdoj.gov/BJA

For grant and funding information contact
U.S. Department of Justice Response Center
1–800–421–6770

This document was prepared by the IJIS Institute, supported by cooperative agreement number 2003-DD-BX-1104, awarded by the Bureau of Justice Assistance, Office of Justice Programs, and U.S. Department of Justice. The opinions, findings, and conclusions or recommendations expressed in this document are those of the authors and do not necessarily represent the official position or policies of the U.S. Department of Justice.

The Bureau of Justice Assistance is a component of the Office of Justice Programs, which also includes the Bureau of Justice Statistics, the National Institute of Justice, the Office of Juvenile Justice and Delinquency Prevention, and the Office for Victims of Crime.

Preface

With the development and initial release of the Global Justice XML Data Model (GJXDM), developers have an opportunity to begin to implement the exchange of information between law enforcement and justice information systems in ways never before possible. To the extent that the data dictionary and structure in the GJXDM are widely accepted throughout the developer and user community, it will become much easier to send information and to have it transformed and applied in disparate information systems. This model is clearly a breakthrough in justice information exchanges.

The GJXDM is also a complex model, particularly with the discipline it imposes on namespace references and the sophisticated use of inheritance in defining data relationships. It has developed into a large model containing over 2500 data elements and complete references to external code tables for completeness.

The model rigorously applies the open standards embodied in the World Wide Web Consortium (W3C) regarding the precise use of XML.

It should be noted that the design of the GJXDM has always been meant to be a superset of possible information exchanges, in that only a defined subset of the model would be used in any given exchange. This was the motivation behind the development of the GJXDM Subset Schema Generator that was released during the course of the testing reported on in this document.

The sheer size, complexity, and thoroughness of the GJXDM created a concern for developers and system administrators regarding the computational and network resources that may be required to support the exchange of information based on this model or properly conceived subsets thereof. The past practices particularly in law enforcement of using cryptic text messages with 3-character mnemonics to delineate content were an efficient way to transmit information but made the computer-to-computer exchange of information extremely difficult. The GJXDM and its XML foundation provide an elegant and economical solution to this problem of exchange transmission, but at a cost of the overhead added to the transmission.

As the GJXDM became ready for release in production, many developers and administrators realized that very little information was available regarding the real-world performance of properly defined subset schemas. The Office of Justice Programs (OJP) in the U.S. Department of Justice concluded that performance testing of the GJXDM was a high priority. The testing would define the relative impact of using the GJXDM as a basis for creating XML schemas representing documents exchanged.

To carry out the testing program, OJP awarded a grant to the IJIS Institute to conduct performance testing of the initial production release to provide industry and government developers and administrators with insights into the impact of using this important new tool. The IJIS Institute created a partnership with The George Washington University (GWU) to conduct the performance testing under the leadership of a project committee. The committee consisted of developers from IJIS Institute member companies and other organizations having specific interest in the test results (Georgia Tech Research Institute, the engineering arm that designed the GJXDM; the Global XML Structure Task Force, the practitioner group guiding the development process; and OJP).

The Project Committee, in conjunction with the GWU research team, developed the test plan further described in this report and supervised the testing and analysis of the results which the committee hopes will be helpful to the justice developer community.

The IJIS Institute is pleased to have been engaged in this research, which we believe makes an important contribution to the knowledge about this powerful new tool for information exchanges in the justice field. This work and the many individual successful implementations of the GJXDM together serve as evidence of the value of the GJXDM and the methodology for constructing XML schemas in an orderly and consistent fashion.

Paul Wormeli
Executive Director
IJIS Institute

Acknowledgements

The XML Performance Testing Committee acknowledges the work of many people in setting the stage for this project and its work. The project has had as its focus the evaluation of performance of the Global Justice XML Data Model (GJXDM), the publication of which is a landmark event in criminal justice, setting the stage for national information sharing without precedent. This project, and the many ongoing implementations of the GJXDM, would not exist without the full support of the Bureau of Justice Assistance of the Office of Justice Programs in the Department of Justice. In particular, the persistent efforts of Patrick McCreary and Bob Greeves to build a national consensus around information sharing mechanisms have resulted in this significant national accomplishment. The leadership of former Assistant Attorney General Deborah Daniels in sustaining DOJ and other national support for this effort through the Global Information Sharing initiative of OJP has been the key to the success of this development from which all criminal justice agencies will derive benefits.

The committee also acknowledges the outstanding work of the computer scientists at the Georgia Tech Research Institute (GTRI) in building this robust and powerful model for use by criminal justice agencies throughout the nation. Their attention to the needs to build a flexible and extensible model that serves all of criminal justice yet embodies the latest state of the art thinking in exchange modeling has created a model worthy of replication anywhere. We also pay tribute to the XML Structure Task Force who so intelligently guided the work of GTRI in the construction of the model.

The IJIS Institute and the committee are grateful for the support of the companies who allowed their representatives time to devote to the work of this committee, including SAIC, TriTech Software Systems, VisionAir, and MTG Management Consultants, L.L.C.

The committee provides special acknowledgement for the work of Dr. Newton Howard and the graduate students of the Cyber Security Policy Institute of the George Washington University who conducted the testing under the guidance of the committee. Their innovative work in developing testing procedures and software was a critical contribution to the success of this testing effort.

We particularly thank SAIC for the contribution of equipment used to conduct the test and to Microsoft for the contribution of system and application software used in the testing.

All of the members of this committee worked very hard to guide the project and participate in the development of this report, but special recognition is in order for Tim Wilson of TriTech. Tim has been a tireless force on this committee that has contributed well beyond the call of duty. He has stepped in twice to support the project when it needed additional resource commitment in order to complete

its tasks. He has invested incalculable personal time in his efforts to extend the research of technical issues and to dissect areas of specific concern and investigation. He has provided senior technical leadership working with a major national corporation in drilling down into the platform performance discrepancies to address performance discrepancies constructively. Additionally, Tim has been the "go to" person for the George Washington University team who helped them solidify their test implementation approach and provided them with the test baselines they needed in a majority of the areas of investigation.

James L. Threatte
Chairman
XML Performance Testing Committee
IJIS Institute

And

Vice President, Public Safety
SAIC

Executive Summary

The IJIS Institute was tasked by the Bureau of Justice Assistance (BJA) to test the Global Justice XML Data Model (GJXDM) to determine its performance characteristics under various conditions and to provide guidance to developers concerning best practices for its use. This report provides the results of those tests and releases to the GJXDM developer community an extraordinary body of system performance data regarding the performance of GJXDM using "real-world" transaction scenarios. A variety of hypotheses were tested in order to address questions concerning the performance of the model when varying such parameters as architectural structure of schemas and their proper subsets, volume of data content, local versus remote accessing of schemas, network speeds, and length of element names (Tag Names). The objective was to determine recommended approaches to implementation of the model in real world information technology application environments.

This report provides analyses and findings based on the project team's interpretation of the data and makes recommendations concerning best practices for employing the GJXDM in a production environment. The appendix to this report contains a repository of raw data that is available in electronic format for further analysis and findings. As further analysis is performed and as new findings or refinements emerge, additional tests and updating of this report will likely be necessary. The IJIS Institute project team has provided a series of repeatable test scenarios and software testing procedures along with code and data that can be used to run future tests. Since the testing environment can remain a constant, evaluating and comparing the performance baseline to future changes to tools, platforms or the model should be relatively simple.

Due to the number of permutations and combinations of transactions tested, the raw data output from this report is extremely complex and can be challenging to comprehend and analyze correctly. The findings and recommendations presented in this report have attempted to abstract this complexity into higher-level statements in lay terms that can be comprehended by non-technical personnel reading and evaluating this report. However, achieving a detailed understanding of the test regimens and the performance data requires an examination of the entire performance data set and an understanding of the precise methods used for the execution of the tests, which will likely require a significant level of technical expertise.

Testing Methodology - The same hardware, software parsers (.NET and J2EE), use case scenarios, and transmission media were used for all tests. The approach was designed to separate the impact of the use of XML itself from the performance of the GJXDM. No attempt was made to evaluate the difference between the uses of XML versus non-XML technology (i.e. legacy systems). A variety of real-world Use Cases (AMBER Alert, Field Reports, Incident Reports,

Inmate Records, and RAP Sheets) were employed. Remote versus Local access to schemas was tested using the Incident Report Use Case. Performance variance using reduced element names (Tag Names) was tested using the AMBER Alert Use Case.

Testing was divided into five distinct phases:

1. *Creating Instances* where the data source is either a relational database or a flat file (actual time to load data was not included)
2. *Validating the Instances using a software validation tool at the Sending Server* location (back-end server that is normally used to control the storage of the data)
3. *Transmission of Instances* – sending the message from the Sending Server to the Receiving Server
4. *Parsing and Validation of Instances using a software validation tool at the Receiving Server* location (front-end server that is the interface to the user (client).
5. *Transformation of GJXDM to a Target Format* – (i.e. using an Extensible Style Sheet Language Translator (XSLT) to format the data content (payload) for presentation using a Web browser)

It is important to note that when this project was initiated the Subset Schema Generator developed by the Georgia Tech Research Institute (GTRI) was not yet completed and available. Therefore, initial emphasis was on testing the entire data model along with subset schemas that were generated by the test team. However, as the Subset Schema Generator became available it was added into the testing regimen and resulted in findings that underscore the importance of operating with appropriately structured Schema Subsets of GJXDM rather than using the entire model during production operations or using poorly constructed subschema.

Testing Structure and Results - Testing structure for the project was divided into five categories:

1. *Varying the Architectural Structure of Schemas and Subset Schemas* - Each scenario tested three different schemas with data content being held constant.
    a. Full GJXDM and all of its associated Proxy Schemas
    b. A Typical Partial Subset Schema containing only elements required for the Use Case with Object Hierarchy included
    c. A Minimum Subset Schema containing only elements required without Object Hierarchy being included and eliminating unused code tables.

The test results concluded that the size and complexity of the model and the schema design have a significant impact on performance in a production environment. The results showed that validation is the critical consumer of time and that its latency is a function of overall depth and breadth of the data model's complexity. This argues for minimizing the use of validation during production mode once testing has established the integrity of the application. However, in situations where front-end and/or back-end validation is deemed essential, employment of hardware accelerators (sometimes referred to as XML Appliances) can be employed at modest cost to eliminate any degradation of performance when conducting on-line validations during production use. Such accelerators employ firmware for parsing, validation, and authentication management. They can reduce elapsed time by 100-fold even with use of the full Schema (significantly less than one second of elapsed time).

2. *Varying the Data Content (Payload)* - Each scenario tested three data content variations. The Typical Partial Subset Schema was employed and held constant for all of these tests.
    a. Minimal Data Content (only mandatory fields)
    b. Typical Data Content (for selected Use Cases)
    c. Large Data Content (all possible fields from the full schema that could be associated with the Use Case)

The test results concluded that data content does not appear to have an appreciable impact on performance.

3. *Varying access to the Schemas and Subset Schemas* - Two schema access approaches were tested.
    a. Local Access (schemas in same Namespace as Instances)
    b. Remote Access (schemas called remotely from Instances)

The test results concluded that there is a significant negative impact when referencing schemas or subset schemas from remote locations.

4. *Varying the speed of the Network* - Various network speeds were tested ranging from high to low bandwidths.
    a. 100 Megabits per second
    b. 10 Megabits per second
    c. 56 Kilobits per second
    d. 9,600 bits per second

The test results concluded that network speed does not appreciably influence performance until bandwidth is reduced to 9,600 bits per second. This has significant implications for wireless applications since few commercial radio

networks offer more that 9,600 pits per second. However, higher bandwidth wireless applications should have no problem using GJXDM.

5. *Reducing the length of Element Names (Tag Names)*
    a. GJXDM Tag Names that employ ISO 11179 and other standards for increasing interoperability
    b. Shortened Tag Names (3 digits)

The test results concluded that communication time was significantly impacted when employing the longer Tag Names. However, this does not always affect total processing time especially when communication time is a small percentage of total transaction time. Reducing the length of Tag Names tends to obscure the semantic meaning of the data and significantly reduce the potential for interoperability. This implies that the cost of reducing the Tag Names outweighs the benefits of improved execution time.

Other Findings

- Increasing the complexity of the Data Model tends to increase the time required for validation.
- Validation of XML instance documents consumes a significant portion of transaction time. Validation latency is mostly the result of overall depth and breadth of the data model's complexity. This was proven by comparing the time to validate using the pre-release version of GJXDM to the operational version.
- Use of the full GJXDM for validation of instances in an operational environment is not feasible unless a hardware accelerator (XML Appliance) is employed. Therefore, the Subset Schema Generator should be used to develop appropriate schemas that can be validated more efficiently. In addition, once application integrity has been verified, the validation function should be disabled for operational use.
- The .NET parser performance was orders of magnitude slower than the J2EE parser performance for all transaction types, categories, and variations during validation. Its design appears to have been oriented to use of Data Type Definition (DTD) standards rather than being optimized for use with XML Schemas. However, the .NET parser can be faster in discrete transaction phases. When .NET is in full compliance with W3C standards and is optimized to support complex schemas it should be competitive with J2EE use.
- Generation and use of proper Subset Schemas significantly improves the performance of GJXDM. The testing shows that this benefit is proportional to the size of the model and the appropriate design of the Subset. Performance degrades linearly as size increases. The Subset Generator was found to produce effective and efficient Subset Schemas. However, it is possible to

spawn a Subset Schema with worse performance characteristics than the full GJXDM if adequate attention is not paid to the design of the Subset Schema.
- Due to the complexity in depth and breadth of GJXDM, many conventional off-the-shelf software development and integration tools available during testing were not up to the task of working with the GJXDM. As technology changes rapidly and dramatically, these problems may be alleviated, but care must be taken in the selection of tools for use in an operational mode.

Best Practices

- Minimize the use of the validation function in production mode. Use it when initially testing exchanges for integrity, but disable it when schemas are moved into production mode.
- Minimize the use of the Document Object Model (DOM). It increases transaction time significantly. Time to load the document into DOM was significant when tested. There are better methods for performing common data processing tasks. (For advice on this topic, see http://www.xml.com/pub/a/2001/11/14/dom-sax.html?page=1).
- Use the Subset Schema Generator to produce appropriately designed Subsets rather than employ the full GJXDM.
- Load elements, objects, and whole files selectively on demand rather than load them all at once.
- Subset Schemas should maintain the list of elements, types, and attributes in the exact same order as the full GJXDM. This decreases parser time.
- Declare Subset Schemas in GJXDM Namespace. Do not reference remote proxy schemas. Re-use proxy schemas by inclusion in the Subset Schema file. Most parsers do not handle that type of deployment with acceptable performance.
- Choose parsers carefully. Since W3C standards leave optimization mostly up to the parser implementation, it can have a significant affect on performance.
- Do not use GJXDM for low bandwidth wireless applications. It is not intended for use in managing internal communications within a "closed system" environment such as mobile computing when low bandwidth is the default environment.
- Use a hardware accelerator (XML Appliance) if validation is required during operational use and if use of the validation function is likely to decrease performance (this frequently depends on the size and complexity of the Subset Schema generated).

Recommendations for Further Research - Consideration should be given to initiating further research concerning the use of GJXDM. The following are some suggestions for that purpose:

- Explore the effects of various levels of data model complexity on performance. Use the results to provide guidelines for building constraint schemas.
- Organize User Groups to share experiences that can lead to improved approaches to Best Practices.
- Explore the potential for splitting the data model into sub components. There is some evidence that parsers may be better at optimizing if they only have to operate on segments that can be loaded dynamically.
- Accelerate the development of information exchange package descriptions that will allow models that be used as a starting point for building exchanges.
- Replicate performance testing on actual exchanges to evaluate alternative ways to improve schema construction.

Document Organization

This report contains four main sections and three appendices. The sections and appendices are named, described, and organized as follows.

- The *Methodology* section describes the research goals and objectives, details on the scope of the testing, and the test approach and environment.
- The *Test Results* section provides the findings, summaries, analyses, and graphic depictions for the tests described.
- The *Lessons Learned* section provides the comments, tricks, tips, and hints captured during testing.
- The *Proposed Next Steps* section summarizes the Committee recommendations for further research.
- *Appendix A* provides the raw data from the GJXDM performance testing in Excel worksheets.
- *Appendix B* is the GJXDM Performance Testing Primer that describes the testing set-up and procedure followed to obtain the testing results and a primer for interpreting the results.
- *Appendix C* provides a list of acronyms used in this report.

Methodology

Goals and Objectives

The GJXDM Performance Project has many goals and objectives. The goals and objectives in the following paragraphs are in priority order.

This project will result in a written report detailing information regarding GJXDM performance testing results. In addition, the report will make recommendations to the practitioner community for implementing the GJXDM.

The project will provide useful information to the practitioner community on the performance of the GJXDM using real world scenarios and data. Performance measurements will be categorized into the following areas of investigation using both the full GJXDM model and only those constructs required by a given event or document (succinct definitions). These include the creation and validation of instances, validation and parsing of instances, communications or transmission of instances, and transformation of instances (e.g., XSL transformations or DOM tree manipulations).

The project shall distinguish between the performance impact of GJXDM, XML, and the data transmission network.

The project shall define a variety of "use cases" that describe the categories of usage in the real world. Use cases shall be an integral part of the performance baselines since they provide the context that would guarantee future comparative consistency.

The project shall evaluate content and structure separately. By testing with multiple element name lengths and content volumes, better perspective is available of the overhead imposed by XML and by the GJXDM naming convention.

The project shall evaluate the impact of abstraction.

The project shall be flexible and dynamic in its testing approach.

The project shall identify performance measurement and tuning parameters, to the degree possible, that will define and document the optimal performance tuning parameters.

The project shall utilize a variety of use case scenarios in order to provide information regarding performance characteristics that include reliability, efficiency, high functionality, security, and robustness.

Testing Scope

The testing scope identifies what is tested and what is not. The following questions define the GJXDM testing scope and seek to answer specific field performance concerns on standard computer conditions and configurations.

The testing of the GJXDM will seek to answer the following questions:

- What are the performance implications of migrating from the JXDD 3.0.0.0 pre-release version to the GJXDM 3.0 initial operating release?
- What are the performance variances when the same transaction uses a variety of data content sizes?
- What are the performance implications of using the full GJXDM versus using sub-schemas?
- What are the performance implications of using the long tag names defined by GJXDM?
- What are the performance variances when GJXDM schemas are retrieved from different locations (e.g., local vs. external site)?
- During the life of GJXDM transactions, what is the percentage of the time the transaction expends at key processing steps?
- What "lessons learned" should the test team share with practitioners that will assist them in selecting tools and configuring their technology environments to optimize the GJXDM?

Test Regimen

A thorough and well-defined regimen of testing and evaluation puts the GJXDM through its paces to assure that user requirements are satisfied. The development of the test regimen has taken into account the following testing scope considerations:

1. What can reasonably be measured (e.g., elapsed wall clock time, CPU time, memory usage, cache size usage, workspace usage, disk storage requirements, message transmission time, message transmission length, etc.)?
2. Determine what measurements are relevant for what we are trying to accomplish and eliminate anything of insignificant value (e.g., required storage).
3. Develop use cases and testing scenarios that select representative events that vary in complexity, documents, and structures.
4. Identify content and create "standard" test instances. Content in the "standard" tests should be representative, but consistent and minimal, so the test measures the processing of the schema and tag variables.
5. Test a variety of content volumes in order to determine how performance correlates to the content size.
6. Test on multiple technology platforms to the degree possible to eliminate test platform variability so that the GJXDM is tested and not the tools.

7. Test the full GJXDM and subsets of the full element definitions (i.e., sub-schemas).
8. Test the same content with more succinct tag names. Be extreme and use the shortest tag names (i.e., the NCIC standards: <nam>, <dob>, and <soc>).
9. Render a set of HTML documents using at least two XSLT engines.
10. Do "positive" testing to prove or disprove the theory that the GJXDM model is usable in its current form and under what circumstances. The circumstances will identify the type of system requirements to make using the model most efficient.
11. Perform testing over a managed IP network with a repeatable and predictable degree of infrastructure that ensures adequate bandwidth and throughput at all times, in order to control the impact of the network infrastructure on the performance test results.
12. Recognize and capture any additional overhead imposed by the GJXDM protocol, separated from the standard XML environment.

Technology Areas

The main areas tested occurred in three measurable technology areas. These areas are the back-end server and the front-end server.

Back-End Server

The performance characteristics of the back-end server were thoroughly tested and evaluated. The data captures a series of specific metrics that measure all actual performance characteristics of a typical user interaction. The performance testing measured the following back-end server functions:

- Transformation of the raw data into a valid GJXDM instance that is ready for transmission (if applicable). The time required to extract the data from the data source is not measured.
- Validation and parsing of the instance before transmission.
- Processing of the instance using representative business logic specific to the kind of transaction (if required).

Transmission of Information

The transmission of information measures from the time that the one server sends the information to the other server until the time that the other server has received all of the data. In addition, a number of other measurements captured will relate to the representation of the data on the network.

Front-End Server

The performance characteristics of the front-end server and client were thoroughly tested and evaluated. The data gathers a series of specific metrics that measure all actual performance characteristics of a typical user interaction.

These actual performance measurements were incorporated along with the back-end server and transmission of information metrics in order to form a metric that will represent the user's perception of performance. The front-end server and client performance tests measured the following front-end server functions:

- Parsing and validation of instances on the front-end server (e.g., an XML request for data to send from the back-end server).
- Processing of the instance using representative business logic specific to the kind of transaction.
- Transformation of the instance for display on a client workstation (if required).

Excluded Areas of Measurement

The GJXDM testing regimen did not measure the relative efficiency of existing data exchange methods when compared to XML and the GJXDM. The value proposition of using XML must be acknowledged and appreciated based on its own merits. Practitioners that migrate from a legacy environment, consisting mostly of unstructured text transactions, to any XML environment will have noticeable impacts in the performance of their current systems. The interoperability gains from using XML and the GJXDM must be appreciated and any reduction in "as is" system performance must be dealt with as part of an overall optimization strategy that may require the restructuring of how data is processed.

The GJXDM testing regimen did not measure the time required to extract data from data sources, such as a text or relational database in order to eliminate the variability of alternative methods of storing data. For example, the design of a relational database schema can be extremely efficient or extremely inefficient. The purpose of this test regimen is to test the performance of GJXDM-based transactions, and not the efficiency of any data storage method (e.g., object-oriented databases) or extraction method, which should be the topics of a separate study.

Test Approach

This section describes the approach for executing the test plan. It categorizes the testing regimen as a hierarchy and provides a detailed description of the items in each level of the test hierarchy.

The following diagram depicts the test level hierarchy.

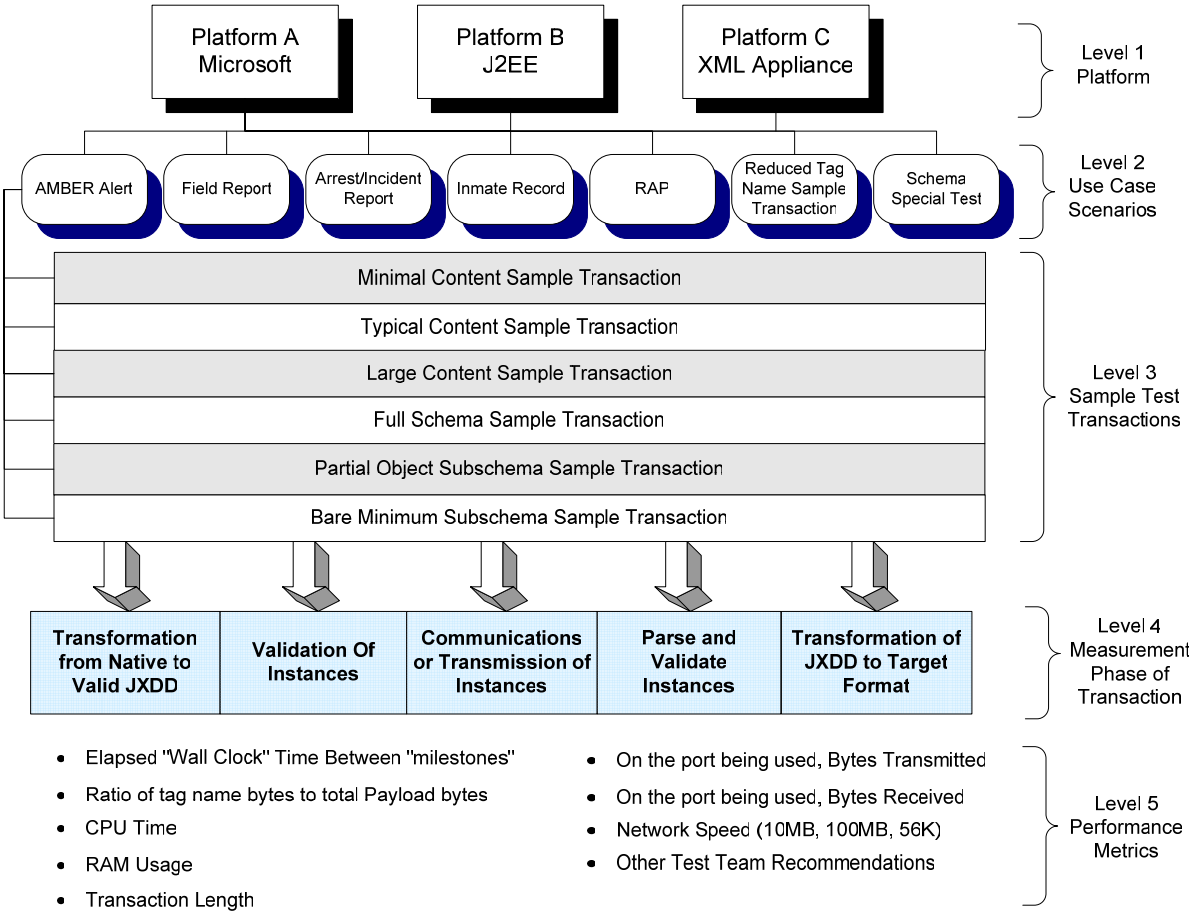

Figure 1 – Test level hierarchy of the GJXDM tests

Testing activity is organized in five test levels. For each test level, one or more sequential cycles execute.

Level 1 – Platform

Level 1 of the testing hierarchy consisted of three operational platforms – Microsoft, J2EE, and XML Appliance.

Microsoft

The Microsoft platform consisted of servers operating on the Microsoft Windows development and deployment platform. This included Windows 2000 Advanced Server and Microsoft Internet Information Server (IIS) 5.0. The test code and operational environment were developed using the Microsoft .NET Framework 1.1 Toolkit.

J2EE

The J2EE platform consisted of servers operating on a Linux development and deployment platform. This included the RedHat Linux 9.0 operating system and

Tomcat Java-based Web server software. The test code and operational environment was developed using version 1.4.2 of the Java Software Development Kit (SDK) and the Java Web Services Developer Pack 1.3.

XML Appliance

The XML Appliance platform consisted of the Microsoft platform and J2EE platform (each described above) with the addition of a Sarvega XPE 2000 XML Guardian Gateway appliance. Processing text-based XML is CPU-intensive, typically consuming 45% to 80% server cycles for compute-intensive operations such as XSLT, Schema Validation, and XML Security algorithms. This makes scaling XML-based Web services expensive and creates business-impacting latencies. Sarvega's XML Appliance utilizes a highly optimized binary data stream that solves these XML processing problems.

Shared Resources for All Platforms

The Microsoft, J2EE, and XML Appliance test platforms shared common client and the network environment resources.

Client

The test environment client used to interact with the testing interface was a laptop running Windows 2000 Advanced Server operating system with Internet Explorer 6.0 Web browser.

Network

A created network environment isolated the test environment from the external networks. This guaranteed repeatable network performance.

Level 2 – Use Case Scenarios

A variety of "use case" operational scenarios defined what the functional and data items were for each test. Use case scenarios run under each of the Level 1 platforms. The selected use case scenarios leverage existing assets such as schemas and XSLT.

The following is a list of the use case scenarios defined under Level 2.

- AMBER Alert
- Field Report
- Los Angeles County Arrest/Incident Report
- Inmate Record
- Report of Arrest and Prosecution (RAP) Sheet
- Reduced Tag Name (based on one of the other test cases)
  - Note: This special test reused the existing AMBER Alert test in order to address the performance question: How much does it matter if I reduce the length of the GJXDM tag names?
- Schema Special Test

- Note: This special test reused the Incident Report test in order to address the performance question: Does it matter whether the schema is obtained from a local or remote location?

Level 3 – Sample Test Transactions

A variety of sample test transactions were executed for each use case scenario. A brief description of each sample test transaction follows below:

Schema Variation Testing

For each test scenario, we established three different GJXDM schemas to be tested – a full schema, a partial Subschema and a minimum Subschema (for some test cases, all three variations may not applicable). In order to minimize the complexity of this test, each of the schema tests used the "Typical Content" data as defined below. With this approach, the data content size holds constant and the performance impact of schema variations were the focus of the test. A definition of each of the three schema variations provides further detail below.

- *Full Schema Sample Transaction* - This transaction utilized the full GJXDM and all proxy schemas.
- *Partial (Typical) Object Subschema Sample Transaction* - This transaction utilized a subset schema of the GJXDM tailored for the use case requirements. The subset schema included only the elements required with the object hierarchy for the reference document based on the domain of the use case.
- *Bare Minimum Subschema Sample Transaction* - This transaction utilized a subset schema of the GJXDM tailored for the use case requirements. The subset schema included only the elements required without the object hierarchy of the reference document based on the domain of the use case. This subschema eliminated unused code tables.

Data Content (Payload) Variation Testing

For each test scenario, we established three different data contents to be tested – minimal, typical, and large. In order to minimize the complexity of this test, each of the data content tests used the partial object subschema as defined above. With this approach, the GJXDM schema holds constant and the performance impact of data content variations were the focus of the test. A definition of each data content variations is below:

- *Minimal Content Sample Transaction* - This transaction included only the mandatory fields. A minimal instance contained content only for those elements that are required to make a valid transaction. An example of this approach is a RAP sheet instance that provides only the required person information and a single arrest event.
- *Typical Content Sample Transaction* - This transaction included the typical fields and typical complexity as it applies to the domain of the use case. A typical

instance contained content for those elements in the minimal instance as well as some commonly occurring elements. In this case, a RAP sheet instance would contain the all data "typically" provided regarding person information and two arrest cycle occurrences including the related court conviction and supervised custody status.

- *Large Content Sample Transaction* - This transaction included all possible fields from the GJXDM schema. Where the schema allows for multiple occurrences or repeating data, the test transaction provided more than two (each) of those multiple occurrences or repeating data where applicable. A large instance contained content for those elements in the typical instance as well as a number of recurring structures. For example, a RAP Sheet instance would contain all data elements allowable in the person information as well as five arrest cycles including the related court convictions and supervised custody status.

Level 4 – Measurement Phase of Transaction

For each sample transaction that executes under Level 3 of the testing hierarchy, the performance measurements captured five distinct phases of the life of the transaction. Considering that the transaction is "born," "lives," and finally "dies," each of these five distinct phases measured the life of the transaction.

Creating GJXDM Instances

This phase of the transaction is where the system creates a valid instance for transmission. The data source is in a relational database or a flat file based on test team preference. Time required to pull the data from the data source is not measured.

Sending Server - Parsing and Validation of Instances

This phase of the transaction uses an XML validation tool or library to validate the instance against the transmission schema at the sending server (depending on the use case, either the back-end or the front-end server may be sending data).

Transmission of Instances

This phase of the transaction sends the message via the network from one server to the other server.

Receiving Server - Parsing and Validation of Instances

This phase of the transaction uses an XML validation tool (or API call) to validate the instance against the transmission schema at the receiving server.

Transformation of GJXDM to Target Format

This phase performs and measures the transformation/manipulation phase of the transaction, such as when an XSLT stylesheet is used to format the payload for presentation on a Web browser.

Level 5 – Performance Metrics

Performance metrics capture specific measurement phases for each transaction executed under Level 4 of the testing hierarchy. The following is a list of the performance metrics captured under network speeds of 100Mb, 10Mb, 56Kb, and 9.6Kb:

- Wall clock time between "milestones"
- Ratio of tag name bytes to total payload (actual data) bytes
- CPU time
- RAM usage
- Total transaction length
- On the port used for sending, the number of bytes transmitted
- On the port used for receiving, the number of bytes received

Interoperability

The testing project followed the interoperability rules as defined by the Global XSTF. For a copy of these rules, see http://it.ojp.gov/topic.jsp?topic_id=138.

Test Environment

The Cyber Security Policy Research Institute (CSPRI) laboratory, at the George Washington University (GWU), Foggy Bottom campus, located in Washington, D.C., performed all the required testing for this project.

This section provides a detailed description of the hardware and software used to support testing.

Hardware

This section defines and describes the hardware environment used to conduct the required testing of the GJXDM.

Figure 2 depicts the hardware environment the CSPRI laboratory used to conduct the GJXDM testing.

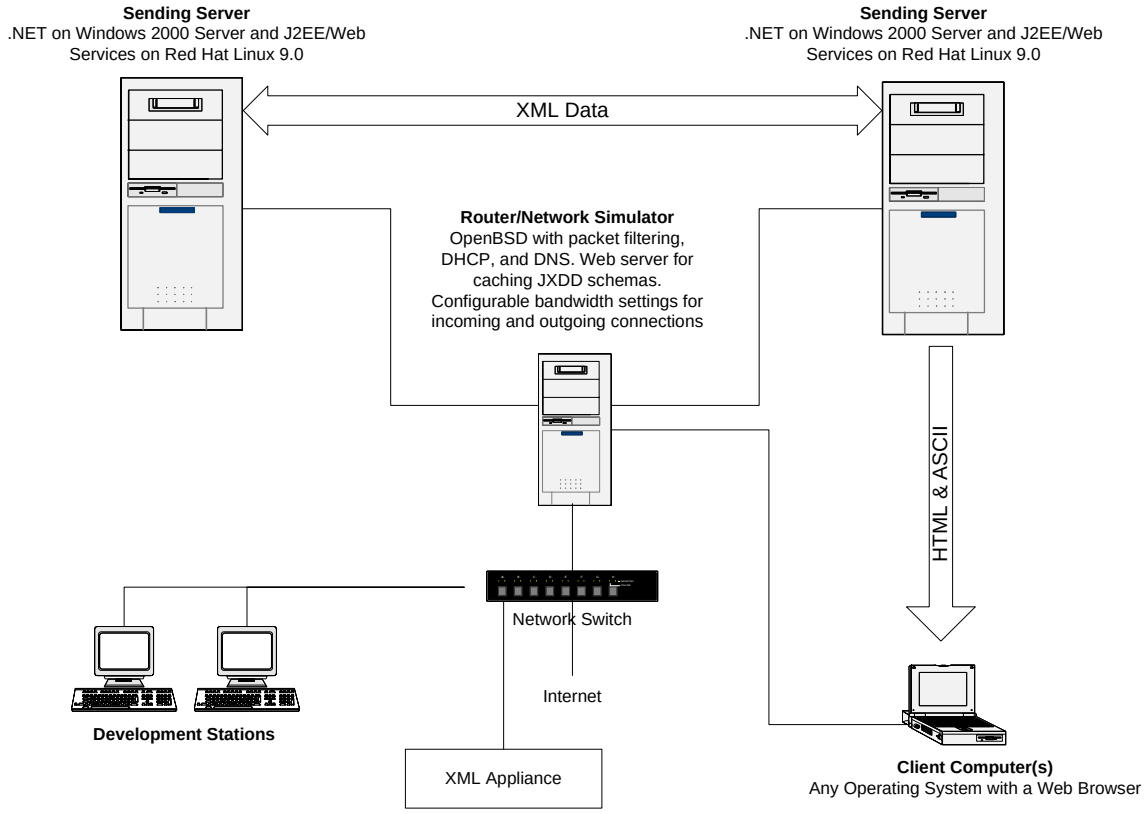

Figure 2 – Hardware environment supporting GJXDM testing

Server 1 (Back-end server)

Back-end server stores information and can receive updates or respond to specific information requests. The IJIS Institute and its industry members provided a back-end server for this project. The computer has a 366MHz processor, 256MB RAM, and is configured as a "double-booted" computer that allows either the Windows 2000 Advanced Server or the Red Hat 9.0 Linux operating system to be running.

Server 2 (Front-end server)

Front-end server provides user access to the system. It runs a Web interface that the client interacts with, and communicates with the back-end server when there is an information request or update. The computer has a 450MHz processor, 256MB RAM, and is configured as a "double-booted" computer that allows either the Windows 2000 Advanced Server or the Red Hat 9.0 Linux operating system to be running.

Network Simulator (router)

Network simulation takes the guesswork out of network management and device design, ensuring better visibility into performance, as well as greater network uptime and reliability. A router is a device that determines the next network point to which a packet should be forwarded toward its destination. The router connects at least two networks and decides which way to send each information packet based on its current understanding of the state of the connected networks. The computer has a 233MHz processor and 128MB RAM. The CSPRI team equipped the machine with four network interface cards to interconnect the two servers and the client machine, and provide them with access to the Internet (which can be disabled during the test runs to ensure quality of service).

Client Computers

In computing, a client is a system that accesses a service on another computer via a network. The IJIS Institute provided one client computer for this project. It was a Dell Latitude C600 laptop computer with a 1.2GHz processor and 256MB RAM.

Network Switch – 100Mbps

A network switch is a computer-networking device that connects network node and/or segments. This switch provided connectivity for additional network nodes (e.g., development or monitoring stations, or the XML appliance) while at the same time isolating the test environment from outside influences that would affect the predictability of network performance. It was a Netgear FS-608 8-port 10/100 switch.

XML Appliance

The XML Appliance used was a Sarvega XML Guardian Gateway appliance.

Software

This section defines and describes the software environment and tools used to conduct the required testing of the GJXDM.

Server 1 (Back-end server)

When operating under the Windows 2000 Advanced Server environment, Microsoft .NET Framework and SDK 1.1 were used. When operating under the Red Hat 9.0 Linux environment, the Sun Java SDK 1.4.2 development environment and the Java Web Service Developer Pack 1.3 were used, with the XML parser upgraded to Xerces-J 2.6.2 (as opposed to the "stock" version 2.3.0+ which came with the pack).

Server 2 (Front-end server)

When operating under the Windows 2000 Server environment, Microsoft .NET Framework and SDK 1.1 were used. When operating under the Red Hat 9.0 Linux environment, the Sun Java SDK 1.4.2 development environment and the Java Web Service Developer Pack 1.3 were used with the XML parser upgraded to Xerces-J 2.6.2 (as opposed to the "stock" version 2.3.0+ which came with the pack).

Network Simulator (Router)

This configured server operated the OpenBSD 3.5 operating system with packet filter (pf) that allows bandwidth management. OpenBSD is a freely available, multi-platform, UNIX-like operating system. Network analysis tools, such as tcpdump, reported on network characteristics and performance. This allowed modeling of system performance based on the bandwidth of the network.

Client Computer

The client computer had the Microsoft Windows 2000 Advanced Server operating system installed. It utilized the Internet Explorer 6.0 Web browser to interact with the Web interface and display information assembled by the front-end server.

XML appliance

The Sarvega appliance comes with a specialized development suite (XESOS Studio) that allows easy configuration of the XML workflows and schema uploads. Sarvega XESOS Studio version 1.5 for XESOS 4.6 system software was used for appliance testing.

Test Results

Summary

The test results showed that the two largest time components were XML validation and DOM processing. These two tasks make up roughly 99% of the transaction time. Most of the test cases utilized XSL transformations for rendering and did not include a DOM processing step. The test harness, a testing tool, was comprised of a test driver and a test comparator, which measured the processing time of a transaction consisting of a set of steps consistent with common XML tasks. For these test cases, the schema validation consumed the vast majority of the processing time.

The tests confirmed the benefit of using a subset schema, because it decreases the latency time of the validation step, when compared with using the full GJXDM reference schema. The time measurement benefit was proportional to the amount of the data model included in the subset schema. As the subset schema increases in size toward the full data model, the expected performance improvement from using a subset schema decreases.

As an example of the use cases tested, the Incident Report offered the highest return on investment for utilizing a subset schema in place of the full GJXDM. The validation performance of the subset schema for incident was 47% of the full GJXDM reference schema. It is important to note that one variant of the AMBER Alert subset schema incorporated 80% of the full reference schema. With this subschema, the full transaction performance only yielded a 20% improvement. The relationship between the size of the subset schema and performance appears to be linear with the upper bound set by referencing the full GJXDM schema.

Production implementations of the GJXDM must carefully consider the performance impacts of XML validation. Due to processing overhead, large models, such as the GJXDM, rarely conduct validation during normal transactional processing and therefore validation benchmarks do not correctly gauge the practical performance of real solutions built on .NET or Java. In normal production scenarios, smaller sections of the model are used or validation is disabled entirely to avoid a negative performance impact. The benchmarks demonstrated that neither of these two environments delivered performance characteristics that would be usable in a production environment.

Currently available off-the-shelf validation tools validate at the point of both origin and reception, which is not practical, regardless of the platform. Many organizations will choose to turn off validation, upon completion of the testing, so the overhead generated by the use of the GJXDM is negligible.

As an alternative, new devices have recently reached the market that are capable of off-loading the parsing and validation work from the Web server to a "plug and play" appliance. These devices conduct parsing and validation in firmware

and frequently include other functions such as authentication management. XML Appliance platform tests conducted used a device made by Sarvega. With this device the elapsed time for parsing, validation, and transformation was reduced 100-fold with full validation against the GJXDM. The XML Appliance completed the total transaction in less than 1 second.

The data content size of the transaction does not appear to have an appreciable impact on performance. Several of the test cases provided data payloads of various sizes in order to compare theories regarding the influences to the overall performance of the XML parser using the GJXDM data model. In addition, the tests indicate that data size has a minor influence of the magnitude of the XML validation and DOM processing. From the limited number of data points, it appears that the relationship of data size and performance is more logarithmic than linear.

"Flattening" the data model, by collapsing the inheritance and complex type trees, actually degrades overall performance. This approach requires that the entire file is loaded all at once, instead of on demand, which appears to negate any performance gains offered by XML parser optimization.

Reducing the tag names within the data model and the instance yields a measurable improvement in performance depending on the test platform. However, given that this idea violates the rules of use for the GJXDM reference schema, obscures the meaning of the content, and significantly impacts interoperability, it is likely that the cost of this technique outweighs the performance benefits.

The speed of the network does not appreciably influence performance until the bandwidth was constrained to 9,600 bps. All of the executed test cases measured the effects of network bandwidth on overall performance.

Referencing the full GJXDM from a remote location has a negative impact on performance. One variation on the network testing had the instance schema referencing the full GJXDM from a remote location. This test added the overhead of pulling the full data model across the provided network link to the overall transaction time. It is important to note that the data model size is nearly 2Mb, which may significantly affect a majority of network bandwidth points.

The test results consistently show a significant difference in observed performance between the tested .NET and J2EE XML parsers. The Microsoft .NET XML parser (managed code version) was determined to be significantly slower than the J2EE parser (Xerces-J 2.6.2) for most tests. This difference is primarily attributed to the differences in the schema parsing behaviors; it appears that the Microsoft .NET parser always parses and loads the entire schema, while the Xerces only operates on the schema elements actually used in the instance.

Overall, the test results confirm several key best practices for utilizing the data model and illustrate the effects of several suggested changes to the data model. The results indicate several suggestions for further research and offer numerous lessons learned.

General Analysis

Validation

The test cases demonstrated that a significant portion of the transaction time was consumed by validation of the XML instance document against either the GJXDM reference schema or a subset schema.

The test data show performance improvements by substituting a subset schema in place of the full GJXDM reference schema.

Validation time effects appear in relation to the overall size and complexity of the data model or subset schema. The test cases included scenarios utilizing the GJXDM 3.0 initial operating release as well as the pre-release JXDD 3.0.0.0 version. It is important to note the significant increase in validation time when comparing pre-release to production GJXDM. Further research suggests this is the result of increased complexity in both depth and breadth in the production schema. The production GJXDM schema includes several additional levels of inheritance as well as a number of new and expanded objects.

The test data suggests that the XML parser(s) provide some level of optimization. This optimization appears to yield better results when elements, objects, and whole files are loaded on demand rather than all at once. This appears to explain why flattening the data model, by eliminating inheritance, and aggregating the subset schema and instance schema into a single file degrades performance. This idea also supports the GTRI suggestion for deploying the subset and instance schema. Figure 3, from the GTRI developer's workshop, illustrates the proposed document structure:

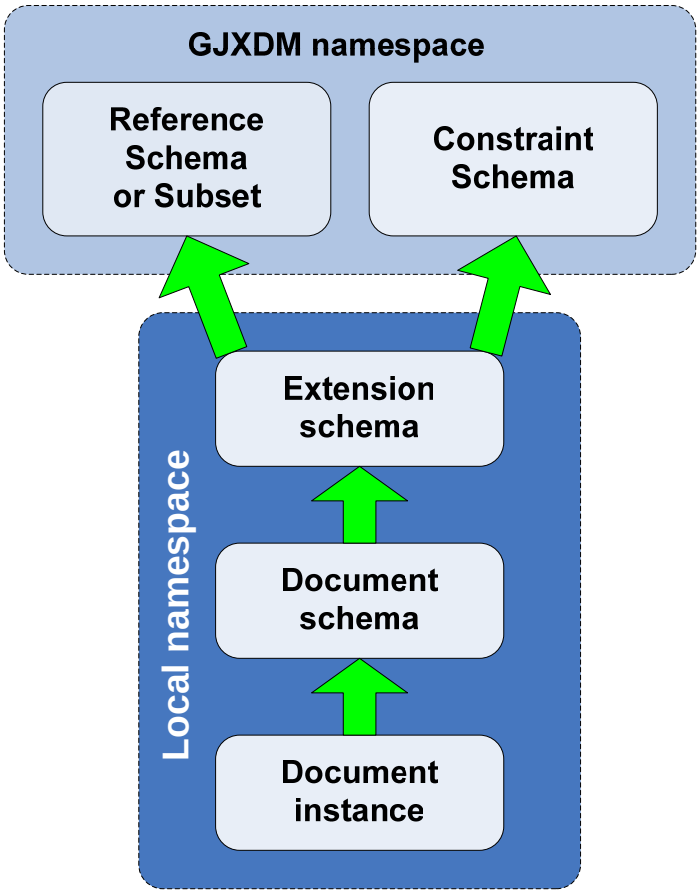

Figure 3 – GTRI proposed document structure

The research also suggests some key best practices adopted by the GTRI team have important performance implications:

- The subset schema must maintain a list of elements, types, and attributes that appear in the same order as the full GJXDM. Several of the test cases including Incident Report maintain the order of elements within types as required, but did not maintain the same ordering of element, type and attribute definitions as the full GJXDM. This resulted in a parser-dependent increase in validation times.
- Declare the subset schema in the GJXDM namespace. It should not reference the proxy schemas. The subset schema should reuse the proxy schemas by inclusion into the subset schema file.
- References to the actual proxy schemas resulted in loading multiple copies of the schema elements. The performance degradation results from the inflated size of the subset schema in memory.

Platform Findings

As described fully in the Methodology Section of this report, a variety of test transactions were constructed and executed in order to observe a number of characteristics of real world GJXDM transactions. Each test transaction executed on two platforms identified as the J2EE platform and the .NET platform.

The J2EE platform consisted of the Sun Microsystems Java Software Development Kit (SDK) to develop the test applications, which were operating on Red Hat Linux Version 9.0. A Java Web Services Developer Pack (JWSDP) version 1.3 and Java API for XML Processing (JAXP) implemented the data exchange. JAXP was upgraded to include the 2.6.2 version of the Xerces-J XML parser, which includes better support for Schema validation. The JWSDP's distribution of Tomcat Version 5.0 Web server is the servlet container for the Java test applications.

The .NET platform consisted of C# test applications developed using the Microsoft .NET Framework 1.1 operating on Windows 2000 Advanced Server. The front-end and back-end server applications ran on Microsoft Internet Information Server (IIS) version 5.0. ASP.NET was used to develop the Web interface on the front-end server. Both servers used .NET Web Services API classes to communicate.

The XML Appliance platform was essentially an add-on component for both the Microsoft platform and J2EE platform. The XML Appliance was incorporated with each platform and was configured to perform XML functions that previously had run on the front-end and back-end servers of both platforms. The appliance was used in proxy mode, i.e., the XML traffic was directed at the appliance instead of the 'real' recipient, and the appliance would only pass the XML content through to the recipient if it was valid, otherwise an error was generated and transaction aborted.

The Platform Test evaluated the differences in performance observed between the two platforms when running the same kind of transactions. The primary findings regarding the platform tests are:

- The J2EE platform is overall notably faster than the .NET platform for all transaction types, categories, and variations.
- The .NET platform is faster in many of the discrete transaction phases, but not during the validation phases. The validation phase of the transaction is a notable performance bottleneck for .NET.
- The XML Appliance platform demonstrated that a significant improvement in performance could be achieved by off-loading XML tasks from either the Microsoft platform or the J2EE platform to a purpose-built device.

Summary Data

This section summarizes the data that supports the platform findings. A detailed analysis of the data follows.

Transaction Time

Figure 4 depicts the total transaction time in seconds of a representative variety of transactions. This figure has grouped the XML Appliance platform to the left, J2EE platform in the center, and the .NET platform to the right of center. The intent of this graphic is to depict that the exact same transactions executing on the XML Appliance platform perform much more efficiently than the J2EE platform and that the J2EE platform perform notably faster than on the .NET platform. Beginning at the far left of the diagram, the total transaction time for the transaction identified as "XMLA-JAI" reports approximately 15 seconds. The transaction identified as "J2EE-JR1" reports 402 seconds. The transaction identified as ".NET-NR1" is the .NET platform version of this same transaction. It reports a total transaction time of 2,466 seconds.

The supporting data represented by this figure buttresses the finding that an XML Appliance is superior in performance and the J2EE platform is notably faster than the .NET platform for all transaction types, categories, and variations.

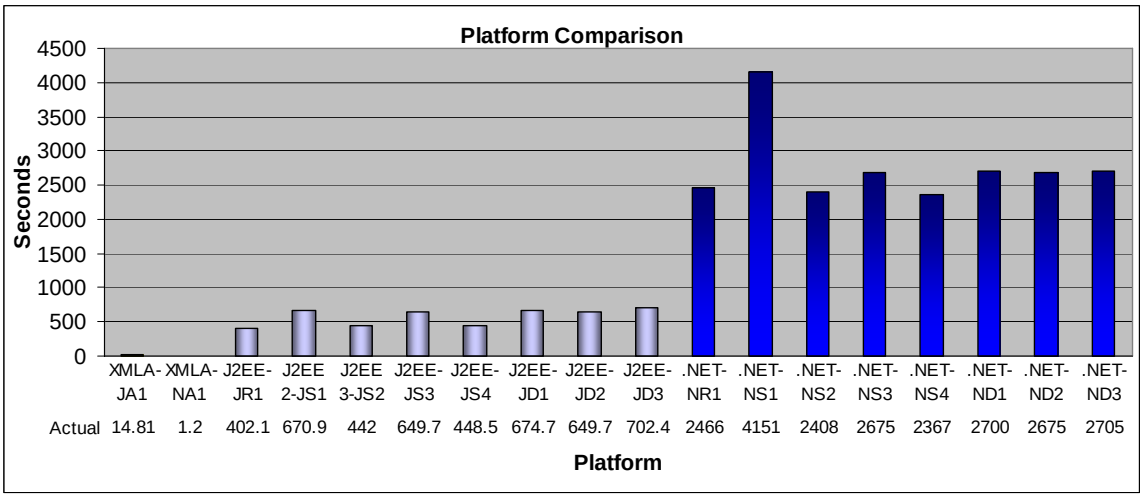

Figure 4 – Platform comparisons

Transaction Stages

During the execution of each transaction, various stages of the transaction captured and recorded performance measurements. A defined time measurement for each of four steps provided a clear definition in conduction these analyses. A definition of each of these four phases follows in Table 1:

Table 1 – Time measurement and descriptions

Time Measurement	Description
Step 1	Sending server request validation time
Step 2	Sending server GJXDM instance transformation time
Step 3	Receiving server GJXDM instance transformation time
Step 4	Receiving server request validation time
Total Communication	This value is calculated time in order to represent all of the data communication time associated with the transaction
Total Overhead	This is the receiving and sending server overhead time which includes the server setup and startup time for each transaction

Figure 5 depicts the percentage of time spent executing each of the testing phases, using the same transactions as Figure 4 on the J2EE and .NET platforms. This figure has grouped the J2EE platform data to the left and the .NET platform to the right.

Beginning at the far left, the transaction identified as "XMLA-JA1"depicts the fact that the vast majority of time is spent during the overhead processing step. It is important to note that due to the high speed of this transaction, this total setup time is just over one second. The transaction labeled "XMLA-NA1" spends the vast majority of its time in the communication phase of the transaction. Also noted, due to the high speed of this transaction, was this total communication time is under 1/10th of a second.

Moving right to the first J2EE platform transaction, "J2EE-JR1" spent the majority of the time in steps 2 and 3 (described above) with some minor time spent in steps 4 and total overhead. The transaction identified as ".NET-NR1" is the .NET platform version of this same transaction is located near the center of the chart. The .NET version of this transaction also spends the majority of its time in steps 2 and 3 with no displayable amount of time in any other step.

The supporting data represented by this figure supports the finding that the .NET platform is faster in many of the discrete transaction phases, but not during the validation phases.

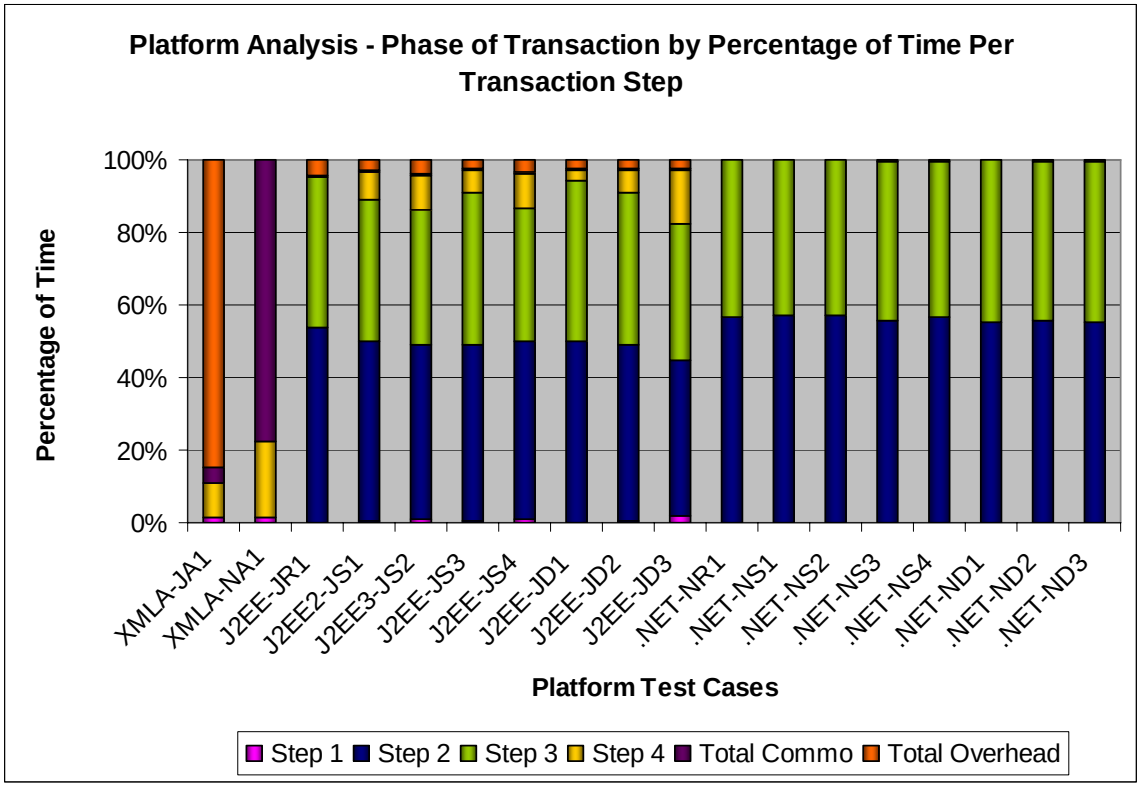

Figure 5 – Transaction step analyses

Detailed Analysis

A variety of test scenarios and supporting transactions were constructed and executed in order to observe a number of characteristics of real world transactions utilizing the GJXDM. These test scenarios and transaction were:

- AMBER Alert
- Field Report
- Arrest Incident Report
- Inmate Record
- RAP Sheet

All of these test scenarios and transactions executed on two platforms identified as the J2EE platform and the .NET platform. Performance analyses conducted on all scenarios, transactions, permutations, and combinations of test transaction characteristics determined that there is a consistent profile of performance. A representative sample of transactions shows these observations. The entire raw data set evaluated in order to establish this representative sample is contained in Appendix A of this report.

The following selected transactions are representative samples. For this platform analyses, a unique test identifier represents each transaction type. A description of the transaction along with its identifier follows in Table 2:

Table 2 – Selected representative sample

Test ID	Platform	Transaction	Schema	Data Size
XMLA-JA1	XML Appliance-J2EE	AMBER Alert	Full Release	Average
XMLA-NA1	XML Appliance-.NET	AMBER Alert	Full Release	Average
JS1	J2EE	Incident	Aggregate	Average
NS1	.NET	Incident	Aggregate	Average
JS2	J2EE	Incident	Full	Average
NS2	.NET	Incident	Full	Average
JS3	J2EE	Incident	Full	Average
NS3	.NET	Incident	Full	Average
JS4	J2EE	Incident	Full Local	Average
NS4	.NET	Incident	Full Local	Average
JD1	J2EE	Incident	Full	Minimal
JD2	J2EE	Incident	Full	Average
JD3	J2EE	Incident	Full	Large
ND1	.NET	Incident	Full	Minimal
ND2	.NET	Incident	Full	Average
ND3	.NET	Incident	Full	Large
JP1	J2EE	AMBER Alert	Pre Release	Average
JR2	J2EE	AMBER Alert	Full Production	Average
NR1	.NET	AMBER Alert	Pre Release	Average
NP1	.NET	AMBER Alert	Full Production	Average

Performance testing during discrete phases in the processing of each transaction provided a wide variety of performance measurements. After a review of all the data, it was determined that the processing time in elapsed seconds was the most significant evaluation measurement for comparisons between the platforms.

Schema Approaches

In order to focus the analyses on platform differences and eliminate the variability of schema usage, the first crucial analysis compared the impact that different schemas might have on the performance on each platform. Figure 6 depicts the performance of both platforms operating test scenario transactions using a variety of different XML schema approaches by presenting a side-by-side comparison of the J2EE and .NET platforms.

The intent of this graphic is to present that the same transactions executing on the J2EE platform perform notably faster in terms of total seconds than on the .NET platform. Beginning at the far left of the diagram, the total transaction time for the transaction identified as "JS1" reported as 671 seconds. The transaction adjacent to it, identified as "NS1", is the .NET platform version of this same

transaction. It reported a total transaction time of 4,151 seconds, over 600% increase.

This diagram depicts four different schema samples with each platform compared side by side. In all cases, the J2EE platform performance is notably faster than the .NET platform performance. This observation leads to a secondary finding that the schema approach employed does not appear to have an impact on performance when comparing platforms.

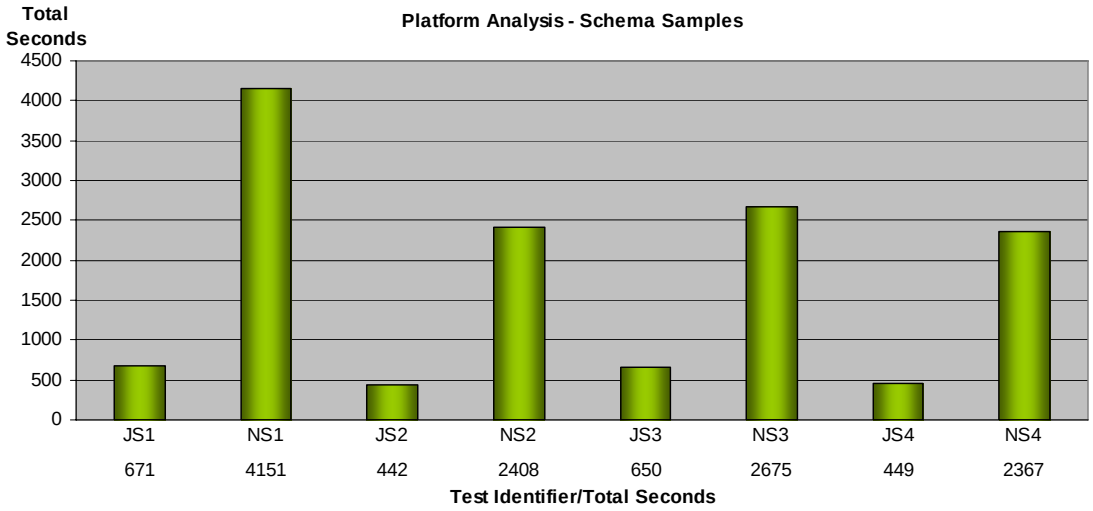

Figure 6 – Platform analysis with various schemas

Payload Size

In order to focus the analysis on platform differences and eliminate the variability of data size (payload), the second crucial analysis was to evaluate the impact that different payload sizes might have on the performance within each platform. Figure 7 depicts the performance of both platforms operating test scenario transactions using a variety of different payload sizes.

Figure 7 groups the J2EE platform to the left of center and the .NET platform to the right of center. The intent of this graphic is to depict that the same transactions executing on the J2EE platform perform notably faster in terms of total seconds than on the .NET platform. Beginning at the far left, the total transaction time for the transaction identified as "JD1" reported 675 seconds. The transaction identified as "ND1" (located just right of center in the diagram) is the .NET platform version of this same transaction. It reported a total transaction time of almost 2,700 seconds, a 400% increase.

This diagram depicts three different data size (payload) transactions as previously defined. In all cases, the J2EE platform performance is notably faster than the .NET platform performance. This observation leads to a secondary

finding that the data content size of the transaction does not appear to have an impact of performance when comparing platforms.

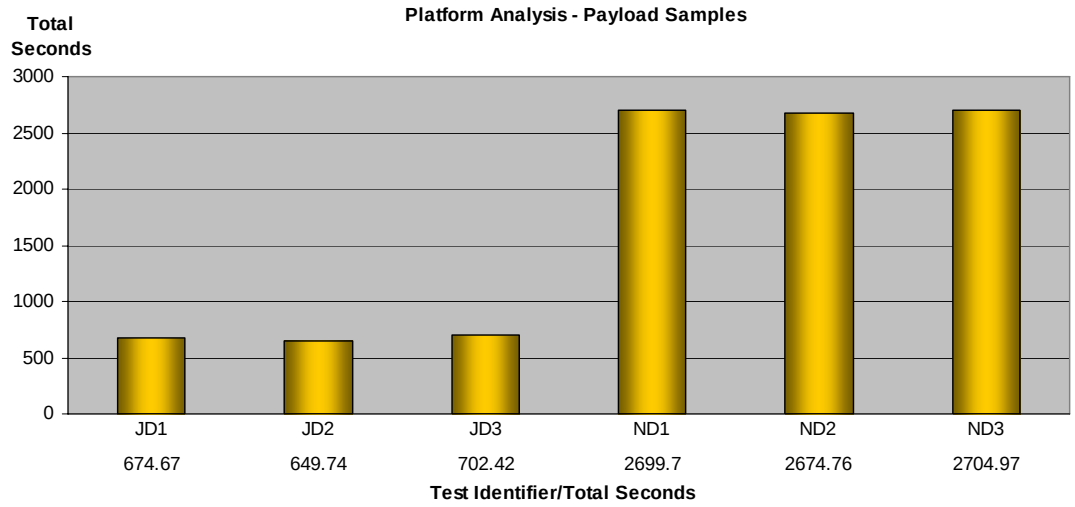

Figure 7 – Platform analysis with various data sizes (Payload)

Work Location

The third crucial analysis was to evaluate performance based on work location (step) when comparing each platform. Figure 5 graphically depicts the percentage of time that each transaction spends executing at various phases of the transaction. Table 3 provides the detailed performance data for each of the transactions as depicted in Figure 5.

Platform Comparisons

Table 3 provides a row-by-row comparison of the performance data for the XML Appliance, J2EE, and .NET platforms. Displayed is the time in seconds for each step of the transaction in successive columns moving left to right with the sum of the steps presented in the far right column. The intent of this table is to accurately report the elapsed time consumed for each step of the transaction and compare the same transactions executing on each platform.

The observation and resulting primary finding is that adding the XML Appliance to a platform speeds up all phases of the transaction with the majority of time being spent in overhead (for the J2EE variant) or data communication time (for the .NET variant).

The .NET platform is notably faster in its execution of steps 1 and 4. Generally, the .J2EE platform is significantly faster in its execution of steps 2 and 3.

Table 3 – Step comparison of the J2EE and .NET platform performance data

Test ID	Total Time, Seconds	Total Processing Time	Step 1	Step 2	Step 3	Step 4	Total Comm-unication	Total Over head
XMLA-JA1	14.81	1.65	0.22	0.01	0.00	1.42	0.63	12.53
XMLA-NA1	1.20	0.27	0.02	0.00	0.00	0.25	0.93	0.00
J2EE-JR1	402.08	384.36	0.21	216.27	165.87	2.01	0.58	17.14
J2EE2-JS1	670.92	650.15	3.17	333.01	262.57	51.40	2.92	17.85
J2EE3-JS2	442.01	423.34	3.32	212.60	165.96	41.46	2.08	16.59
J2EE-JS3	649.74	630.80	3.35	314.21	272.57	40.67	2.64	16.30
J2EE-JS4	448.50	431.64	3.29	220.99	164.80	42.56	2.13	14.73
J2EE-JD1	674.67	655.84	0.56	336.32	299.96	19.00	1.55	17.28
J2EE-JD2	649.74	630.80	3.35	314.21	272.57	40.67	2.64	16.30
J2EE-JD3	702.42	682.64	12.77	301.62	265.60	102.65	3.22	16.56
.NET-NR1	2,466.36	2,463.72	0.02	1,402.20	1,061.06	0.44	0.88	1.76
.NET-NS1	4,150.81	4,149.39	0.03	2,370.11	1,776.01	3.24	1.40	0.02
.NET-NS2	2,407.50	2,406.40	0.03	1,374.86	1,028.33	3.18	1.08	0.02
.NET-NS3	2,674.76	2,673.66	0.03	1,496.10	1,171.86	5.67	1.08	0.02
.NET-NS4	2,366.99	2,363.19	0.33	1,337.09	1,020.21	5.56	3.78	0.02
.NET-ND1	2,699.70	2,698.84	0.02	1,495.73	1,199.01	4.08	0.84	0.02
.NET-ND2	2,674.76	2,673.66	0.03	1,496.10	1,171.86	5.67	1.08	0.02
.NET-ND3	2,704.97	2,703.47	0.26	1,490.83	1,204.40	7.98	1.48	0.02

NOTE: Elapsed time in seconds during each phase of transaction

GJXDM Version Comparison

When the performance testing first began, the test team collected data using the JXD 3.0.0.0 pre-release version. Later, the GJXDM 3.0 initial operating release became available and the test team ran the battery of tests using the 3.0 version of the GJXDM. As the team started to update their performance data with the new data gathered using the 3.0 release, they noted a significant performance anomaly.

Figure 8 depicts a representative example of the performance of both platforms operating test scenario transactions using the JXDD pre-release and the GJXDM initial operating release. Working left to right, this figure first presents a side-by-side comparison of the J2EE platform first executing the pre-production release and then the initial operating release of the GJXDM. As the figure depicts, the transaction identified as "JP1" operating on the J2EE platform consumes 129 Total Seconds using the pre-release GJXDM. That same transaction identified as "JR1" consumes 402 Total Seconds using the initial operating release of the GJXDM. This is an increase of approximately 311% for the J2EE platform when comparing the pre-release version to the initial operating version.

Looking further to the right of Figure 8, there is a side-by-side comparison of the .NET platform executing the pre-production release and then the initial operating release of the GJXDM. As the figure depicts, the transaction identified as "NP1" operating on the .NET platform consumes 102 Total Seconds using the Pre-release GJXDM. That same transaction identified as "NR1" consumes 2,466

Total Seconds using the initial operating release of the GJXDM. This is an increase of approximately 2,417% for the .NET platform when comparing the pre-release version to the initial operating release version.

These observations led to a series of secondary findings:

- The .NET platform appears to have had similar performance as the J2EE platform when operating under the JXDD 3.0.0.0 pre-release version.
- Changes made to the GJXDM as it migrated from the pre-release version to the initial operating release version had a negative impact on performance resulting in increases in the total seconds for all transactions tested under both the J2EE and the .NET platforms.
- Changes made to the GJXDM schema as it migrated from the pre-release version to the initial operating release version had a significant negative performance impact for the .NET platform.

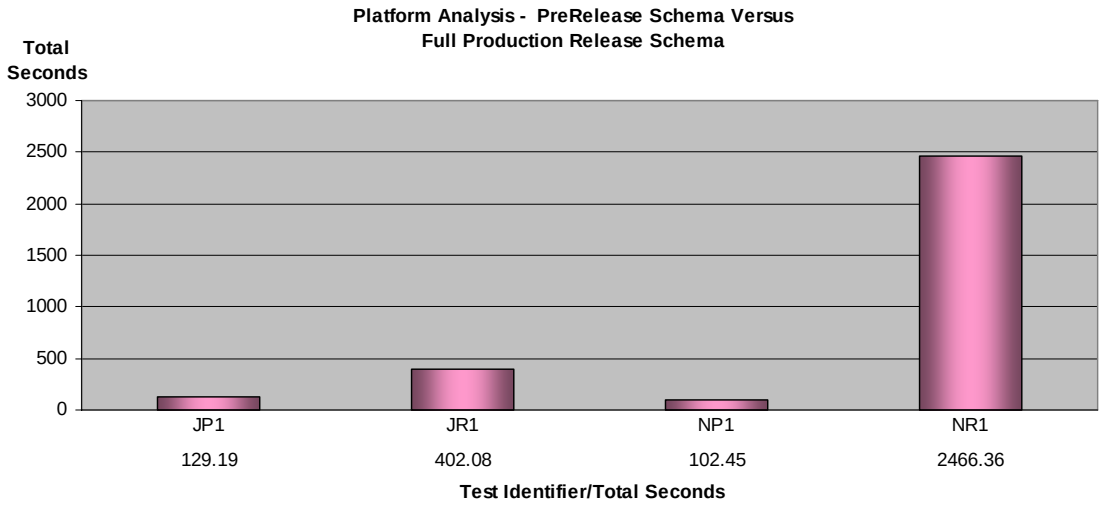

Figure 8 – Pre-production versus Initial Operating Release schema comparison

Test committee members observed the major difference between the pre-release version and initial operating release version of the GJXDM. The major difference was the conversion of a significant number of attributes, in the pre-release version, to elements, in the initial operating release. This increased the depth and breadth of the hierarchical representation of the GJXDM.

To explain the behavior exhibited by the .NET platform, the test team theorized that the .NET environment has difficulty dealing with larger, more complex schemas because of the evolution of the .NET environment's use of the MS-XML component. Reports indicate the following evolution occurred within the MS-XML component:

- MS-XML 1.0 - Only supports DTD; no schema support
- MS-XML 2.0 - Supports XDR (reduced Tagname Schemas)

- MS-XML 4.0 - Support for XSD (XML Schema Definition Standard)

The XDR reduced tagname schema language represents Microsoft's de-facto schema language and is mostly not compatible with XSD. The W3C standards body rejected many of the design elements of XDR in favor of XSD. Because MSXML 3.0 and prior versions do not comply with the standard, Microsoft made the decision to ship version 4.0 with two code paths. The two code paths included a default code path to support the non-standard 3.0 version and a W3C compliant code path.

Since the performance of the .NET platform was very similar to the J2EE platform using the JXDD pre-release, the team theorized that the observed degradation in performance of the .NET environment appears to be associated with its use of MS-XML Version 4.0 and is not inherent in the .NET environment itself. Additionally, some of the existing .NET platform tools appear optimized for the earlier generation DTD approach and therefore not optimized for a schema approach.

The team believes that once the MS-XML portion of the .NET platform is in full compliance and optimized to support complex schemas, that the performance of the .NET platform will be competitive with the J2EE platform.

Although validation of these theories is incomplete, it is important that this report address the initial thoughts of the test committee on this issue.

Use Case Findings

Use case scenarios provide real-world justice examples for the performance-testing project. Presented in an organized, systematic framework, they document exchange requirements and components in commonly understood language. The team selected a range of scenarios and schema construction methods to capture performance statistics for exchanges of varying complexity, from simple messages to complex data payloads. The selected use cases take advantage of completed development work. Additional benefits of this approach included the opportunity to compare schemas based on earlier versions of the model with the initial operating 3.0 release as well as to capture extensive metrics for the highest priority exchanges.

The primary findings regarding the Use Case Scenarios are:

- Schema design was a critical factor in processing speed; there were significant differences between different methods.
- While processing resources were allocated differently in each scenario, the schema approach or exchange platform did not significantly affect the performance ratios within an individual use case scenario.

Use Case Content

This section summarizes the intent and content of each use case in order to provide a high-level explanation of key schema components. An analysis of the test data findings and implications follows in the next section.

AMBER Alert

AMBER[1] Alert messages containing information about a specific child (or disabled adult) abduction are transmitted from an authorized law enforcement agency to public and private agencies for distribution to the public. It circulates to state emergency notification systems, departments of transportation (for publication on highway signs), broadcast media, and other designated agencies, depending on the specific circumstances of the incident and the specific information sharing agreements in place. While the exact content of the message can vary, the National Center for Missing & Exploited Children suggests meeting the following three criteria before activating an AMBER Alert:

- Law enforcement confirms a child has been abducted
- Law enforcement believes the circumstances surrounding the abduction indicate that the child is in danger of serious bodily harm or death
- There is enough descriptive information about the child, suspected abductor, and/or suspect's vehicle to believe an immediate broadcast alert will help obtain the safe recovery of the missing person.

The alert information that is distributed after meeting these criteria can include descriptions and pictures of the missing child, the suspected abductor, a suspected vehicle, and any other available information that can be used to identify the child and suspect.

Since AMBER Alert messages are usually low volume, intermittent occurrences, this scenario focused on testing the impact of different types of schema construction with a consistent message size. Five different schema formats tested on each platform:

- *Aggregate (Release)* incorporates a copy of the full GJXDM 3.0 initial operating release (3.0) into a single reference document (instead of using import or include statements), limiting the use of import statements for the proxy schemas.
- *Aggregate Flat (Release)* focused on evaluating the impacts of abstraction by using 'groups' and 'attributeGroups' to eliminate 3.0 duplicated elements and create a sectioned data model.
- *Full (Release)* references the whole 3.0 schema, which is bundled with the test case in a separate directory.

[1] America's Missing: Broadcast Emergency Response (http://www.AMBERalert.gov/)

- *Full (Pre-Release)* is the same as Full (Release) except that it references the 3.0.0.0 pre-release schema instead of the 3.0 initial operating release.
- *Subschema (Release)* references a reduced version of the 3.0 schema that was created manually.
- *Subschema (Optimized)* references a schema produced using the GTRI subschema generator.

The message content was defined with a schema that was assembled from Incident, Subject, and Vehicle elements as defined in the GJXDM schemas along with two locally defined elements (AMBERAlert and MessageSentDateTime). The schemas obtained all referenced schemas locally and did not use type substitution or constraints.

Arrest/Incident Report

The Arrest/Incident use case describes a criminal event requiring law enforcement intervention that ultimately culminates in the arrest of one or more subjects. Test exchanges focus on a single scenario, where a law enforcement agency transmits the arrest incident information to the prosecutor's office after completing the booking process of a subject.

An instance of this exchange is comprised of single Incident containing a narrative summary of the event, followed by repeating Person, Charge, and Property sections. An important distinguishing feature of these scenarios is the use of the dedicated GJXDM reference elements to establish relationships between persons, property, and documents. Four different schema formats were tested on each platform:

- *Aggregate (Release)* directly incorporated all of the GJXDM types from the 3.0 release into a single reference document.
- *Subschema (Release)* referenced a local subset of the GJXDM.
- *Full External (Release)* referenced the full 3.0 release from the OJP Web site.
- *Full Local (Release)* was the same as the Full External version, except that the 3.0 release import namespace was stored locally.

All schemas were assembled from Document, Incident, Booking, Case, and Arrest Charge elements as defined in the GJXDM along with a number of locally defined elements, including forty-nine sets of code enumerations. The schemas did not use type substitution or constraints.

Inmate Record

The inmate record is a person's history containing the subject's name, address, and identifying information together with detention information for specific offenses and sentences. A complete inmate record contains the subject's current

name and address, all name aliases, addresses associated with each name alias, and detention information that includes supervision information along with the associated court case and sentenced offense. This exchange represents communication between law enforcement and detention agencies with a records repository and includes a number of add, update, and query (display) transactions utilizing different combinations of the exchange components. These transactions were:

- *Alias Name Add* - The law enforcement agency records a new subject alias name to the list of existing name alias records.
- *Alias Address Add* - The law enforcement agency adds a new subject alias address to the list of existing address alias records.
- *Current Address Update* - The law enforcement agency updates the subject's current address, which results in the movement of the original address to a list of the subject's alias address records.
- *Detention Add* - The detention facility adds a new detention record to the inmate detention history. This represents the Person portion of the Inmate subschema along with a single Detention element and is a repeating element in Inmate.
- *Detention Update* - The detention facility provides a revised detention record, which updates the current inmate detention record.
- *Person Update* – An authorized agency provides an updated person record that will update the current person record. It does not contain any repeating elements and the actual size of the record should vary only by the size of the data transmitted.
- *Basic Query* - In response to a request (not included in the testing scenarios) from an authorized agency, the records repository will provide a person record without the alias, alias address, or detention records attached.
- *Full Query* - In response to a request from an authorized agency, the records repository provide a full inmate record that includes one or more of the repeating alias, alias address, and detention records.

Although schema frameworks were tested on both platforms, they were more limited initially than in other use cases due to the large number of exchange transactions. Additional schemas added using the GJXDM 3.0 initial operating release provided more comparison that is consistent across use cases. The five different schemas were:

- *Aggregate (Pre-Release)* - includes copies of all elements and dependency trees for referenced JXDD elements, based on the 3.0.0.0 pre release, into a single reference document instead of using import or include statements.
- *Aggregate (Release)* - consolidates the 3.0 GJXDM and Inmate Record reference document into a single schema.

- *Full (Pre-Release)* - imports the 3.0.0.0 pre-release from a local import namespace.
- *Full (Release)* - imports the full GJXDM 3.0 from a local import namespace. This reference document follows the GTRI recommended practice that uses a single root element (InmateType) based on the GJXDM document type
- *Subschema (Release)* - references local subset schema based on the GJXDM 3.0. For the Full Query transaction only, this Subschema provides comparisons to facilitate between exchanges.

The Incident schema is comprised of two locally defined complex elements: InmateType that includes the GJXDM Subject along with local elements, and DetentionType that contains the GJXDM Supervision, Case, Incident, Sentence, and SameAsRelationship elements, in addition to several other local extensions. In contrast to the Arrest/Incident Report, relationships are defined using standard XML ID and REF elements. It did not use type substitution or constraints.

RAP Sheet

The RAP (Record of Arrest and Prosecution) Sheet is a record of an individual's criminal history.

A typical RAP Sheet transaction involves the transmission of a Person Check request containing basic identifying information about subject from a law enforcement agency to a records repository. The repository will return a response indicating either no match or a report containing match information, usually multiple records for different individuals. The response also contains the original query content to enable the requesting system to match up the new data with the request. This test case involves requesting the RAP Sheet information from the back-end server and displaying it on the user's screen; a complete description of the testing transactions is located in the Test Results Primer contained in Appendix B of this report.

This test case primarily explores the effects of larger repeating data sets. The four different schemas were:

- *Aggregate (Release)* - incorporates a copy of the full GJXDM 3.0 initial operating release (3.0) into a single reference document (instead of using import or include statements) and only uses import statements for the proxy schemas.
- *Full (Pre-Release)* - is the same as Full (Release) except that it references the 3.0.0.0 pre-release schema instead of the 3.0 initial operating release.
- *Full (Release)* - references the whole 3.0 schema, which is bundled with the test case in a separate directory.
- *Subschema (Release)* - references a reduced version of the 3.0 schema.

Construction of the local container elements, (e.g. RapSheetRequest and Cycle), employs the GJXDM person identifiers, Arrest, Charge, Sentence, DisciplinaryAction, Supervision objects, and other local extensions. They use the GXJDM reference elements to document the relationships between components.

Summary Data

This analysis compared the transaction time in seconds for average data payloads on a 100M network to focus the analysis on a high-level review of any differences within and between use cases. (Network speed and data size were not included in this analysis, as it is already addressed elsewhere in this report). The time measurements were divided into three categories: total communication time (total time less overhead and processing); total overhead (startup and "garbage collection" time on the servers), and total processing (request generation, validation, parsing, manipulation, and serialization/storage) time.

There were a number of key findings from this analysis:

Schema design was a critical factor in processing speed; there were significant differences between different methods.

- The Aggregate Flat schema documented in the AMBER Alert was significantly slower than any other schema, processing, on average, 340% slower in .NET and 269% slower in J2EE. Figures 9 and 10 depict the J2EE and .NET platform tests using the average data size and a 100M network.

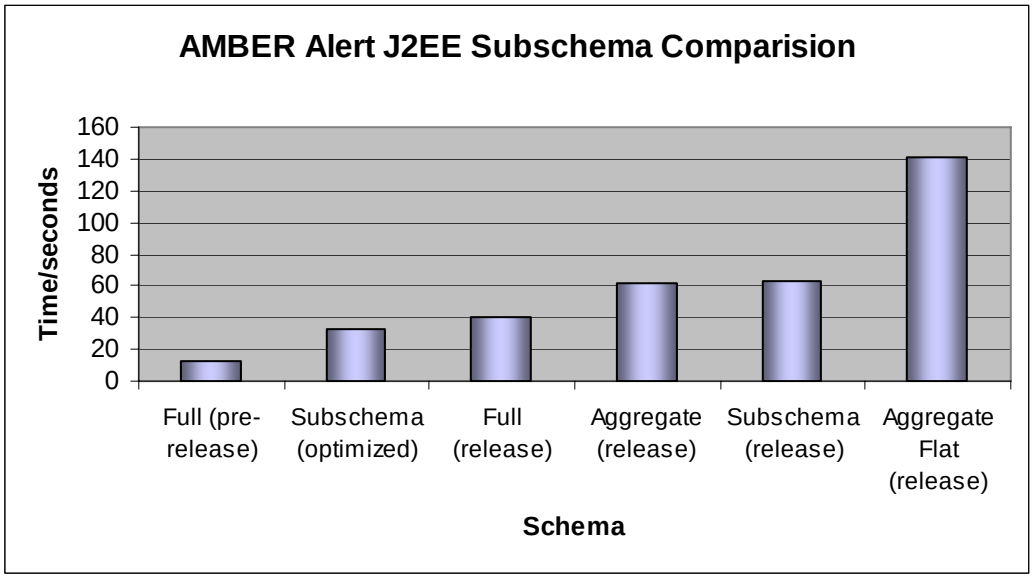

Figure 9 – AMBER Alert J2EE subschema comparison

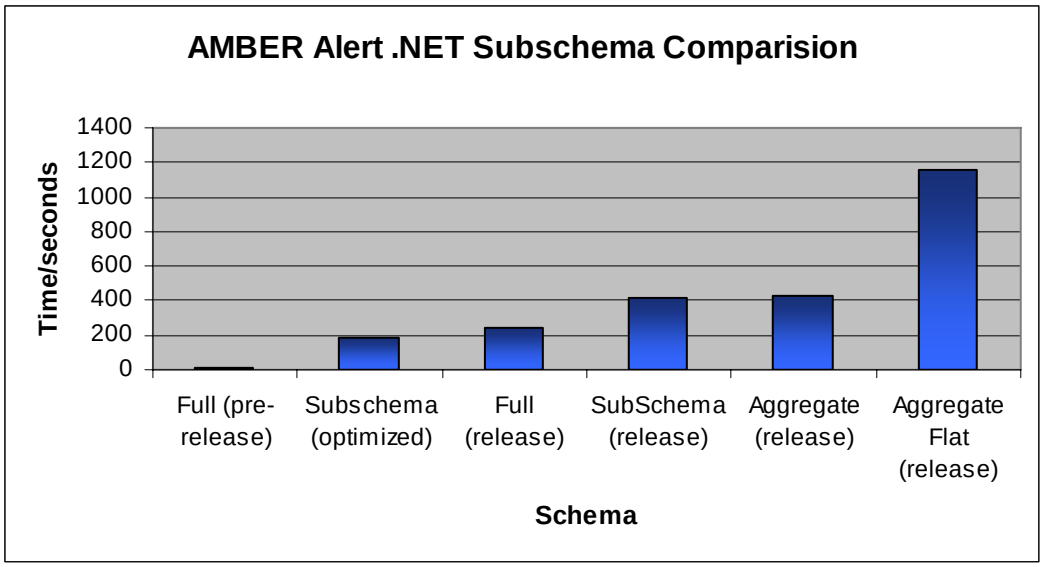

Figure 10 – AMBER Alert .NET subschema comparison

- A head-to-head comparison, depicted in Figure 11, of aggregated and imported schemas shows that performance is faster when schemas divided into smaller components.

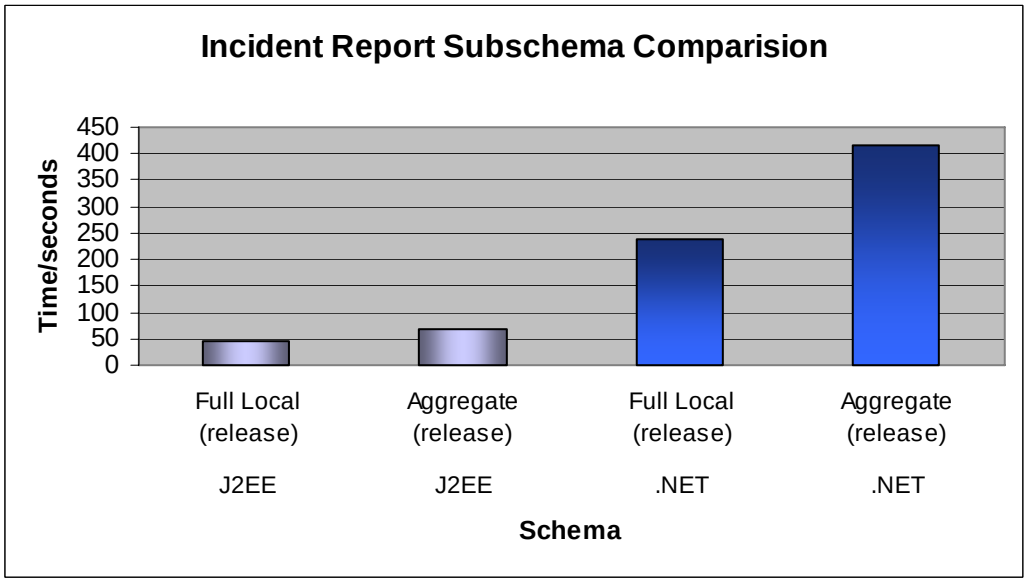

Figure 11 – Comparison between Full and Aggregate schemas

- In Figure 12, subschemas did not improve performance as much as expected, and were noticeably slower than equivalent schemas based on the full release in the AMBER Alert, Inmate Record, and RAP Sheet use cases. Overall performance was about the same in the Arrest Incident Report, although J2EE

did show improvements in the communication, request generation, and request transformation categories.

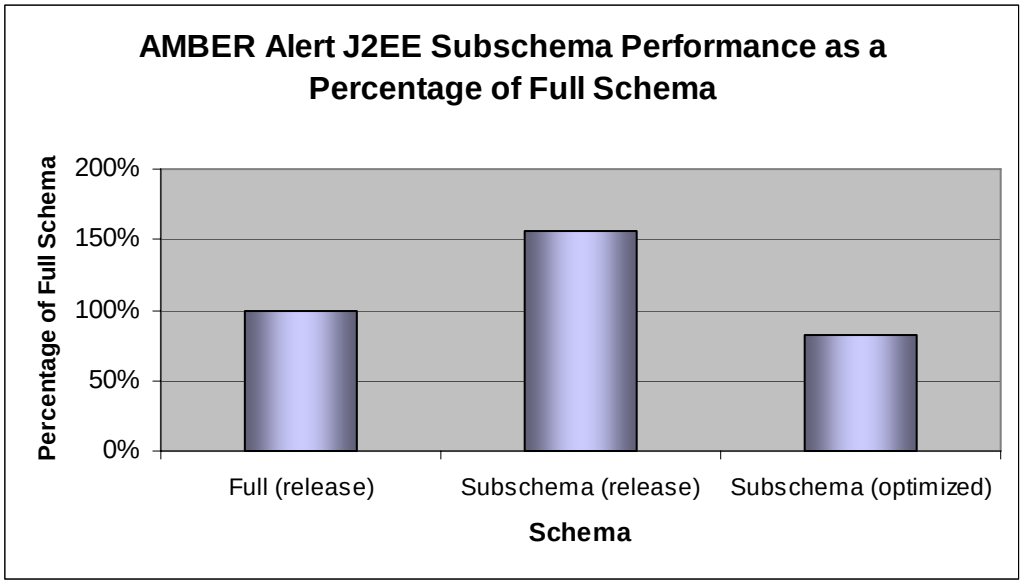

Figure 12 – Subschema performance as percentage of Full Schema

An important implication of this finding is that subschemas must be carefully designed and are not always the "silver bullet" solution to performance concerns. Additional research is required to investigate this issue further.

Processing resources allocated differently in each scenario, but performance ratios within an individual use case scenario, were essentially the same regardless of the exchange platform or schema approach.

- In Figure 13, the ratio of time spent in communication, overhead, and processing was consistent within a single scenario regardless of data size or schema type, including the pre-release and initial operating release[2] versions of the scenarios. However, the ratio of time spent in each processing step varied between use cases.

Table 4 describes the transaction type identifiers used in Figures 13 and 14.

Table 4 – Transaction type identification by platform, schema, and data size

Transaction ID	Platform	Schema	Data Size
JPM	J2EE	Aggregate (pre-release)	Minimal
JPA	J2EE	Aggregate (pre-release)	Average
JPL	J2EE	Aggregate (pre-release)	Large
JFM	J2EE	Full (pre-release)	Minimal
JFA	J2EE	Full (pre-release)	Average

[2] The subschema (release) schema numbers were slightly different, but not considered in this analysis as the testing team felt the numbers were misleading.

Transaction ID	Platform	Schema	Data Size
JFL	J2EE	Full (pre-release)	Large
NPM	.NET	Aggregate (pre-release)	Minimal
NPA	.NET	Aggregate (pre-release)	Average
NPL	.NET	Aggregate (pre-release)	Large
NFM	.NET	Full (pre-release)	Minimal
FNA	.NET	Full (pre-release)	Average
NFL	.NET	Full (pre-release)	Large

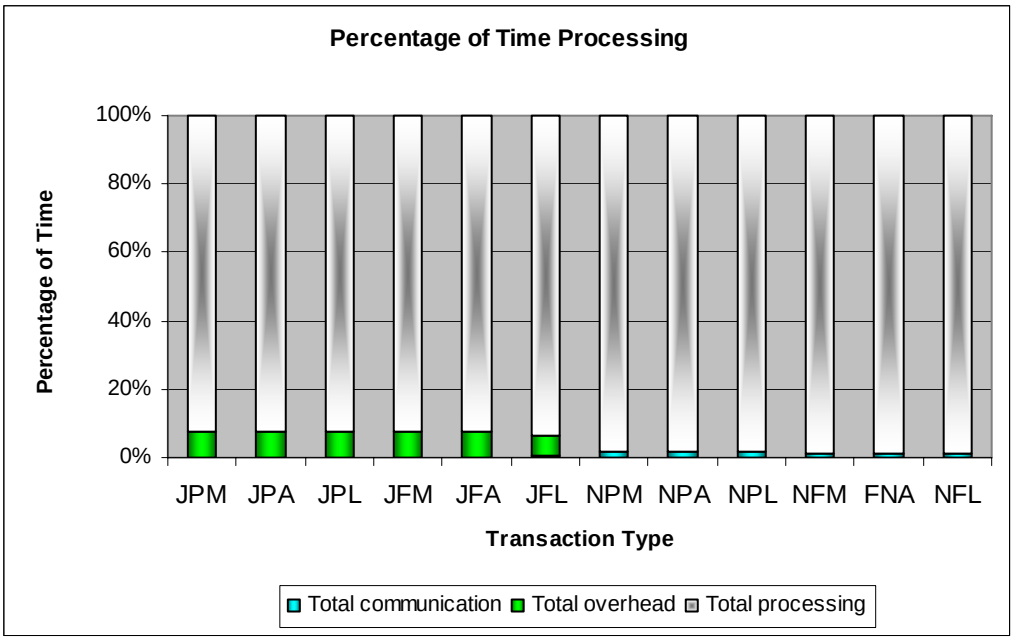

Figure 13 – Time ratios for major processing steps

- INMATE Add and Update Scenarios (Figure 14)
  - In .NET, processing time was split between request validation on the front-end and the back-end servers during the add/update scenarios
  - In contrast, J2EE, the processing time divided roughly evenly between four stages of the transaction. The request validation on the front-end server, request validation on the back-end server, DOM parsing of the request on the back-end server and DOM parsing of storage on the back-end server all consumed an equal amount of transaction percentage time. Transaction startup consumed the remaining notable time of the J2EE transaction for both the front-end and back-end servers.

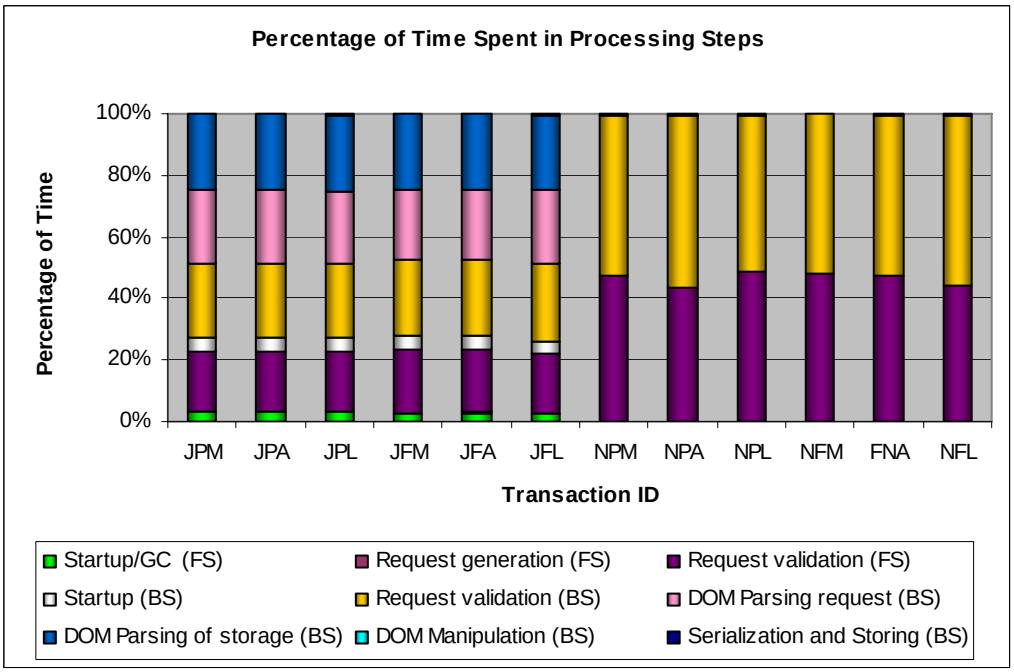

Figure 14 – Processing ratio differences between .NET and J2EE

- INMATE Query Scenarios
  - In Figure 15, query scenarios used different processing measurements. In both .NET and J2EE, processing time was split between the four validation steps, request validation on the front-end and back-end servers, and the return validation steps on the back-end and front-end servers.

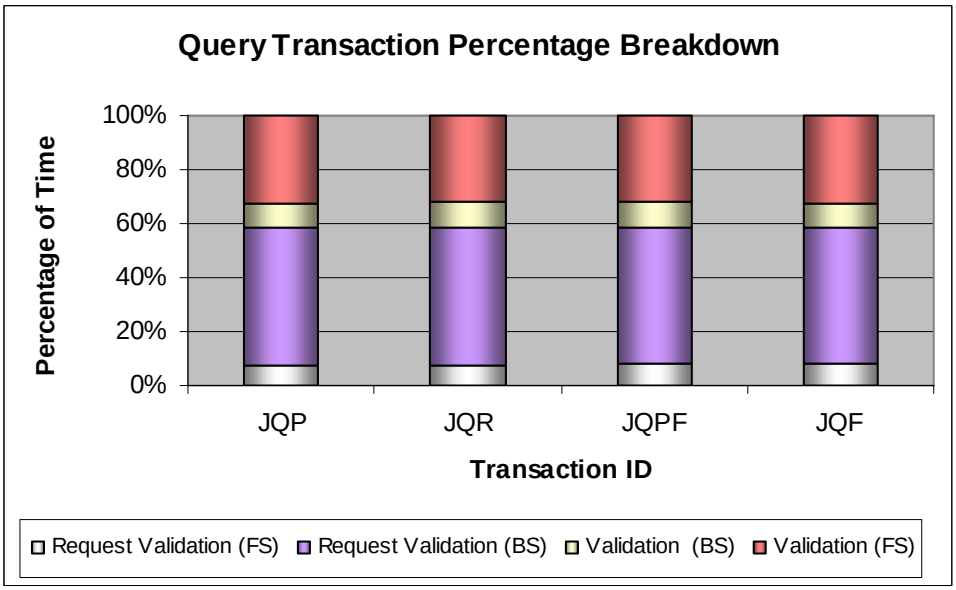

Figure 15 – Processing ratios for .NET and J2EE

Table 5 describes the transaction type identifiers used in Figures 15. All of these transactions used an Average data size.

Table 5 – Transaction type identification by platform and schema

Transaction Id	Platform	Schema
JQP	J2EE	Aggregate (pre-release)
JQR	J2EE	Aggregate (release)
JQPF	J2EE	Full (pre-release)
JQF	J2EE	Full (release)
NQP	.NET	Aggregate (pre-release)
NQR	.NET	Aggregate (release)
NQPF	.NET	Full (pre-release)
NQF	.NET	Full (release)

Data Size (Payload) Findings

Most of the test included multiple data sets of varying sizes to determine the relationship of data size to performance. For Inmate Report, only the Report itself included multiple data sets.

Typically, the data sets characterized as Minimal, Average, or Large. All test cases, except AMBER Alert, had a request data set that in some cases provided an additional data set for comparison.

Summary Data

Each of the test cases were executed with each data set and resulted in a complete set of timings for each platform. Figure 16 compares the J2EE Incident Report using both an Aggregate Schema and Subschema for Minimal, Average, and Large data payload sizes. The darker color represents the Aggregate Schema performance times and the Subschema times are the lighter color. The Aggregate performance is consistently slower for all three data payload sizes and Data Size does not appear to have a significant impact on performance.

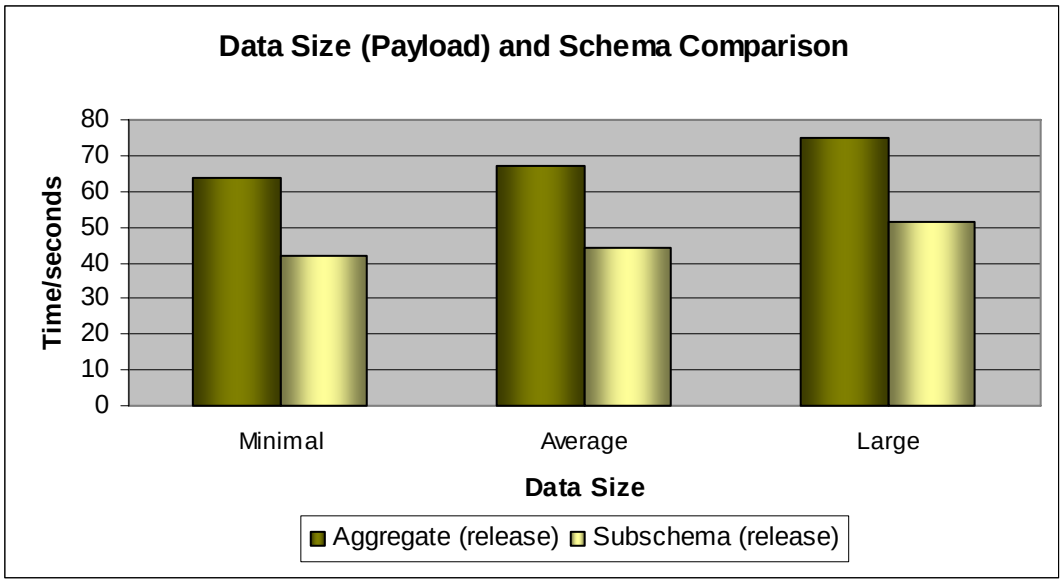

Figure 16 – Data size (Payload) and schema comparison

Analysis

From the limited set of data points, it appears that the relationship between data size and performance appears to be more logarithmic than linear. However, it is likely that this relationship might be different for very large data sets, such as the extensive "Large" data sets used in the RAP Sheet and Inmate Record transactions.

Schema Design Findings

The investigation of the schema design determined the effects of several key design considerations for the GJXDM data model. The results of this investigation confirmed that many of the GTRI suggestions are best practices.

It is important to note, that most of the findings regarding schema design were applicable to both platforms and all use cases.

Summary Data

Flattening the Data Model

A selected test case measured how the object model complexity affects performance. In addition, a specially developed subset schema collapsed the object hierarchy into a single complex type and expanded elements of complex types. The result was that every required element contained a single complex type containing elements of simple types.

The test results showed that flattening the data model significantly degraded performance. Analysis suggests that flattening the data model precluded any XML parser optimizations, and did not add any processing benefits.

In general, the tests support the GTRI suggestions for GJXDM usage and support the suggestions for structuring the schema and XML instance documents. Figure 17 is a comparison of AMBER Alert Aggregate schema versus Flat schema. On the left side of the figure are the J2EE comparisons and on the right present the .NET findings. The .NET times are more than twice the J2EE performance times.

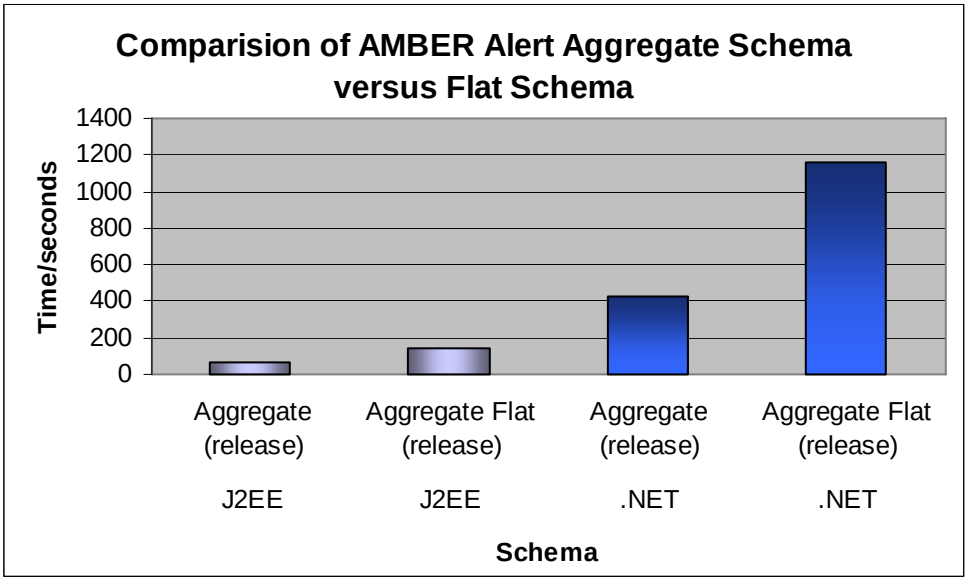

Figure 17 – Comparison of Aggregate schema versus Flat schema

Subset Schema Architecture

Research into the test results indicate that the XML parser optimizations work best when elements, object, and files can be loaded on demand rather than together at the front-end.

Analysis

The test results confirm that the GTRI suggested deployment architecture for reference and instance schemas appears to offer the best overall performance. This conclusion appears to hold true for both platforms tested and all scenarios.

In addition, the test results also confirm that the object-oriented data model offers the most opportunity for parser optimization.

The W3C standard leaves optimization mostly up to the parser implementation. That means other parsers, not included in the evaluation for this report, may provide better optimization profiles, which would require additional evaluation.

Reduced Tag Names

All tag names in the GJXDM reference schema conform to the ISO 11179 Naming Convention Standards and are subsequently quite lengthy. One scenario of the AMBER Alert test case measured the effect of lengthy tag names.

By reducing the tag names to three character names and making substitutions in the GJXDM reference schema, the instance schema, and the instance document, the transaction time improved by 50% on average.

It appears that overall string length of tags and the size of the document as a whole does have an effect on performance given that subset schemas as well as reducing tag name size improved performance.

Tag Name Length Findings

This special test reused the simplest scenario, AMBER Alert, in order to examine the concern that the long tag names specified by the GJXDM naming conventions adversely affect the time required to create, transmit, consume, and transform XML instances. The 1-, 2-, or 3-letter alpha character codes were used to replace the original element, attribute names, and references defined in the GJXDM reference schema and instance documents. However, reduced tag names were not assigned to other components of the reference schema, such as attributeGroup name, nor were they used in any imported proxy schemas, such as those used to access the NIBRS code types.

The primary findings regarding the Tag Name Length test are:

- Reduced tag names significantly reduced total communication time.
- Reduced tag names did not always affect the total processing time, especially when the total communication time was a smaller percentage of the total transaction time.

Summary Data

Figure 18 depicts the J2EE and .NET standard and reduced tag traffic numbers for both the Aggregate (release) and Full (release). The left side of the figure represents the J2EE data and the right side represents the .NET data. The performance time for J2EE is significantly less in all tests in the Standard and Reduced releases.

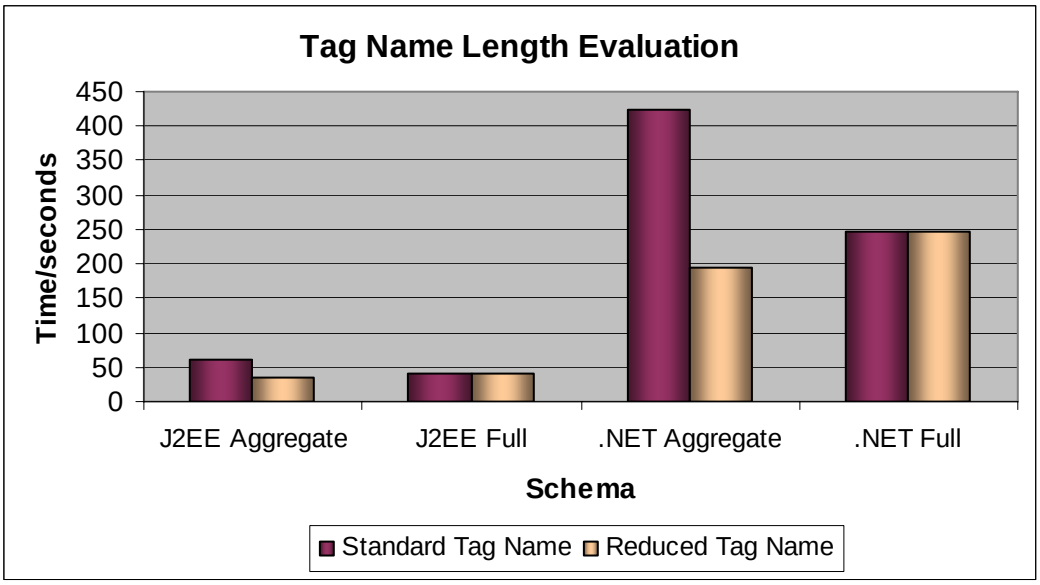

Figure 18 – Performance comparison between standard and reduced tag names

- Figure 19 depicts the percentages of Communication Time versus Total Time in several combinations. The communication times compare a set of J2EE Aggregate Standard and Reduced and J2EE Full Standard and Reduced combinations. The same combinations for .NET are on the right side of the figure.

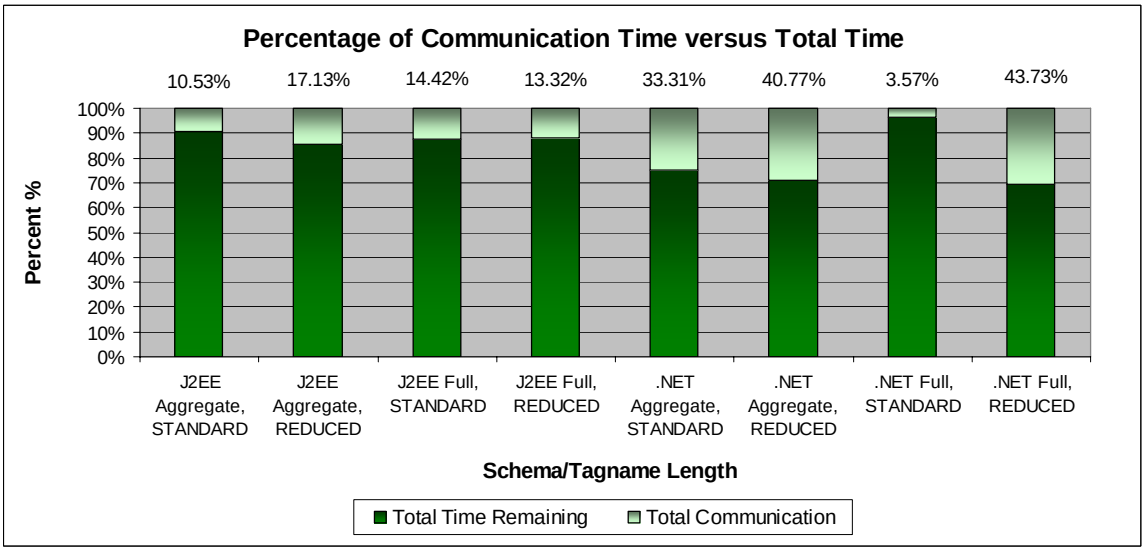

Figure 19 – Percentages of communication time for reduced tag names

- The team found no observable gains in total processing time for Full Schema Reduced Tags in either J2EE or .NET. J2EE processing time was actually slightly higher that the Full Schema Standard Tags (101%), with most of the

extra work occurring in the Request Generation phase on the back-end server. The team observed no changes in .NET processing time, except that front-end server Response Transformation was faster while Total Communication time was slower.

- In contrast, total processing time for Aggregate Schema Reduced tags took 56% less time in J2EE and 46% less time in .NET. While it is not clear why this occurred, one factor may be that the exclusion of the attributeGroup components (used to define SuperType metadata) from the reduced tag name implementation diluted the impact of this approach. Another explanation may be the smaller size of the Aggregate schema.

Network Considerations Findings

Most of the test cases included scenarios for testing the effects of network bandwidth. Extensive testing of the transmission times for instance documents and at least one of the test cases evaluated the effect of referencing a remote schema location.

Summary Data

Calculating the latency time for the transmission steps in milliseconds provided the measurements of network bandwidth in the test harness. Figure 20 shows how network bandwidth affects the latency time for the transmission step.

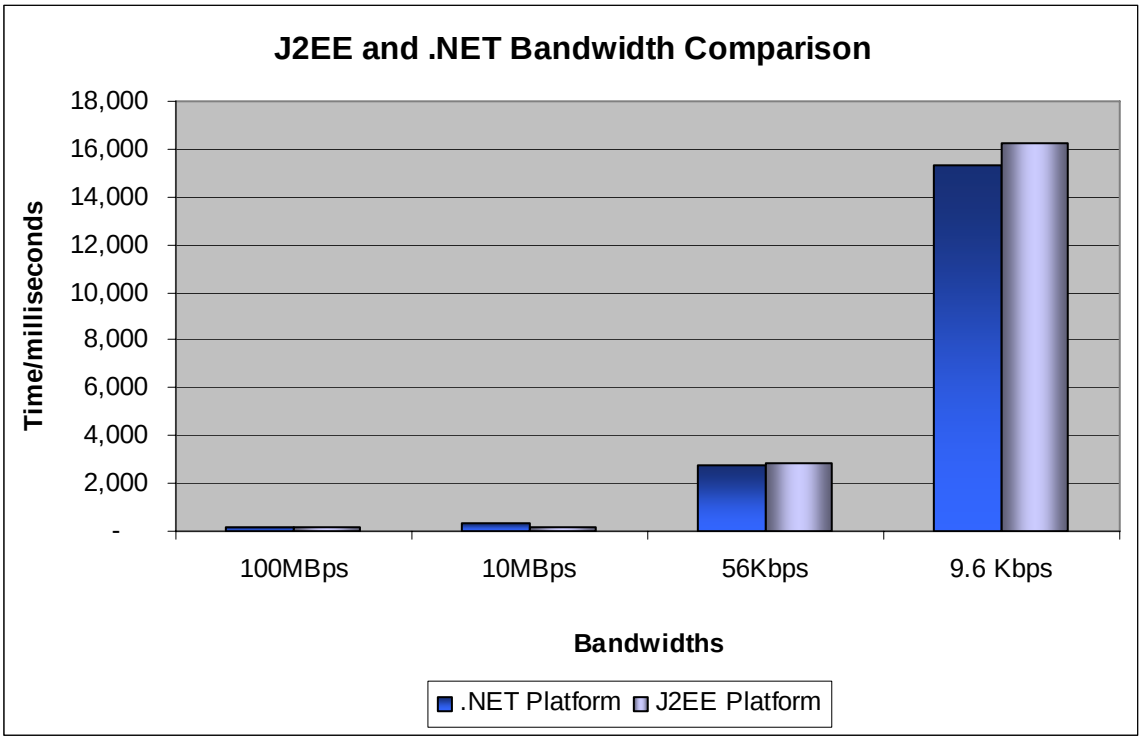

Figure 20 – Network bandwidth performance comparison

Additionally, the Incident Report test case included a scenario for testing the affect of referencing a remote schema. For typical network settings, it added about thirty seconds to the overall transaction time and degraded quickly into hours of latency time in network bandwidth-constrained environments.

Analysis

The test results indicate when the network bandwidth drops to 9,600 bps, that transaction times are significantly impacted.

This has implications mostly for wireless applications of the GJXDM. Many older public safety radio channels and even a few commercial radio networks may not offer much more than 9,600 bps.

The GJXDM should not be used in low bandwidth applications. The intended use of the GJXDM is to expose an application's external touch points and is not intended to manage internal communication within a "closed system" environment such as mobile computing.

Some wireless justice applications may benefit from GJXDM. Factoring the network bandwidth needs into the early designs would reduce the total effort to repair it later.

The test results also indicate that the cost of referencing a remote schema location is significant.

Given XSTF's commitment to maintaining compatibility of the model, there does not appear to be much benefit to referencing a remote version of the GJXDM.

Where Work Happens Findings

Much of this project has focused on the level of effort required to perform common tasks with the GJXDM data model. The result of the work the team performed is a better understanding of not only the level of effort, but also what tasks require more effort.

Summary Data

In all test cases, the largest component of the latency time for the transaction was validation. All of the transactions performed at least one validation step and at least one performed four validations steps.

The other large component of the latency time for the transaction was DOM processing. To test DOM processing the team chose Inmate Record because it contained several transaction schemas with data sets. This appears to be the case for both platforms and with all test scenarios where the GJXDM is involved.

The Figures 21 and 22, that follow, illustrate this point:

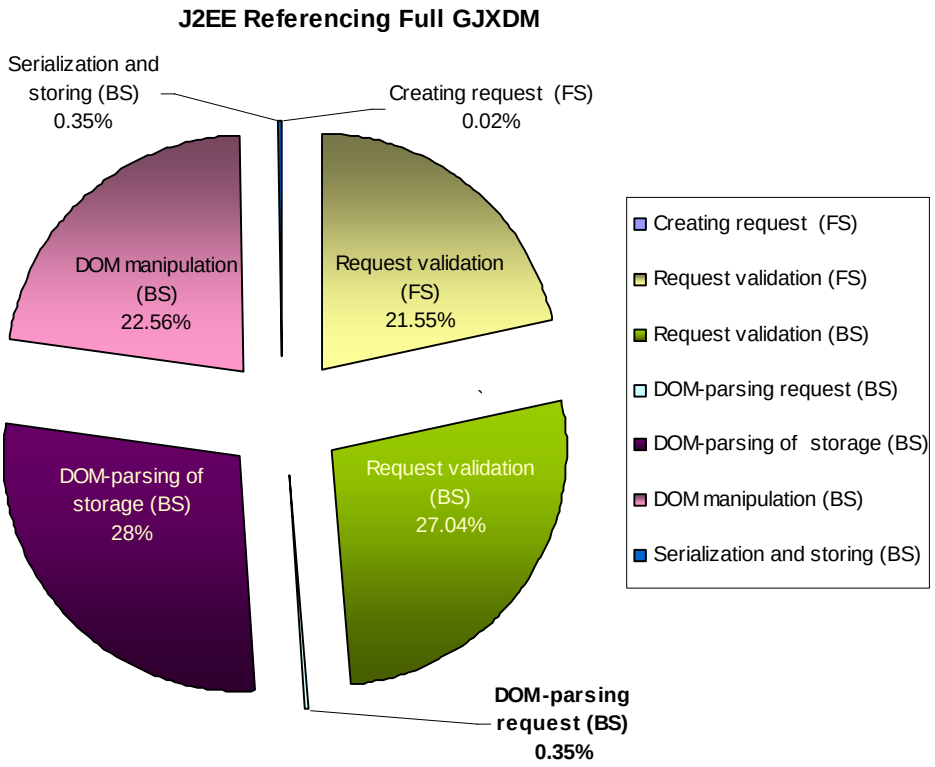

Figure 21 – J2EE where work happens breakdown

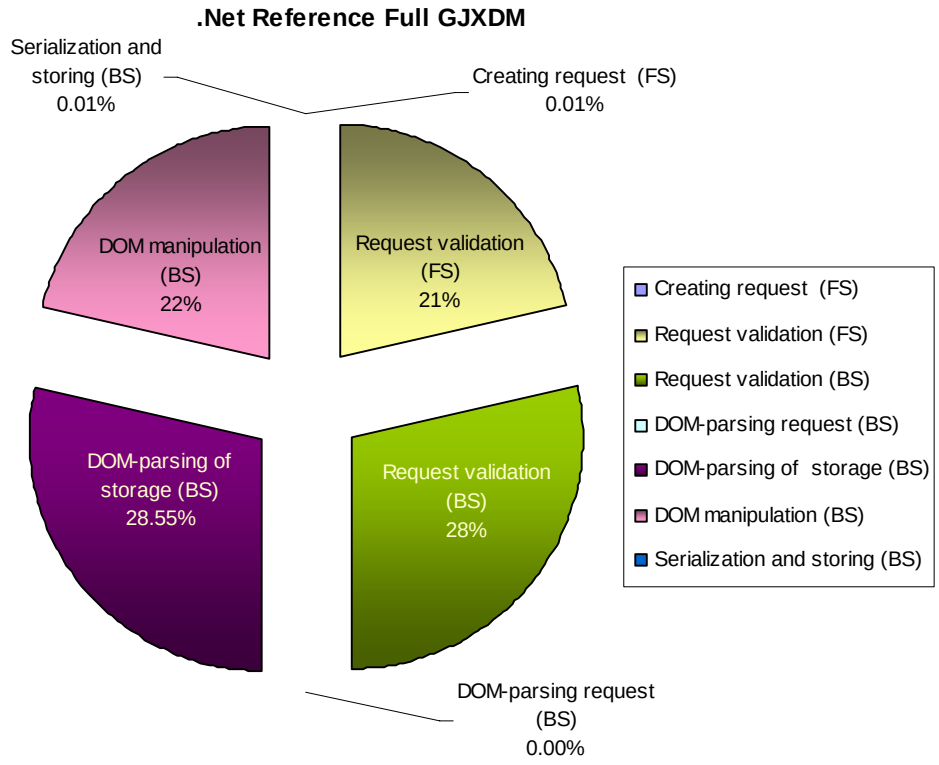

Figure 22 – .NET where work happens breakdown

Analysis

Validation latency is mostly the result of the overall depth and breadth of the data model.

By comparing the validation latency between JXDD 3.0.0.0 pre-release and GJXDM 3.0 initial operating release, the impact of the model complexity becomes clearly visible. JXDD 3.0.0.0 validation time is a small fraction of the validation time for required by the GJXDM 3.0 initial operating release.

Additional research of data model differences and the search for the primary cause for dramatic validation time increases confirmed that complexity increases the validation time.

After evaluation of all of the added elements including proxy schemas, none added significance by itself.

Several of the tests confirm that both XML parsers tested offer some sort of optimization.

Optimization appears based on the parser's ability to load elements, objects and files on demand. The poor performance of the aggregated and flattened schemas proved the basis of this theory. In both cases, it appears that the parser must load the full schema rather than just parts of it.

Based on the test results, it appears that the best schema deployment is what GTRI has recommended as part of the GJXDM data model documentation.

Additionally, validation time can be significant. It is important to define thoroughly the role of conformance validation needs early in the design of a software product.

In practice, it is likely that validation is a "Validation Testing" tool and in normal production, mode validation is disabled.

For the one test case where DOM was tested, the time it took to load the document into DOM was just as significant as schema validation.

Using DOM to perform production work requires careful evaluation, as there are better methods for performing common data processing tasks.

Lessons Learned (Tricks, tips and hints)

Summary
MSXML 3.0, used by Visual Studio .NET by default, cannot validate with schemas and needs to be upgraded to 4.0 version.
To validate with schema under .NET, it is necessary to put it into the schema cache and make an explicit call to the API.
When the WS Developer's Pack 1.3 is installed, it provides its own endorsed JARs for XML processing; this breaks some other Java programs that do XML processing (e.g., command line utils); the solution is to copy the endorsed JARs into jre/lib/endorsed directory in the SDK.
It is possible to enable explicit caches for data and pages in .NET, which would be beneficial for GJXDM processing because the schema information stays the same, and cached in memory as opposed to reading from filesystem repeatedly.
Looking into standard SOAP compression (see http://www-106.ibm.com/developerworks/webservices/library/ws-sqzsoap.html) may help communication times.
When the schemaLocation needs to be set, J2EE on Linux and Windows will look for the entities in different places – J2EE on Windows looks in the home directory of the user running Tomcat, while J2EE on Linux looks in the JWSDP home directory. Therefore, the systemID of the schema will not be the same and the custom entity resolver will wind up being unportable.
On .NET, it is evident that it is parsing GJXDM that is taking all the validation time. The call to AddSchema(), which associates a namespace with a URL where the schema file can be reached, takes almost 100% of the time we count as 'validation'. E.g., in AMBER Alert, a common validation run will consist of 177 seconds for schema parse and 56 milliseconds for the validating parse. On Java, explicitly adding a schema to cache using the API does not result in schema parsing, which is apparently done later, during a call to parse(). Lesson: on .NET, only load schema if it is definitely used, because .NET is significantly slower when loading the schema.
The prefixed elements could not be retrieved because we were not using the correct namespace URI in the .NET's AddNamespace() call. Apparently, the prefix we use does not matter (as long as we are consistent), but it is the URI that has to match the targetNamespace in the XSD file. Therefore, we can add a new prefix to the namespace manager, and use it in the XPath. Although in this particular case we did not face this problem, there appears to be a known 'feature' in .NET XML implementation, which does not handle default name spaces well. If the XPath with mixed prefixed and non-prefixed element addressing does not work, the workaround is to define (through the namespace manager) a dummy namespace matching the URI of the XML file's default namespace and using the prefix when addressing with XPath elements that do not have prefixes. Meaning, add a 'blah' namespace matching 'http://tempuri.org/JXDDv3_InmateAlias.xsd' and pull out ClientID using '/inm:Inmate/blah:ClientID'. For more information, see http://weblogs.asp.net/wallen/archive/2003/04/02/4725.aspx, which sums it up well, and has links to other resources.
Subschema loads show an improvement on J2EE, but not on .NET, which supports the idea that J2EE only brings in the needed parts of the schema. On Java, a marked difference exists in processing time between a file that just imports the GJXDM vs. a file that uses the Person/Vehicle. On .NET, there is no such difference.

Default data encodings used by J2EE and .NET are different. .NET uses "document-style" (i.e., simple html-entities replacement) encoding, while Java uses lengthier RPC encoding, which shows longer transmission times. The settings of JWSDP 1.3 (which were used during our tests) are “document-style”, which is the same as in .NET, so this is not an issue. NOTE: In Java, Web Service compilation/build setting in the Ant scripts (calling wscompile with ‘-f:docliteral’) does not seem to change the on-the-wire representation (needs more investigation later).

Proposed Next Steps

Recommendations for Further Research

- *Explore the effects of data sets of "N" complexity on performance.* The intent is to build a framework set of guidelines for building constraint schemas.
  - Given that GJXDM is "Non-Deterministic", it is possible to build conforming XML instance documents of arbitrary complexity. "Non-Deterministic" describes the JXDM because the data model contains recursive references to elements and does not establish bounds or constraints on these elements. Moreover, the instance document determines the level of recursion.
  - The suggestion is to perform boundary testing with existing XML Parsers to establish reasonable guidelines for complexity level. Limiting the construction of constraint schemas provides those boundaries and serves two purposes. It restrains the complexity limits and establishes the performance expectations based on complexity.

- *Organize a GJXDM Users Group (National or Regional)*
  - One of the biggest risks identified by the team for the adoption of GJXDM was a lack of information about how to apply and implement the data model. Classes and seminars are only the first step.
  - A User's Group establishes a support network of peers to provide real world help and examples to solve problems. It would also be a vehicle to disseminate important information quickly about GJXDM to the user community.

- *Test the effects of splitting the data model into subdocuments*
  - Preliminary research indicates that current implementations of XML parsers are able to better optimize tree construction if the schema is separated into a number of smaller imported files.
  - If the whole model is stored in a single file, the whole file must be loaded. If the model is broken up into several files, only those files containing referenced elements are loaded into the tree.
  - Additional research should focus on better understanding if behavior is this implementation detail of the XML parsers or a reliable best practice.

APPENDIX A – Raw Data

Inmate Record data

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS					START-UP/GC (FS)		REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		DOM-PARSING REQUEST (BS)		DOM-PARSING OF STORAGE (BS)		DOM MANIPULATION (BS)		SERIALIZATION AND STORING (BS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Comm, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
ADD ALIAS																										
J2EE	Aggregate (pre-release)	Minimal	100M	20476	39	1565	18872	3864	681	12317	3	12644	3994	22411	884	10834	4963	20668	4841	29080	5038	39647	26	40693	7	40824
J2EE	Aggregate (pre-release)	Minimal	10M	20527	47	1563	18917	3864	678	12272	3	12600	4010	22367	885	6780	4949	16614	4847	25022	5068	35593	25	36639	15	36770
J2EE	Aggregate (pre-release)	Minimal	56K	20739	255	1563	18921	3864	677	6793	3	7120	4014	16888	886	12531	4961	22365	4853	30773	5057	41344	26	42390	7	42520
J2EE	Aggregate (pre-release)	Minimal	9.6K	21712	1317	1504	18891	3864	618	10788	3	11115	4000	20883	886	6780	4952	16614	4844	25022	5060	35593	26	36639	6	36769
J2EE	Aggregate (pre-release)	Average	100M	20573	45	1579	18949	3864	686	12277	3	12605	4010	22373	893	6780	4950	16614	4855	25022	5087	35658	33	36731	11	36964
J2EE	Aggregate (pre-release)	Average	10M	20548	56	1584	18908	3864	679	12288	3	12615	4003	22383	905	10833	4946	20667	4841	29076	5067	39712	37	40785	11	41018
J2EE	Aggregate (pre-release)	Average	56K	20736	242	1563	18931	3864	678	6794	3	7121	4011	16889	885	12535	4955	22382	4830	30780	5088	41455	33	42528	11	42760
J2EE	Aggregate (pre-release)	Average	9.6K	21836	1336	1564	18936	3864	679	10785	3	11113	3998	20881	885	12538	4971	22372	4842	30780	5078	41417	32	42490	12	42722
J2EE	Aggregate (pre-release)	Large	100M	20639	41	1559	19039	3864	677	6793	3	7120	3999	16908	882	12536	4961	22369	4837	30781	5141	41508	78	42625	20	43067
J2EE	Aggregate (pre-release)	Large	10M	20659	39	1559	19061	3864	677	12282	3	12609	4036	22397	882	10830	4982	20663	4818	29075	5114	39802	89	40919	19	41361
J2EE	Aggregate (pre-release)	Large	56K	20842	273	1566	19003	3864	677	10782	3	11109	4020	20897	889	12534	4960	22367	4840	30779	5115	41506	46	42623	19	43065
J2EE	Aggregate (pre-release)	Large	9.6K	21872	1310	1568	18994	3864	680	10782	3	11109	4000	20897	888	12556	4958	22390	4839	30802	5129	41529	46	42646	19	43088
J2EE	Full (pre-release)	Minimal	100M	22866	48	1677	21141	3870	656	12946	3	13274	4621	24688	1021	14366	5712	25825	5206	39667	5564	48530	28	49562	7	49694
J2EE	Full (pre-release)	Minimal	10M	22850	51	1696	21103	3870	656	12936	3	13264	4633	24679	1040	8910	5737	20375	5182	34221	5514	43059	27	44091	7	44223
J2EE	Full (pre-release)	Minimal	56K	23175	267	1678	21230	3870	656	12925	3	13253	4688	24668	1022	14335	5762	25794	5197	39631	5545	48476	28	49508	7	49640
J2EE	Full (pre-release)	Minimal	9.6K	24097	1317	1683	21097	3870	656	12926	3	13254	4639	24668	1027	14356	5729	25803	5191	39637	5500	48487	28	49519	7	49651
J2EE	Full (pre-release)	Average	100M	23050	45	1710	21295	3870	671	12956	3	13284	4640	24705	1039	14370	5749	25834	5179	39664	5672	48567	39	49644	13	49878
J2EE	Full (pre-release)	Average	10M	22946	44	1718	21184	3870	678	7150	3	7478	4679	18893	1040	14787	5740	26251	5184	40080	5526	48983	39	50060	13	50294
J2EE	Full (pre-release)	Average	56K	23124	260	1711	21153	3870	671	12935	3	13263	4641	24678	1040	8910	5741	20368	5183	34206	5532	43098	39	44175	14	44409
J2EE	Full (pre-release)	Average	9.6K	24185	1317	1714	21154	3870	672	12936	4	13264	4642	24678	1042	14342	5725	25801	5187	39644	5544	48536	39	49613	13	49847
J2EE	Full (pre-release)	Large	100M	25412	173	1459	23780	3870	591	11714	3	12042	5047	25353	868	12583	6462	25940	5962	35755	6162	46294	92	47381	52	47827
J2EE	Full (pre-release)	Large	10M	24176	48	1563	22565	3870	667	10822	3	11150	4926	24424	896	6780	6072	20137	5660	29969	5805	40488	74	41576	25	42021
J2EE	Full (pre-release)	Large	56K	24553	274	1554	22725	3870	674	12441	3	12769	5123	26061	880	12725	6070	26052	5636	35896	5815	46435	59	47523	19	47968
J2EE	Full (pre-release)	Large	9.6K	25348	1317	1567	22464	3870	677	10826	3	11154	4896	24446	890	6780	6060	20137	5641	29951	5790	40490	55	41578	19	42023
UPDATE ADDRESS																										
J2EE	Aggregate (pre-release)	Minimal	100M	19989	107	1731	18151	5578	690	12912	21	13675	3904	21804	1041	14813	4865	28820	4555	35892	4778	45098	19	45806	9	45941
J2EE	Aggregate (pre-release)	Minimal	9.6K	20694	1345	1742	17607	5578	677	13073	7	13836	3767	21981	1065	14801	4524	28809	4541	35879	4731	45067	27	45774	10	45910
J2EE	Aggregate (pre-release)	Average	100M	20490	139	1801	18550	5578	679	12929	7	13691	3781	21820	1122	14845	4617	28837	4553	35922	5555	45156	15	45863	22	46124
J2EE	Aggregate (pre-release)	Average	9.6K	21956	2605	1685	17666	5578	660	7157	6	7919	3749	16064	1025	8916	4494	22922	4517	29994	4863	39248	15	39955	22	40216
J2EE	Aggregate (pre-release)	Large	100M	21626	114	1627	19885	5578	730	6169	7	6931	4138	16584	897	12592	5495	22425	4876	30854	5264	41617	26	42319	79	42770
J2EE	Aggregate (pre-release)	Large	9.6K	23192	2586	1620	18986	5578	709	12744	6	13506	4009	23159	911	12653	4961	22468	4850	30897	5121	41660	15	42362	24	42813
J2EE	Full (pre-release)	Minimal	100M	23502	54	1691	21757	5585	662	7157	55	7920	4888	19205	1029	8915	5939	20374	5226	34194	5625	43039	15	43746	9	43883
J2EE	Full (pre-release)	Minimal	9.6K	25343	2586	1685	21072	5585	661	12293	6	13056	4671	24341	1024	14345	5729	25802	5221	39624	5420	48473	15	49180	10	49316
J2EE	Full (pre-release)	Average	100M	23005	55	1721	21229	5585	677	13041	7	13804	4637	25089	1044	14793	5747	26252	5186	40077	5616	48972	15	49679	21	49942
J2EE	Full (pre-release)	Average	9.6K	25501	2603	1741	21157	5585	697	12308	6	13072	4657	24401	1044	14348	5750	25807	5202	39644	5505	48519	15	49226	22	49489
J2EE	Full (pre-release)	Large	100M	24573	93	1557	22923	5585	594	10093	7	10857	5079	24063	963	6784	6240	20073	5707	29924	5828	40462	16	41164	46	41617
J2EE	Full (pre-release)	Large	9.6K	26628	2584	1559	22485	5585	677	12760	17	13523	4905	26676	882	12638	6085	25928	5659	35782	5781	46320	14	47022	24	47476

Inmate Record data (cont'd)

BASIC (PARTIAL) QUERY				GENERAL TRANSACTION STATISTICS					START-UP/GC (FS)		REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		RESPONSE GEN (BS)		VALIDATION (BS)		VALIDATION (FS)		TRANSFORMATION (FS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Comm, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
J2EE	Aggregate (pre-release)	Average	100M	19258	180	1746	17332	11956	798	13757	11	15102	3843	23075	948	14835	4545	28772	143	24300	4580	38060	3888	31787	322	33739
J2EE	Aggregate (pre-release)	Average	10M	19006	184	1794	17028	11956	813	13577	11	14923	3760	22896	981	14834	4549	28771	147	24299	4495	38059	3745	31608	321	33568
J2EE	Aggregate (pre-release)	Average	56K	19690	933	1796	16961	11956	814	13667	11	15012	3764	22986	982	14874	4537	28812	149	24339	4499	38099	3750	31697	251	33650
J2EE	Aggregate (pre-release)	Average	9.6K	23694	5048	1801	16845	11956	815	13572	11	14917	3760	22891	986	14839	4501	28759	150	24287	4494	38047	3738	31607	191	33567
J2EE	Full (pre-release)	Average	100M	23716	84	1860	21772	11978	837	7810	21	9159	4849	20393	1023	8924	5852	20384	54	21732	5822	32935	4853	32377	321	34414
J2EE	Full (pre-release)	Average	10M	23076	237	1730	21109	11978	783	13622	11	14972	4721	26172	947	14836	5804	26255	12	27603	5685	38834	4623	38155	253	40108
J2EE	Full (pre-release)	Average	56K	23805	935	1849	21021	11978	901	7833	11	9183	4667	20395	948	8924	5747	20384	12	21732	5720	32935	4646	32380	218	34332
J2EE	Full (pre-release)	Average	9.6K	27662	5075	1741	20846	11978	788	13638	11	14988	4645	26200	953	14841	5723	26284	13	27632	5700	38848	4596	38184	158	40137
FULL QUERY																										
J2EE	Aggregate (pre-release)	Minimal	100M	20190	305	1539	18346	14669	739	10556	11	11901	3848	19895	800	11558	4744	25465	161	22673	5060	32777	4015	31255	507	33791
J2EE	Aggregate (pre-release)	Minimal	10M	19569	81	1566	17922	14669	762	12587	11	13932	3799	21906	804	12814	4550	26723	166	23933	5023	34020	4132	33284	241	35786
J2EE	Aggregate (pre-release)	Minimal	56K	20290	1347	1576	17367	14669	771	12678	11	14023	3682	22015	805	12783	4534	26691	161	23898	4905	34003	3913	33377	161	35829
J2EE	Aggregate (pre-release)	Minimal	9.6K	26302	7430	1586	17286	14669	775	12642	11	13987	3683	21961	811	12825	4510	26734	162	23944	4914	34032	3856	33338	150	35791
J2EE	Aggregate (pre-release)	Average	100M	20887	264	1962	18661	24777	956	6901	11	8247	3765	16259	1006	6899	4606	20807	379	22632	5245	25588	4181	28067	474	30633
J2EE	Aggregate (pre-release)	Average	10M	19645	122	1544	17979	24777	745	12636	12	13981	3703	21973	799	12807	4767	26715	321	28540	5002	31496	3904	33789	270	36347
J2EE	Aggregate (pre-release)	Average	56K	22010	2876	1621	17513	24777	815	6827	11	8173	3662	16146	806	6829	4493	20736	326	22562	4920	25519	3866	27983	235	30541
J2EE	Aggregate (pre-release)	Average	9.6K	35365	16270	1564	17531	24777	754	12686	11	14032	3671	22005	810	12819	4510	26727	329	28552	4919	31512	3911	33842	180	36400
J2EE	Aggregate (pre-release)	Large	100M	24433	541	1541	22351	43290	615	11911	11	13256	4187	22757	926	12664	5332	22403	706	29460	5334	33876	4598	30991	2183	36290
J2EE	Aggregate (pre-release)	Large	10M	21886	215	1635	20036	43290	731	6793	11	8138	4174	17633	904	12648	4998	22391	709	29463	5262	33868	4446	25872	436	28217
J2EE	Aggregate (pre-release)	Large	56K	27350	6005	1599	19746	43290	694	12572	11	13917	4070	23419	905	12648	4978	22390	702	29297	5276	33725	4329	31665	380	34002
J2EE	Aggregate (pre-release)	Large	9.6K	54311	32960	1612	19739	43290	706	6817	11	8162	4052	17643	906	12650	4963	22405	702	29462	5302	33867	4366	25896	343	28232
J2EE	Aggregate (release)	Average	100M	128945	1384	7647	119914		1163	8875	20	10532	25679	42603	6484	12926	32970	45539	304	42892	33869	75339	26871	67257	201	69493
J2EE	Full (pre-release)	Minimal	100M	24059	150	1515	22394	14691	739	11975	51	13324	4890	24527	776	12508	6057	23966	41	27009	6142	38058	4871	36749	342	39203
J2EE	Full (pre-release)	Minimal	10M	22488	89	1577	20822	14691	775	6828	12	8177	4535	19411	802	6829	5753	18287	27	21330	5681	32379	4595	31632	219	34084
J2EE	Full (pre-release)	Minimal	56K	23592	1347	1564	20681	14691	754	11792	11	13142	4537	24342	810	12325	5770	23767	27	26810	5688	37877	4502	36543	146	38995
J2EE	Full (pre-release)	Minimal	9.6K	29662	7419	1553	20690	14691	759	10463	11	11813	4538	23013	794	10877	5740	22318	28	25361	5694	36428	4534	35240	145	37693
J2EE	Full (pre-release)	Average	100M	23747	138	1587	22022	24799	755	12020	12	13370	4605	24570	832	12520	5903	23937	240	31673	6004	35030	4900	37338	358	39914
J2EE	Full (pre-release)	Average	10M	22760	126	1589	21045	24799	766	12588	11	13938	4549	25138	823	12761	5777	24219	215	31982	5713	35339	4530	37906	250	40464
J2EE	Full (pre-release)	Average	56K	25394	2880	1608	20906	24799	784	6836	11	8186	4527	19419	824	6829	5747	18287	199	26050	5712	29368	4526	32204	184	34763
J2EE	Full (pre-release)	Average	9.6K	38773	16251	1599	20923	24799	773	12158	11	13507	4552	24710	826	12758	5760	24192	200	31972	5694	35290	4527	37513	179	40071
J2EE	Full (pre-release)	Large	100M	25789	230	1569	23990	43312	737	12608	12	13958	5145	27023	832	12695	6274	25986	688	25249	6051	38442	5129	36103	691	38476
J2EE	Full (pre-release)	Large	10M	25250	208	1556	23486	43312	723	10648	11	11997	4967	25042	833	6829	6075	20121	694	19384	6066	32577	5216	34126	457	36475
J2EE	Full (pre-release)	Large	56K	30258	5594	1560	23104	43312	727	12649	12	13999	4936	27064	833	12695	6064	25968	682	25249	6064	38445	4999	36140	347	38481
J2EE	Full (pre-release)	Large	9.6K	56401	31719	1566	23116	43312	731	10643	11	11993	4921	25059	835	6830	6066	20121	683	19384	6046	32577	5033	34187	356	36553
J2EE	Full (release)	Average	100M	78004	172	2027	75805		1011	14666	13	16286	16335	39475	1016	13592	20498	37378	268	34875	21327	58245	17102	59125	262	55759
J2EE/Sarvega	Full (release)	Average	100M	1910	137	1348	**425**		741	8920	13	10540	0	10540	607	6599	0	6600	192	10107	1	10108	1	12090	218	14294
J2EE	SubSchema (release)	Average	100M	99804	238	1845	97721		915	14715	12	16335	16157	39603	930	13659	20677	37431	270	34920	33722	67070	26658	67744	225	64280

Inmate Record data (cont'd)

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS					START-UP/GC (FS)		REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		DOM-PARSING REQUEST (BS)		DOM-PARSING OF STORAGE (BS)		DOM MANIPULATION (BS)		SERIALIZATION AND STORING (BS)	
Platform	Schema	Data size	Network	Total time, ms	Total Comm, ms	Total overhead, ms	Total processing, ms	TCP conversation	Time, ms	RAM, MB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
ADD ALIAS																										
.NET	Aggregate (pre-release)	Minimal	100M	7463	132	3	7328	5235	1	18080	20	18080	3446	8825	2	15463	3809	13166	2	13166	40	13166	7	13166	4	13166
.NET	Aggregate (pre-release)	Minimal	10M	6684	121	3	6560	5235	1	25685	1	25685	3178	31086	2	24673	3361	27940	2	27940	8	27940	6	27940	4	27940
.NET	Aggregate (pre-release)	Minimal	56K	7385	295	1	7089	5235	0	25501	1	28380	3203	21028	1	14792	3865	20595	2	20595	8	20595	6	20595	4	20595
.NET	Aggregate (pre-release)	Minimal	9.6K	6546	114	1	6431	5235	0	19241	1	19241	2909	19192	1	12481	3501	12500	2	12500	8	12500	6	12500	4	12500
.NET	Aggregate (pre-release)	Average	100M	7900	121	1	7778	5237	0	8123	1	8123	3390	8555	1	13242	4316	13355	2	13355	50	13355	10	13355	9	13355
.NET	Aggregate (pre-release)	Average	10M	6523	116	1	6406	5237	0	31086	1	31086	2995	19249	1	1333	3376	12716	2	12716	15	12716	10	12716	7	12716
.NET	Aggregate (pre-release)	Average	56K	6858	115	2	6741	5237	1	21028	1	21028	3005	20036	1	20595	3701	25585	2	25585	15	25585	10	25585	7	25585
.NET	Aggregate (pre-release)	Average	9.6K	7030	116	1	6913	5237	0	19192	1	19192	3186	20775	1	12500	3692	17399	2	17399	15	17399	10	17399	7	17399
.NET	Aggregate (pre-release)	Large	100M	7030	118	1	6911	5236	0	9300	1	9300	3383	9748	1	13503	3468	13511	2	13511	27	13511	17	13511	13	13511
.NET	Aggregate (pre-release)	Large	10M	6692	116	1	6575	5236	0	19249	1	19249	2934	19184	1	12716	3580	12784	2	12784	28	12784	17	12784	13	12784
.NET	Aggregate (pre-release)	Large	56K	6804	117	1	6686	5236	0	20036	1	20036	2986	19835	1	25585	3640	12632	2	12632	27	12632	17	12632	13	12632
.NET	Aggregate (pre-release)	Large	9.6K	6668	117	1	6550	5236	0	20775	1	20775	3003	20042	1	17399	3485	22429	2	22429	28	22429	18	22429	13	22429
.NET	Full (pre-release)	Minimal	100M	11168	142	3	11023	5241	1	9439	1	9439	5297	11861	2	13528	5704	15570	2	15570	8	15570	6	15570	5	15570
.NET	Full (pre-release)	Minimal	10M	10657	115	3	10539	5241	1	19184	1	19184	4748	21726	2	12784	5770	19890	2	19890	8	19890	6	19890	4	19890
.NET	Full (pre-release)	Minimal	56K	10704	117	1	10586	5241	0	19835	1	19835	4811	21739	1	12632	5754	19698	2	19698	8	19698	6	19698	4	19698
.NET	Full (pre-release)	Minimal	9.6K	10629	115	3	10511	5241	1	20042	1	20042	4833	21998	2	22429	5657	14375	2	14375	8	14375	6	14375	4	14375
.NET	Full (pre-release)	Average	100M	11475	119	1	11355	5243	0	11425	1	11425	5402	11921	1	15022	5888	22822	2	22822	44	22822	10	22822	8	22822
.NET	Full (pre-release)	Average	10M	10674	115	1	10558	5243	0	21726	1	21726	4818	21016	1	19890	5705	14833	2	14833	15	14833	10	14833	7	14833
.NET	Full (pre-release)	Average	56K	10660	115	2	10543	5243	1	21739	1	21739	4814	21190	1	19698	5694	14660	2	14660	15	14660	10	14660	7	14660
.NET	Full (pre-release)	Average	9.6K	10618	116	1	10501	5243	0	21998	1	21998	4844	21453	1	14375	5622	14440	2	14440	15	14440	10	14440	7	14440
.NET	Full (pre-release)	Large	100M	10762	118	2	10642	5242	0	11690	1	11690	4672	23742	2	30491	5883	16953	2	16953	54	16953	17	16953	13	16953
.NET	Full (pre-release)	Large	10M	11468	117	1	11350	5242	0	21016	1	21016	5144	25931	1	14833	6145	22498	2	22498	27	22498	18	22498	13	22498
.NET	Full (pre-release)	Large	56K	11253	115	1	11137	5242	0	21190	1	21190	5160	26247	1	14660	5915	22352	2	22352	28	22352	18	22352	13	22352
.NET	Full (pre-release)	Large	9.6K	11583	745	1	10837	5242	0	21453	1	21453	4984	23611	1	16990	5793	17032	2	17032	28	17032	17	17032	12	17032
UPDATE ADDRESS																										
.NET	Aggregate (pre-release)	Minimal	100M	7065	124	3	6938	6972	1	26070	24	26070	3152	19909	2	24014	3724	13459	3	13459	8	13459	23	13459	4	13459
.NET	Aggregate (pre-release)	Minimal	9.6K	8050	1393	3	6654	6972	1	19661	1	19661	2943	30004	2	17032	3688	21964	3	21964	8	21964	7	21964	4	21964
.NET	Aggregate (pre-release)	Average	100M	6619	121	1	6497	6973	0	19568	1	19568	2982	19389	1	13568	3482	13792	3	13792	15	13792	7	13792	7	13792
.NET	Aggregate (pre-release)	Average	9.6K	7976	1396	1	6579	6973	0	30004	1	30004	2887	19514	1	21964	3345	25195	3	25195	330	21921	7	21921	6	21921
.NET	Aggregate (pre-release)	Large	100M	6685	124	1	6560	6972	0	19123	1	19123	2936	19042	1	14071	3574	14316	3	14316	27	14316	7	14316	12	14316
.NET	Aggregate (pre-release)	Large	9.6K	7641	1392	1	6248	6972	0	19514	1	19514	2850	19527	1	21921	3347	25174	3	25174	28	25174	7	25174	12	25174
.NET	Full (pre-release)	Minimal	100M	11405	123	3	11279	6977	1	19042	12	19042	5182	23420	2	14316	6063	16634	3	16634	8	16634	7	16634	4	16634
.NET	Full (pre-release)	Minimal	9.6K	11790	1575	3	10212	6977	1	19527	1	19527	4813	21122	2	13923	5376	28816	3	28816	8	28816	7	28816	4	28816
.NET	Full (pre-release)	Average	100M	10842	121	1	10720	6978	0	23420	1	23420	4871	22220	1	16634	5817	16196	3	16196	15	16196	7	16196	6	16196
.NET	Full (pre-release)	Average	9.6K	12515	1422	1	11092	6978	0	21122	1	21122	5142	25945	1	28816	5694	14351	3	14351	239	17259	7	17259	6	17259
.NET	Full (pre-release)	Large	100M	11085	125	1	10959	6977	0	22220	1	22220	4841	22017	1	16196	6068	23461	3	23461	27	23461	7	23461	12	23461
.NET	Full (pre-release)	Large	9.6K	12341	1396	1	10944	6977	0	25945	1	25945	5029	22285	1	17259	5865	25016	3	25016	27	25016	7	25016	12	25016

Inmate Record data (cont'd)

BASIC (PARTIAL) QUERY				General transaction statistics					Startup/GC (FS)		REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		RESPONSE GEN (BS)		VALIDATION (BS)		VALIDATION (FS)		TRANSFORMATION (FS)	
Platform	Schema	Data Size	Network	Total time, ms	Total Comm, ms	Total overhead, ms	Total processing, ms	TCP conversation	Time, ms	RAM, MB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, MB	Time, ms	RAM, KB	Time, ms	RAM, KB
.NET	Aggregate (pre-release)	Average	100M	13952	170	3	13779	13588	1	22017	27	22017	2920	19771	2	23461	3658	28921	28	28921	3665	13908	3212	21620	269	21620
.NET	Aggregate (pre-release)	Average	10M	13827	124	3	13700	13588	1	25931	1	25931	3179	20121	2	22498	3632	12499	2	12499	3723	17444	3137	24734	26	24734
.NET	Aggregate (pre-release)	Average	56K	11253	115	1	11137	13588	0	21190	1	21190	5160	26247	1	14660	5915	22352	2	22352	28	22352	18	22352	13	22352
.NET	Aggregate (pre-release)	Average	9.6K	16856	3935	3	12918	13588	1	22285	1	22285	2860	20061	2	25016	3547	14302	2	14302	3303	12482	3179	21781	26	21781
.NET	Full (pre-release)	Average	100M	22368	125	3	22240	13580	1	31202	18	31202	5192	23319	2	12485	6011	19378	19	19378	5729	14453	5230	26758	41	26758
.NET	Full (pre-release)	Average	10M	22086	122	2	21962	13580	1	24734	2	24734	4851	26955	1	17444	5628	14827	2	14827	5935	22621	5518	28923	26	28923
.NET	Full (pre-release)	Average	56K	22878	791	3	22084	13580	1	24807	1	24807	4848	27132	2	19007	5816	26650	2	26650	5864	16758	5527	28830	26	28830
.NET	Full (pre-release)	Average	9.6K	28781	7571	3	21207	13580	1	21781	1	21781	4852	26474	2	12482	5542	14493	2	14493	5550	15200	5234	24858	26	24858
FULL QUERY																										
.NET	Aggregate (pre-release)	Minimal	100M	14621	134	1	14486	16371	0	32975	1	32975	3087	20581	1	24074	3748	13113	2	13113	4134	18306	3477	24964	37	24964
.NET	Aggregate (pre-release)	Minimal	10M	14633	368	1	14264	16371	0	28923	1	28923	3092	30323	1	22621	3647	12617	2	12617	4115	17709	3370	36078	37	36078
.NET	Aggregate (pre-release)	Minimal	56K	15315	1461	3	13851	16371	1	28830	1	28830	3114	30607	2	16758	3324	20143	2	20143	4009	14742	3364	35697	37	35697
.NET	Aggregate (pre-release)	Minimal	9.6K	27784	13456	1	14327	16371	0	24858	1	24858	3053	30711	1	15200	3639	12961	2	12961	4111	18096	3484	25116	37	25116
.NET	Aggregate (pre-release)	Average	100M	14777	148	1	14628	26894	0	24964	1	24964	3045	26074	1	18306	3764	23516	3	23516	4196	13264	3556	32624	63	32624
.NET	Aggregate (pre-release)	Average	10M	14580	430	1	14149	26894	0	36078	1	36078	2888	19343	1	17709	3593	12524	3	12524	4099	17726	3513	25147	52	25147
.NET	Aggregate (pre-release)	Average	56K	17061	2956	1	14104	26894	0	35697	2	35697	3132	20614	1	14742	3371	12448	3	12448	4050	14172	3494	24973	52	24973
.NET	Aggregate (pre-release)	Average	9.6K	39719	25361	1	14357	26894	0	25116	1	25116	3052	26143	1	18096	3616	12803	3	12803	4116	17916	3517	33051	52	33051
.NET	Aggregate (pre-release)	Large	100M	14802	179	1	14622	46135	0	32624	1	32624	2923	20143	1	13264	3765	18482	4	18482	4263	23650	3589	21930	77	21930
.NET	Aggregate (pre-release)	Large	10M	14662	196	1	14465	46135	0	25147	1	25147	3015	26270	1	17726	3558	12483	4	12483	4305	19808	3506	32931	76	32931
.NET	Aggregate (pre-release)	Large	56K	20430	5769	1	14660	46135	0	24973	1	24973	3043	26240	1	14172	3726	19466	4	19466	3925	14010	3885	25629	76	25629
.NET	Aggregate (pre-release)	Large	9.6K	55899	41661	1	14237	46135	0	33051	1	33051	2941	20256	1	17916	3576	12613	4	12613	4123	17746	3516	22001	76	22001
.NET	Aggregate (release)	Average	100M	832395	1282	1	831112		0	40463	2	40463	176126	40216	1	36559	234923	36483	16	36483	235772	36947	184208	67319	65	67319
.NET	Full (pre-release)	Minimal	100M	22180	132	1	22047	16391	0	26153	1	26153	5195	21514	1	22274	5628	14388	2	14388	5992	14183	5192	26738	37	26738
.NET	Full (pre-release)	Minimal	10M	21733	130	1	21602	16391	0	32931	1	32931	4869	22247	1	19808	5813	16634	2	16634	5742	16554	5137	25810	38	25810
.NET	Full (pre-release)	Minimal	56K	22800	1227	1	21572	16391	0	25629	1	25629	4855	27510	1	14010	5670	15672	2	15672	5595	15118	5412	26264	37	26264
.NET	Full (pre-release)	Minimal	9.6K	28296	6458	1	21837	16391	0	22001	1	22001	5106	29380	1	17746	5595	14529	2	14529	5899	22223	5197	27017	37	27017
.NET	Full (pre-release)	Average	100M	22454	148	1	22305	26914	0	26719	1	26719	4997	32475	1	22551	5754	14621	2	14621	6086	22408	5408	25220	57	25220
.NET	Full (pre-release)	Average	10M	22247	338	1	21908	26914	0	25810	2	25810	5198	20950	1	16554	5988	23977	3	23977	5692	15378	4973	25252	52	25252
.NET	Full (pre-release)	Average	56K	24939	2744	1	22194	26914	0	26264	1	26264	5134	34973	1	15118	5866	22934	3	22934	5936	30772	5202	26317	52	26317
.NET	Full (pre-release)	Average	9.6K	37320	15285	1	22034	26914	0	27017	1	27017	4986	32791	1	22223	5683	14443	3	14443	6008	14222	5301	25325	52	25325
.NET	Full (pre-release)	Large	100M	22387	178	3	22206	46157	1	27939	1	27939	5037	22778	2	22408	5702	14527	4	14527	6110	22274	5274	26153	78	26153
.NET	Full (pre-release)	Large	10M	22329	335	1	21993	46157	0	25252	1	25252	4990	22017	1	15378	5994	22959	4	22959	5679	14792	5247	25501	78	25501
.NET	Full (pre-release)	Large	56K	28006	5801	1	22204	46157	0	26317	1	26317	5334	23702	1	30772	5885	17192	4	17192	5914	24984	4988	25220	78	25220
.NET	Full (pre-release)	Large	9.6K	53930	31586	3	22341	46157	1	26259	1	26259	5204	21562	2	24680	5782	16460	4	16460	6056	24011	5217	26612	77	26612
.NET	Full (release)	Average	100M	470922	149	3	470770		1	13867	38	13867	100676	28594	2	38984	134333	26001	12	26001	134425	25862	101219	46631	67	46631
.NET	SubSchema (release)	Average	100M	654762	5967	357	648438		343	6961	26	6961	100984	27026	14	7324	135924	26067	20	26067	234934	38770	176461	57036	89	57036

AMBER Alert (+tags) data

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS								Setup/GC (FS)		Startup/GS (BS)		Response generation (BS)		Response validation (BS)		Response validation (FS)		Response transformation (FS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Communication, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	BS-FS traffic	RAM, KB	TCP conversation	RAM, KB	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
J2EE	Full (release)	Average	100M	40208	58	1714	38436	1611	8636	10247	8647	818	13544	896	7155	21	9584	21627	27824	16587	38604	201	34416
J2EE	Full (release)	Average	10M	40068	66	1691	38311	1611	8636	10247	8647	820	12185	871	13181	22	15610	21551	33858	16557	37224	181	33026
J2EE	Full (release)	Average	56K	40663	783	1741	38139	1809	8702	10511	8647	848	8056	893	13061	23	15490	21356	33748	16563	33095	197	28897
J2EE	Full (release)	Average	9.6K	44118	4246	1699	38173	1677	8636	10313	8647	822	13611	877	12908	21	15337	21402	33577	16567	38650	183	34452
J2EE/Sarvega	Full (release)	Average	100M	1481	63	1253	165	1611	8636	10247	8647	640	8433	613	6456	22	8885	1	8886	0	9074	142	10404
J2EE/Sarvega	Full (release)	Average	9.6K	6537	5120	1250	167	1677	8636	10313	8647	649	8406	601	6507	23	8936	0	8937	1	9047	143	10361
J2EE	Sub-schema (release)	Average	100M	62661	90	1577	60994	1480	6634	8114	8647	802	7160	775	12509	48	14940	34073	42165	26688	40927	185	36889
J2EE	Sub-schema (optimized)	Average	100M	32805	49	1685	31071	1480	6634	8114	8647	814	15092	871	13107	22	15536	17305	35475	13635	36471	109	37785
J2EE	Sub-schema (optimized)	Average	9.6K	38206	5539	1762	30905	1756	8148	9904	8647	889	9052	873	7582	22	10011	17275	29967	13522	30435	86	31749
REDUCED TAG NAMES																							
J2EE	Aggregate (release)	Average	10M	33326	79	1600	31647	1486	4947	6433	5164	785	8054	815	13183	10	14326	17794	34139	13729	28613	114	29909
J2EE	Aggregate (release)	Average	56K	33609	247	1591	31771	1484	4950	6434	5164	775	12288	816	13063	10	14206	17916	34020	13713	32885	132	34181
J2EE	Aggregate (release)	Average	9.6K	35650	2504	1590	31556	1484	4950	6434	5164	773	13638	817	7137	10	8280	17705	28076	13720	34255	121	35551
J2EE	Full (release)	Average	100M	40529	54	1638	38837	1486	4947	6433	5163	811	8162	827	13281	45	14421	22185	35319	16443	30124	164	31419
J2EE	Full (release)	Average	10M	39521	53	1614	37854	1486	4947	6433	5163	784	13903	830	13294	10	14434	21250	35330	16470	35866	124	37161
J2EE	Full (release)	Average	56K	39702	246	1615	37841	1486	4947	6433	5163	786	14173	829	7244	9	8385	21196	29256	16529	36134	107	37429
J2EE	Full (release)	Average	9.6K	40568	1150	1617	37801	1486	4947	6433	5163	788	13773	829	12920	10	14060	21221	34955	16466	35735	104	37031
STANDARD TAG NAMES																							
.NET	Aggregate (release)	Average	100M	424138	1413	367	422358	982	8533	9515	8909	367	8728	0	39590	2	39590	240438	37635	181688	38614	230	40464
.NET	Aggregate (release)	Average	9.6K	428347	5627	2	422718	1522	16183	17705	8909	1	40190	1	36839	2	36839	241693	36837	180818	39093	205	40710
.NET	Aggregate Flat (release)	Average	100M	1160910	35	2	1160873	926	8020	8946	8400	1	40464	1	37635	18	37635	658976	84312	501836	87155	43	87155
.NET	Aggregate Flat (release)	Average	9.6K	1171087	10162	2	1160923	1526	14644	16170	8400	1	40710	1	36837	13	36837	660906	85163	499980	87821	24	86677
.NET	Full (release)	Average	100M	246636	88	176	246372	984	8630	9614	8908	175	23734	1	15305	2	15305	140220	27202	106106	28765	44	28765
BASIC (PARTIAL) QUERY																							
.NET	Sub-Schema (release)	Average	100M	421676	90	2	421584	985	8630	9615	8908	1	40543	1	39165	2	39165	240806	37569	180751	38462	25	38462
.NET	Sub-schema (optimized)	Average	100M	190028	123	0	189905	985	8630	9615	8908	0	25298	0	23289	2	23289	108060	23150	81693	25027	150	26612
.NET	Sub-schema (optimized)	Average	9.6K	193686	3945	0	189741	1525	18408	19933	8908	0	24384	0	24346	2	24346	107748	23289	81965	25298	26	25298
.NET/Sarvega	Full (release)	Average	100M	120	93	0	27	985	8630	9615	8908	0	7776	0	6944	2	6944	0	6944	0	7776	25	7776
.NET/Sarvega	Full (release)	Average	9.6K	12301	12274	0	27	1525	18408	19933	8908	0	9356	0	6944	2	6944	0	6944	0	9356	25	9356
FULL QUERY																							
.NET	Full (release)	Average	100M	247881	1084	2	246795	931	4929	5860	5422	1	25441	1	22323	2	22323	141057	26034	105712	27879	24	27879
.NET	Full (release)	Average	9.6K	249712	2019	2	247691	1291	8399	9690	5422	1	24636	1	22818	10	22818	141711	27414	105946	28287	24	28287

Incident Report data

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS								Setup/GC (FS)		Request generation (FS)		Request validation (FS)		Request transformation (FS)		Setup/GC (BS)		Request validation (BS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Comm-unication, ms	Total Overhead, ms	Total Pro-cessing, ms	TCP Conversation	BS-FS traffic	RAM, KB	TCP conversation	RAM, KB	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
J2EE	Aggregate (release)	Minimal	100M	63834	196	1725	61913	12144	1705	13849	11787	800	13540	44	18650	25929	44895	3001	45407	925	12987	32939	46374
J2EE	Aggregate (release)	Minimal	10M	63599	129	1709	61761	12210	1705	13915	11787	823	7431	44	12541	25877	38703	3001	39255	886	13611	32839	46995
J2EE	Aggregate (release)	Minimal	56K	64512	1180	1696	61636	13790	1839	15629	11787	811	13382	44	18492	25789	44739	2997	45916	885	12539	32806	45907
J2EE	Aggregate (release)	Minimal	9.6K	70009	6508	1663	61838	12276	1771	14047	11787	786	12499	44	17609	26012	43748	3003	45328	877	7639	32779	41024
J2EE	Aggregate (release)	Average	100M	67092	292	1785	65015	27813	2365	30178	26796	878	7552	317	15305	26257	40713	5140	44350	907	13616	33301	47034
J2EE	Aggregate (release)	Average	10M	66052	160	1747	64145	27813	2365	30178	26796	797	13501	301	21233	25790	46623	5170	51123	950	7699	32884	41167
J2EE	Aggregate (release)	Average	56K	69387	3546	1706	64135	27945	2431	30376	26796	793	13453	302	21206	25798	46614	5178	50319	913	12606	32857	46072
J2EE	Aggregate (release)	Average	9.6K	86153	20316	1708	64129	27879	2365	30244	26796	795	13183	302	20936	25783	46352	5155	50308	913	13600	32889	47066
J2EE	Aggregate (release)	Large	100M	75001	426	1865	72710	56280	2695	58975	54009	831	13369	1253	14575	26356	47785	11508	52238	1034	7794	33593	41249
J2EE	Aggregate (release)	Large	10M	73428	403	1712	71313	56412	3091	59503	54009	822	7486	1233	8692	25808	41932	11388	47230	890	13753	32884	47209
J2EE	Aggregate (release)	Large	56K	81270	8026	1713	71531	56412	3091	59503	54009	823	7432	1236	8637	25813	41859	11578	47258	890	12702	32904	46175
J2EE	Aggregate (release)	Large	9.6K	117876	44552	1714	71610	56412	3685	60097	54009	822	13211	1236	14416	25828	47641	11619	53254	892	13696	32927	47170
J2EE	Aggregate (release)	Average	100M	67092	292	1785	65015	27813	2365	30178	26796	878	7552	317	15305	26257	40713	5140	44350	907	13616	33301	47034
J2EE	Subschema (release)	Minimal	10M	41405	174	1697	39534	12604	1771	14375	12115	788	13371	46	18699	16619	35890	1886	39996	909	13602	20983	37903
J2EE	Subschema (release)	Minimal	56K	42691	1447	1727	39517	12604	1771	14375	12115	788	13304	46	18631	16613	35822	1873	40202	939	7636	20985	31957
J2EE	Subschema (release)	Minimal	9.6K	48914	7682	1705	39527	12742	1639	14381	12115	795	7429	46	12756	16615	29874	1878	40211	910	13685	20988	38006
J2EE	Subschema (release)	Average	100M	44201	208	1659	42334	28798	2551	31349	27715	777	13425	332	14555	16596	38506	4146	38961	882	13741	21260	38073
J2EE	Subschema (release)	Average	10M	43814	203	1675	41936	28798	2431	31229	27715	786	7433	319	8563	16546	32514	4066	39345	889	13752	21005	38082
J2EE	Subschema (release)	Average	56K	47520	3838	1675	42007	28930	2497	31427	27715	787	13352	319	14461	16572	38412	4079	37026	888	12442	21037	36744
J2EE	Subschema (release)	Average	9.6K	65268	21509	1697	42062	28864	2431	31295	27715	783	12887	320	14017	16585	37986	4146	33549	914	7719	21011	32050
J2EE	Subschema (release)	Large	100M	51545	374	1684	49487	56405	2893	59298	50217	820	7424	1284	12400	16740	29588	10286	42265	864	13857	21177	38262
J2EE	Subschema (release)	Large	10M	51298	261	1653	49384	57919	3157	61076	50217	789	11944	1281	16920	16730	34107	10280	42787	864	7797	21093	32102
J2EE	Subschema (release)	Large	56K	59001	8045	1654	49302	57919	3157	61076	50217	789	13358	1277	18334	16701	35491	10226	36774	865	13548	21098	37874
J2EE	Subschema (release)	Large	9.6K	96605	55913	1655	39037	57853	3685	61538	50217	790	12914	1280	17890	16694	35077			865	12604	21063	37009
J2EE	Full External (release)	Minimal	10M	62211	130	1712	60369	14047	1783	15830	12109	791	11712	46	17053	26320	34388	1891	34899	921	7640	32112	32001
J2EE	Full External (release)	Minimal	56K	628959	1370	1699	625890	14046	1783	15829	12109	811	7432	47	12773	310548	30404	1912	41198	888	13634	3E+05	38317
J2EE	Full External (release)	Minimal	9.6K	~ 1 hour (est.)			0																
J2EE	Full External (release)	Average	100M	64974	264	1630	63080	2431	28787	31218	27704	775	13343	335	14472	27257	38545	4067	33991	855	12271	31421	36822
J2EE	Full External (release)	Average	10M	66238	295	1634	64309	2497	28919	31416	27704	778	12659	319	13789	27485	37842	4054	39126	856	12122	32451	36673
J2EE	Full External (release)	Average	56K	~ 10 minutes (est.)			0																
J2EE	Full External (release)	Average	9.6K	~ 1 hour (est.)			0																
J2EE	Full External (release)	Large	100M	70242	322	1656	68264	2893	57908	60801	55439	790	13376	1277	18352	26560	35717	10265	41941	866	13769	30162	38318
J2EE	Full External (release)	Large	10M	75972	341	1659	73972	3091	57842	60933	55439	792	13371	1275	18347	26438	35788	10207	43559	867	13616	36052	38164
J2EE	Full External (release)	Large	56K	~ 10 minutes (est.)																			
J2EE	Full External (release)	Large	9.6K	~ 1 hour (est.)																			
J2EE	Full Local (release)	Minimal	100M	41684	569	1526	39589					764	12819	48	18161	16501	35237	2135	39640	762	10892	20905	35230
J2EE	Full Local (release)	Minimal	9.6K	49128	8536	1517	39075					732	12884	55	18225	16320	35173	2062	39520	785	6494	20638	30831
J2EE	Full Local (release)	Average	100M	44850	213	1473	43164					738	13253	329	14382	16480	38306	4256	39243	735	12227	22099	36472

Incident Report data (cont'd)

BASIC (PARTIAL) QUERY				GENERAL TRANSACTION STATISTICS								Setup/GC (FS)		Request generation (FS)		Request validation (FS)	Request Validation (BS)	Request trans-formation (FS)		Setup/GC (BS)		Request validation (BS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Communication, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	BS-FS traffic	RAM, KB	TCP conversation	RAM, KB	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
.NET	Aggregate (release)	Minimal	100M	416772	135	2	416635	11766	1013	12779	11909	1	29728	2	29728	178744	41359	227	41359	1	27577	2E+05	37936
.NET	Aggregate (release)	Minimal	9.6K	422666	6392	2	416272	23360	4077	27437	11909	1	34760	2	34760	177993	41312	457	44057	1	29939	2E+05	38482
.NET	Aggregate (release)	Average	100M	415081	140	2	414939	27557	1283	28840	27160	1	41359	3	41359	177601	40424	324	40424	1	37936	2E+05	37400
.NET	Aggregate (release)	Average	9.6K	453112	37915	2	415195	57970	5319	63289	27160	1	44057	3	44057	177986	40897	308	40897	1	38482	2E+05	38214
.NET	Aggregate (release)	Large	100M	417530	385	2	417143	56295	1769	58064	54872	1	40424	5	40424	178040	39345	806	42756	1	39604	2E+05	38776
.NET	Aggregate (release)	Large	9.6K	494878	80789	2	414087	110859	5316	116175	54872	1	37208	5	37208	178406	43452	604	43452	1	39446	2E+05	40036
FULL QUERY																							
.NET	Subschema (release)	Average	100M	240750	108	2	240640	28537	1283	29820	28086	1	41564	3	41564	102833	34365	318	34365	1	27879	1E+05	27636
.NET	Subschema (release)	Average	9.6K	259838	20322	2	239514	28597	2400	30997	28086	1	42414	3	42414	103312	33602	326	33602	1	28156	1E+05	28139
.NET	Subschema (release)	Large	100M	242236	378	2	241856	57790	1823	59613	56313	1	34365	19	34365	103140	33433	850	34760	1	29856	1E+05	29939
.NET	Subschema (release)	Large	9.6K	284228	44266	2	239960	57910	4057	61967	56313	1	33602	5	33602	103297	33019	842	35856	1	28139	1E+05	28813
.NET	Full External (release)	Minimal	100M	269970	84	2	269884	12093	1013	13106	12236	1	35856	2	35856	119901	34004	408	36545	1	28813	1E+05	28419
.NET	Full External (release)	Minimal	9.6K	~6 hours (est.)																			
.NET	Full External (release)	Average	100M	267476	108	2	267366	28525	1283	29808	28074	1	36545	3	36545	117186	33173	567	35956	1	28419	1E+05	28657
.NET	Full External (release)	Average	9.6K	~6 hours (est.)																			
.NET	Full External (release)	Large	100M	270497	148	2	270347	57778	1823	59601	56301	1	35956	26	35956	120440	33264	798	35944	1	28657	1E+05	27203
.NET	Full External (release)	Large	9.6K	~6 hours (est.)		0	0																
.NET	Full Local (release)	Minimal	100M	245818	7612	350	237856					342	8942	43	8942	101945	27650	587	29834	8	27127	1E+05	26518
.NET	Full Local (release)	Minimal	9.6K	242497	6758	2	235737					1	30169	2	30169	101086	30569	220	30569	1	26657	1E+05	27044
.NET	Full Local (release)	Average	100M	236699	378	2	236319					1	29834	33	29834	102021	27933	556	30400	1	26518	1E+05	26594
.NET	Full Local (release)	Average	9.6K	256738	20296	2	236440					1	30569	3	30569	101794	30380	324	30380	1	27044	1E+05	26422
.NET	Full Local (release)	Large	100M	237936	532	2	237402					1	30400	39	30400	101773	28072	817	30169	1	26594	1E+05	26657
.NET	Full Local (release)	Large	9.6K	280826	44606	2	236218					1	30380	5	30380	100866	28538	863	31899	1	26422	1E+05	27503

Field Report data

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS							REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		DOM-PARSING REQUEST (BS)		DOM-PARSING OF STORAGE (BS)		DOM MANIPULATION (BS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Comm-unication, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	Total traffic	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
J2EE	SubSchema (optimized)	Average	100M	7284	82	7202					1	48646	1431	52617	2470	16876	5	17336	1760	21602	1534	57512	1	57512
J2EE	SubSchema (optimized)	Average	9.6K	7862	625	7237					1	46775	1419	51221	2570	15379	4	15839	1795	20102	1447	48408	1	48408
.NET	SubSchema (optimized)	Average	100M	3714	734	2980					1	6056	416	6056	547	7042	1	9726	789	10110	601	10110	625	10110
.NET	SubSchema (optimized)	Average	9.6K	3268	643	2625					1	14860	360	14860	477	14860	1	10110	607	10110	460	10110	719	11010

RAP Sheet data

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS							REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		DOM-PARSING REQUEST (BS)		DOM-PARSING OF STORAGE (BS)		DOM MANIPU-LATION (BS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Comm-unication, ms	Total Over-head, ms	Total Pro-cessing, ms	TCP Conversation	Total traffic	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
J2EE	Aggregate (pre-release)	Minimal	100M	19528	124	19404	3222	17748	20970	19040	2	66071	3889	72914	4906	85257	157	89427	5576	93464	4575	78983	299	80548
J2EE	Aggregate (pre-release)	Minimal	10M	18602	156	18446	3420	17748	21168	19040	2	84110	3813	91119	4796	87312	154	91212	5322	103047	4228	97049	131	98556
J2EE	Aggregate (pre-release)	Minimal	56K	20808	2338	18470	3828	19262	23090	19040	2	98604	3770	112530	4802	108008	153	112184	5338	116560	4288	118332	117	119839
J2EE	Aggregate (pre-release)	Minimal	9.6K	34776	12761	22015	3750	17748	21498	19040	1	119521	5351	27996	6569	23031	151	27155	5469	31499	4343	33767	131	35274
J2EE	Aggregate (pre-release)	Average	100M	19216	154	19062	3420	29785	33205	30483	2	57155	3829	63855	4740	75362	435	70999	5330	82914	4423	70448	303	72052
J2EE	Aggregate (pre-release)	Average	10M	18900	131	18769	3486	29719	33205	30483	2	99055	3810	105266	4781	110136	429	105661	5342	117432	4254	119648	151	121252
J2EE	Aggregate (pre-release)	Average	56K	26063	3817	22246	4212	29719	33931	30483	2	120321	5474	22283	6501	26054	418	21477	5414	33334	4238	36714	199	31151
J2EE	Aggregate (pre-release)	Average	9.6K	40238	21554	18684	4278	29719	33997	30483	2	36010	3806	42466	4769	46133	430	41729	5272	53631	4213	56902	192	51235
J2EE	Aggregate (pre-release)	Large	100M	20184	253	19931	3616	42221	45837	42323	2	33849	3745	47507	4957	55520	819	56473	5565	60806	4477	54758	366	56425
J2EE	Aggregate (pre-release)	Large	10M	22979	198	22781	3748	42089	45837	42323	2	114526	5457	25040	6731	20448	773	21630	5339	33492	4308	32021	171	33704
J2EE	Aggregate (pre-release)	Large	56K	24713	5592	19121	4738	42089	46827	42323	2	31721	3779	45442	4875	40459	778	41396	5268	53328	4242	52579	177	54246
J2EE	Aggregate (pre-release)	Large	9.6K	50733	31721	19012	4870	42155	47025	42323	2	51874	3759	65443	4791	60855	785	61936	5291	73761	4224	72874	160	74541
J2EE	Aggregate (pre-release)	Average	100M	19216	154	19062	3420	29785	33205	30483	2	57155	3829	63855	4740	75362	435	70999	5330	82914	4423	70448	303	72052
J2EE	Aggregate (release)	Average	9.6K	162009	39611	122398	4278	29719	33997	30483	2	78086	26302	102934	33506	68293	652	69254	33584	94806	28228	38605	124	40251
J2EE	Full (pre-release)	Minimal	100M	23023	220	22803	3366	17758	21124	19062	2	96122	4890	109473	6245	107849	232	104488	6242	117201	4995	116988	197	118516
J2EE	Full (pre-release)	Minimal	10M	22787	84	22703	3366	17758	21124	19062	2	34345	5010	40704	6222	40471	148	44176	6220	56915	4918	55207	183	56800
J2EE	Full (pre-release)	Minimal	56K	24504	2112	22392	3762	17758	21520	19062	2	54936	4944	61134	6210	60149	153	64010	6075	76749	4882	75688	126	77196
J2EE	Full (pre-release)	Minimal	9.6K	33962	11465	22497	4374	17758	22132	19062	1	75294	5019	81304	6218	80500	158	84551	6100	97266	4886	95863	115	97371
J2EE	Full (pre-release)	Average	100M	22954	147	22807	3300	29729	33029	30505	2	60514	4928	66786	6100	69976	460	66108	6141	79426	5023	81867	153	83492
J2EE	Full (pre-release)	Average	10M	22945	167	22778	3630	29861	33491	30505	2	57347	4967	63706	6172	63890	419	67249	6097	79991	4939	78891	182	80496
J2EE	Full (pre-release)	Average	56K	26541	3795	22746	4224	29729	33953	30505	2	77658	4933	84009	6215	83631	426	87026	6157	99769	4874	99097	139	100703
J2EE	Full (pre-release)	Average	9.6K	45738	22895	22843	4290	29729	34019	30505	1	97876	4934	104251	6316	104032	414	107664	6116	120403	4922	119566	140	121171
J2EE	Full (pre-release)	Large	100M	23521	235	23286	3430	42231	45661	42345	2	35978	4841	49376	6090	49420	828	51035	6288	56201	5060	58136	177	59804
J2EE	Full (pre-release)	Large	10M	23667	203	23464	3628	42099	45727	42345	2	81073	4939	87366	6206	87014	852	87796	6331	100952	4940	103179	194	104848
J2EE	Full (pre-release)	Large	9.6K	58529	31661	26868	4882	42165	47047	42345	1	114866	6607	21928	8236	21478	771	22532	6119	35487	4901	38137	233	32841
J2EE	Full (release)	Average	100M	79340	328	79012	3300	29729	33029	30505	2	77962	16637	94024	22635	59138	523	60148	22267	77089	16832	113975	116	115575
J2EE	Full (release)	Average	9.6K	109417	31837	77580	4290	29729	34019	30505	2	42127	16472	58456	20847	114069	478	121730	22976	31527	16699	78553	106	80152
J2EE / Sarvega	Full (release)	Average	100M	1095	254	841	3300	29729	33029	30505	82	11234	1	11234	1	9181	402	8331	1	8332	0	13269	354	14919
J2EE / Sarvega	Full (release)	Average	9.6K	31760	30987	773	4290	29729	34019	30505	80	11592	1	11592	1	9887	396	8875	0	8876	8	13703	287	15303
J2EE	SubSchema (release)	Average	100M	128371	251	128120	3300	29729	33029	30505	2	117906	28362	45586	34823	105117	500	104873	37229	40344	27067	74305	137	75959
J2EE	SubSchema (release)	Average	9.6K	162546	40833	121713	4290	29729	34019	30505	2	81944	26018	106877	33415	59572	487	60616	33454	86318	28213	38547	124	40193
J2EE	SubSchema (optimized)	Average	100M	66789	207	66582	3300	29729	33029	30505	416	95158	13971	115251	17895	48006	569	48062	17593	67557	15921	33720	217	35357
J2EE	SubSchema (optimized)	Average	9.6K	115446	51891	63555	4290	29729	34019	30505	374	73983	13500	86182	16812	119306	473	120344	18815	27214	13354	102156	227	103782
.NET	Aggregate (pre-release)	Minimal	100M	15140	1032	14108	3242	18731	21973	20003	1	9913	3500	14136	3974	14284	2	14284	3880	14030	2721	9440	29	14136
.NET	Aggregate (pre-release)	Minimal	10M	14949	686	14264	3321	17549	20870	20003	1	9652	3820	9909	3882	21873	3	21873	3899	21829	2631	14103	29	9909
.NET	Aggregate (pre-release)	Minimal	56K	17828	3179	14649	3758	18262	22020	20003	1	10107	3683	9913	4188	14759	3	14759	4221	20138	2524	13694	29	9913
.NET	Aggregate (pre-release)	Minimal	9.6K	26418	12062	14356	3945	19141	23086	20003	1	9909	3692	10107	3842	21908	2	21908	4257	27406	2533	13667	29	10107

RAP Sheet data (cont'd)

BASIC (PARTIAL) QUERY				GENERAL TRANSACTION STATISTICS							REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		RESPONSE GEN (BS)		VALIDATION (BS)		VALIDATION (FS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Communication, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	Total traffic	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, KB
.NET	Aggregate (pre-release)	Average	9.6K	36934	23134	13800	4049	31152	35201	31902	1	22521	3135	23416	4356	21259	4	21259	4015	14989	2250	25789	40	23416
.NET	Aggregate (pre-release)	Large	100M	15182	712	14470	3951	42512	46463	44207	1	22602	3367	22335	4243	20459	5	20459	4281	25764	2524	23020	50	22335
.NET	Aggregate (pre-release)	Large	10M	14701	366	14335	4002	42759	46761	44207	1	22813	3328	22602	4224	15152	5	15958	4224	15152	2502	23089	50	22602
.NET	Aggregate (pre-release)	Large	56K	19617	5987	13631	4936	42319	47255	44207	1	23020	3285	22813	3910	13993	5	14210	3910	13993	2469	23428	50	22813
.NET	Aggregate (pre-release)	Large	9.6K	48198	33266	14932	5103	42365	47468	44207	1	23416	3755	23020	4341	14486	15	14703	4341	14486	2419	23872	59	23020
.NET	Aggregate (release)	Average	100M	830500	1786	828714					4	9647	176233	38249	234564	37004	4	37004	234899	37728	182967	68105	43	68105
FULL QUERY																								
.NET	Full (pre-release)	Average	100M	22538	667	21871	3712	29014	32726	31925	1	28451	4830	26548	6057	16156	4	16156	6085	23986	4855	23874	40	26548
.NET	Full (pre-release)	Average	10M	22484	1492	20992	3789	29470	33259	31925	1	14809	4660	25642	5890	26240	4	26240	5985	25852	4411	25409	41	25642
.NET	Full (pre-release)	Average	56K	25947	4440	21508	4001	29932	33933	31925	1	25130	4668	25799	5824	15502	4	15502	6216	15303	4628	25555	167	14809
.NET	Full (pre-release)	Average	9.6K	68321	45872	22449	4030	31199	35229	31925	1	26535	5036	25130	6508	15617	19	15617	6120	23500	4724	23425	41	25130
.NET	Full (pre-release)	Large	100M	23427	1135	22293	3970	42499	46469	44228	1	26548	4904	25186	6648	15617	27	15617	5992	23553	4666	23425	55	25186
.NET	Full (pre-release)	Large	10M	22052	694	21358	3976	42716	46692	44228	1	25131	4728	26377	5900	25579	5	25579	5997	17443	4677	28659	50	26377
.NET	Full (pre-release)	Large	56K	27584	6020	21563	4931	42340	47271	44228	1	26377	4770	22828	6147	24919	5	24919	5869	16330	4721	24530	51	22828
.NET	Full (pre-release)	Large	9.6K	55156	33389	21768	5091	42376	47467	44228	1	22828	5008	22638	6153	24051	5	24051	5802	15875	4749	25839	51	22638
.NET	Full (release)	Average	100M	473258	573	472685	3970	42499	46469	44228	2	26457	102466	43913	134347	26117	13	26117	134821	25898	100964	46836	72	46836
.NET / Sarvega	Full (release)	Average	100M	289	184	105	3970	42499	46469	44228	1	5082	1	8452	0	8452	4	8452	0	8452	0	5082	99	3550
.NET / Sarvega	Full (release)	Average	9.6K	25980	25935	45	5091	42376	47467	44228	1	10472	1	8491	0	8491	4	8491	0	8491	0	12082	39	12082
.NET	Sub-schema (release)	Average	100M	833722	536	833186	3970	42499	46469	44228	1	46836	177191	41253	235413	38905	16	38905	235376	37286	185132	69336	57	69336
.NET	Sub-schema (optimized)	Average	100M	344042	199	343843	3970	42499	46469	44228	1	27362	58491	27362	77044	26270	1	22715	118333	22911	89933	22911	40	22911

Baseline data

TEST CASE SPECIFICS				GENERAL TRANSACTION STATISTICS							REQUEST GENERATION (FS)		REQUEST VALIDATION (FS)		START-UP/GC (BS)		REQUEST VALIDATION (BS)		DOM-PARSING REQUEST (BS)		DOM-PARSING OF STORAGE (BS)		DOM MANIPULATION (BS)	
Platform	Schema	Data Size	Network	Total Time, ms	Total Comm-unication, ms	Total Overhead, ms	Total Processing, ms	TCP Conversation	Total traffic	RAM, KB	Time, ms	RAM, KB	Time, ms	RAM, MB	Time, ms	RAM, MB	Time, ms	RAM, MB	Time, ms	RAM, MB	Time, ms	RAM, MB	Time, ms	RAM, MB
J2EE	Non-JXDD	V. Small	100M	186	34	152	1935	1660	3595	2391	1	13221	21	13354	34	10109	1	10157	21	10292	31	13718	43	14333
J2EE	Importing Pre-release	V. Small	100M	271	53	218	2057	1781	3838	2502	1	11336	54	11535	39	13345	1	13397	46	13603	41	11991	36	12606
J2EE	Importing/using Pre-release	Small	100M	24359	42	24317					3	58484	4988	71854	8017	24476	3	24767	6328	31160	4946	78620	32	79280
J2EE	Importing Full Release	V. Small	100M	29335	51	29284	2120	1853	3973	2646	1	40390	12808	54542	31	15198	1	15253	64	7303	16334	73388	45	74004
J2EE	Importing/using Full Release	Small	100M	82714	81	82633					5	78927	19420	97218	21168	49208	5	49714	21022	68428	20159	121759	854	117092
.NET	Non-JXDD	V. Small	100M	137	107	30	1594	1394	2988	2528	1	5686	5	5686	7	5464	1	5464	6	5464	5	5686	5	5686
.NET	Importing Pre-release	V. Small	100M	21642	112	21530	1574	1495	3069	2739	1	6743	5098	16466	5670	16289	1	16289	5649	14764	5105	16513	6	16513
.NET	Importing/using Pre-release	Small	100M	22577	123	22454					5	46291	4810	54306	5940	14746	1	14746	5839	22618	5846	29218	13	29218
.NET	Importing Full Release	V. Small	100M	514815	93	514722	1575	1499	3074	2744	1	16222	123193	27664	134535	26760	1	26760	133805	25666	123181	27789	6	27789
.NET	Importing/using Full Release	Small	100M	478769	893	477876				2744	1	29218	102593	28489	136520	25980	1	25980	136376	27412	102378	46483	7	46483
J2EE	Importing Full Release	V. Small	100M	29335	51	29284	2120	1853	3973	2646	1	40390	12808	54542	31	15198	1	15253	64	7303	16334	73388	45	74004

APPENDIX B – Test Results Primer

GJXDM Performance Testing Primer

June 28, 2004

Introduction

This document describes the testing setup and the procedure followed to obtain the GJXDM performance testing results. This document then presents a primer for interpreting the GJXDM performance testing results. It accompanies the raw data in Excel spreadsheets developed by the George Washington University (GWU) testing team.

Test setup and procedure

Test environment

Hardware details

The test environment consists of four pieces of equipment: a back-end server, a front-end server, a router, and a client laptop computer. The following diagram depicts the network and equipment used for the GJXDM performance testing:

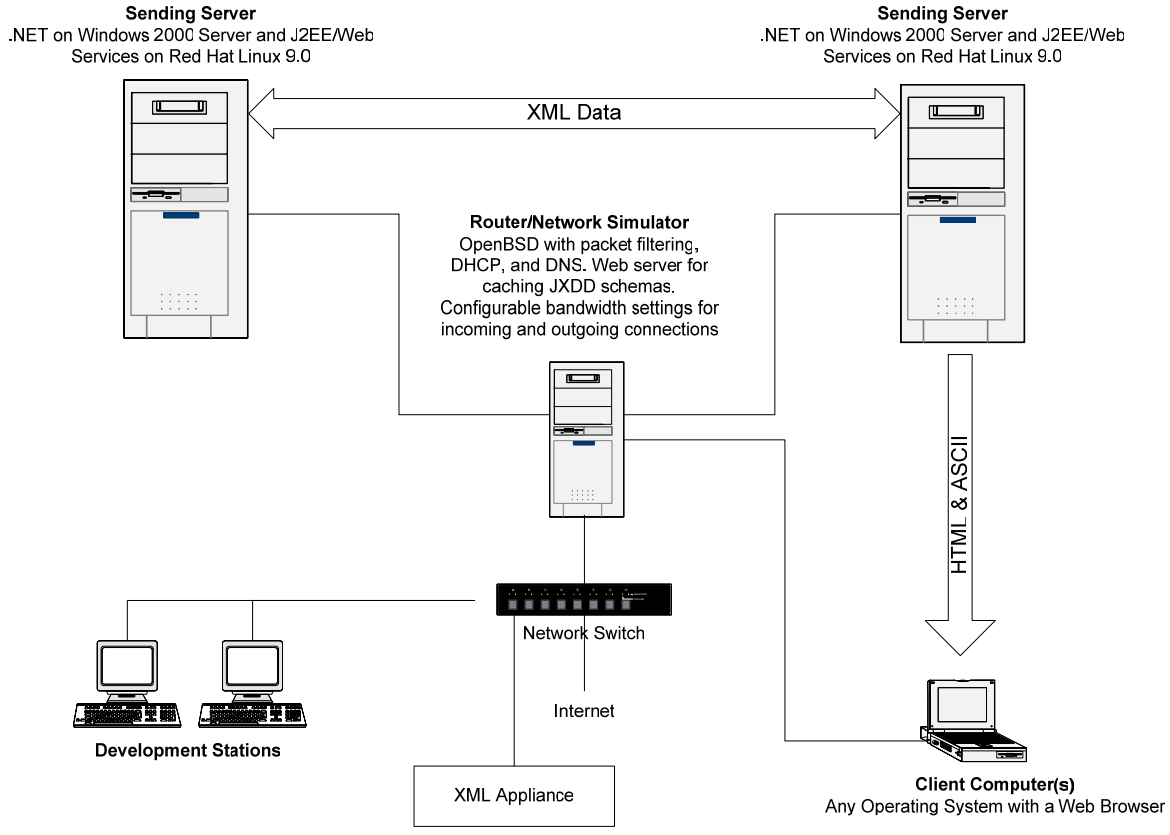

Figure 23 – Hardware setup diagram

Back-end server

- 450MHz Pentium II CPU
- 256MB RAM
- Configured as a "double-booted" computer, allows choosing between Windows 2000 Advanced Server or RedHat 9.0 Linux operating system.

Front-end server

- 366MHz Pentium II CPU
- 256MB RAM

- Configured as a "double-booted" computer, allows choosing between Windows 2000 Advanced Server or RedHat 9.0 Linux operating system.

Network Simulator (router)

- 233MHz Pentium II CPU
- 128MB RAM
- 4 PCI network cards
- OpenBSD 3.5 operating system

Client Laptop Computer

- 1.2GHz Pentium III Mobile CPU (Dell Latitude C600)
- 256MB RAM
- Windows 2000 Advanced Server operating system

Network Switch

A simple network switch connected the test hardware to the Internet. This switch provided connection to the Internet (e.g., if an external schema was needed), while isolating the test environment from outside traffic that would affect the predictability of network performance. The model used was a Netgear FS608 8 Port 10/100 switch.

Software details

Test applications architecture

The front-end server contained a Web application, which allowed the tester to change settings and initiate test case runs. The front-end server communicated with the back-end server to complete the transaction. The results of the test runs displayed in the browser on the client laptop that the tester used to access the Web interface.

The sample applications used the platforms, tools, and APIs typically used for implementing the XML data exchanges on each platform selected for testing. The Web Services data exchange model used on both platforms appeared as the most likely approach taken by future implementers of the GJXDM. Recent publications about GJXDM (i.e. XSTF report) also recommended the Web Services approach. The sample applications minimized the steps not directly related to the performance of the schema and only included functionality relevant to performance testing; the business logic followed the outline in the test cases.

.NET platform (both front- and back-end server)

The C# test applications developed using Microsoft .NET Framework version 1.1 and deployed on a Windows 2000 Advanced Server. The front-end server and back-end server applications ran on a Microsoft IIS Web Server. The Web

interface on the front-end server was developed using ASP.NET, and both servers use .NET Web Services API classes to communicate.

J2EE platform (both front- and back-end server)

The Java test applications developed using Sun Microsystems Java SDK 1.4.2 on Red Hat Linux 9.0. The Java Web Services Developer Pack (JWSDP) and Java API for XML Processing (JAXP) implemented the data exchange. JWSDP's distribution of Tomcat 5.0 Web server is the servlet container for the Java test applications.

Network Simulator / Router software

OpenBSD 3.5 distribution included a packet filter (PF), which allowed setting bandwidth limits on network connections. This filter modified the network speed between the servers in the tests aimed to determine the time it takes to transfer GJXDM instances over the network. While using packet filter queuing does not accurately represent all the real network features, it provides a reflection of the probable transfer speeds future implementers of GJXDM should expect.

XML Appliance

Processing text-based XML is CPU-intensive, typically consuming 45% to 80% server cycles for compute-intensive operations such as XSLT, Schema Validation, and XML Security algorithms. This makes scaling XML-based Web services expensive and creates business-impacting latencies. Sarvega's XML Appliance utilizes a highly optimized binary data stream that solves these XML processing problems. The XML Appliance used was a Sarvega XPE 2000 XML Appliance.

Client machine software

Since the primary aim of this testing was to determine the performance of data exchange transactions using GJXDM, the specifics of the client machine that drove the tests are not important and were not captured. The tests ran using Internet Explorer 6 on the client machine.

Test transaction overview

The two servers exchange data using the Web Services Simple Object Access Protocol (SOAP). The front-end server receives a HTTP POST request from the client machine. The client request contains parameters that instruct the front-end server to invoke a Web Service on the back-end server using a SOAP request message. The SOAP message sent to back-end server contains several string parameters that specify the characteristics of expected response (i.e., whether the response checks against the full schema and whether it will be of minimal, average, or large size). In most cases, the request also contains XML data that works on the back-end server. The back-end server returns the result in a similar

SOAP message, except it may also contain error messages. Then the front-end server makes a separate Web service request to obtain the metrics collected on the back-end server during the last transaction.

Changing parameters

- Schema type: selects from the options available in the Web interface
- Response data size: selects from the options available in the Web interface
- Test case scenario: selects from the options available in the Web interface (this only applies to test cases that include sub-cases, such as Inmate Record.)
- Network settings: in the root home directory on the router, run one of the scripts (e.g., *pf.conf.56K* to change the OpenBSD packet filter (pf) settings and limit the bandwidth to/from the servers to the specified amount.

Collecting metrics

Most of the results were collected within the test applications using the appropriate APIs and display on the Web interface when the test is completed. The Web interface displays the metrics for both servers in character-delimited format, ready to import into a spreadsheet application such as Microsoft Excel. Other methods of collecting information include using the output of performance monitoring tools available on the platform (e.g., *Windows Performance Monitor* or *top* on Linux) and running a network analyzer (such as Ethereal, *tethereal*, or *tcpdump*) on one of the servers or on the router.

Test procedure

In all test cases, *server1.jxddtest.net* works as the back-end server, and *server2.jxddtest.net* works as the front-end server.

1. From the client laptop, open the Web interface page on the front-end server. When testing .NET applications, the address is http://server2.jxddtest.net/FrontServer/UIMain.aspx. When testing J2EE applications, the Web interface is located at http://server2.jxddtest.net/jxddtest_frontend/.
2. Select the test parameters (schema type, response size) on the Web interface.
3. Change/verify the network settings on the router (e.g., *cat /etc/pf.conf*)
4. When testing on Windows, switch KVM to *server1* and start the Ethereal network analyzer capture on *server1*. On Linux, start the *tethereal* as root in the shell (*-w <log_file>*). This step performs once for every unique transaction and is separate because the network sniffers tend to drop packets when CPU use is high. In addition, they add CPU load.
5. Submit the request on the Web interface from client machine.

6. Observe the output of Performance Monitor or *top* on Linux. This is optional and provides a general idea of what is happening. These tools add an extra load, so re-run the request without them when recording metrics.
7. Wait until the result displays in the client machine Web browser. Verify that there were no processing errors on either server (reported in the Web interface and in the log files of IIS and Tomcat). If the test case sequence includes an XSLT transformation, verify that the HTML output is correct.
8. If an Ethereal/tethereal capture starts, stop it, and save the capture file. Naming convention: *{schema type}_{data size}_{network speed}.cap* (e.g., *ref_lrg_100.cap* for a transaction that uses 'Full' schema , large data size, and network speed of 100Mbps). For each test case/platform combination, create a separate directory to store the captured files.
9. Copy the character-delimited transaction record shown in the Web interface and append it to a temporary text file or import into Excel directly.
10. If there is an Ethereal capture available, right-click on the *server2->server1* POST request displayed in the Ethereal workspace. If this is the first time this transaction is run, open 'Follow TCP stream' menu option, record the 'TCP conversation' number in the spreadsheet, and save the TCP stream as {*schema type}_{data size}_{network speed}.txt*. The stream capture files can be used later to examine the SOAP protocol overhead, etc.
11. *Statistics>Conversations* menu displays the number of bytes transmitted in each direction, which should be copied to 'FS-BS traffic' and 'BS-FS traffic fields of the transaction row.

Results spreadsheet

Format overview

The layout of the results developed by the GWU team differs from the one suggested in original test plan. The original design did not break the transaction into multiple steps and record the metrics for these steps separately. For example, it proposed reporting one "CPU time" metric for a transaction. However, since the transaction is between two servers, reporting the sum of all times spent in key steps of the transaction would hide important information such as the distribution of processing time between the steps.

Furthermore, the original layout proposed putting all collected metrics in one table. While this approach has its benefits (especially for spreadsheet manipulation), it has some significant drawbacks: some of the metrics may not be applicable for all cases, resulting in a sparsely populated result matrix, which is harder to browse and interpret.

The updated data layout tries to resolve these and other issues and present the data in a clear way closely matched to the organization of the test cases and sample applications. A Microsoft Excel file presents the test results using multiple tabs.

The first four tabs names are *Inmate Record*, *AMBER Alert*, *Incident Report*, Field Report and *RAP Sheet* test cases. The AMBER Alert test case tab also includes the data for the Reduced Tag Names (RTN) test case. The RTN case bases itself on AMBER Alert and its data, and combining them emphasized their comparison.

The Inmate Record, AMBER Alert, and RAP Sheet test cases include test cases to support the XML Appliance platform option.

The *Baseline* tab contains information about several small transactions aimed as determining the base overhead from using the XML tools and the network and are considered special test cases.

The *Notes* tab contains useful information collected during the test case development, and ideas for future investigations. It may also contain extended descriptions of solutions to problems encountered.

The *Issues* tab tracks significant problems encountered by the test team while developing or running the test applications. The status field is color coded to assist in readability. The description and solution fields have extended information about the problem and its solution tracking by appending UPDATE (xx/yy/zz) sections.

Test case results layout

Each row represents one transaction. The grouped rows represent a transaction, each group containing the test runs of one transaction at different network speeds. In Inmate Record and AMBER Alert cases, the rows are also grouped by the sub-case (please refer to their respective descriptions for more information).

Transaction parameters: the Y axis

The first four values in a row are transaction parameters. Their combination describes the test transaction. In Excel, the frozen parameters (*Platform*, *Schema*, *Data Size*, and *Network)* are static to differentiate from the collected data.

- **Platform*:*** shows whether the test application ran in the .NET environment (Windows) or on the J2EE platform (Linux). The Platform value also indicates if the XML Appliance platform option was evaluated. XML Appliance platform option simply added the XML Appliance to either the .NET or the J2EE platform.
- **Schema*:***
  - All schemas are marked either as 'release' or 'pre-release', which indicates whether the schema refers to (or is based on) the full production release version 3.0 of GJXDM or a Pre-release version (3.0.0.0).
  - 'Aggregate' schema is a self-contained schema that is an application-specific subset of the GJXDM. In the original test plan, this schema referred to 'Partial Schema'.
  - 'Full' schema imports the full GJXDM to support object inheritance.
  - A 'Subschema' is a subset of a schema used by programs at runtime. It consists of all the data elements, records, sets, and areas defined in the schema or a subset thereof. It includes database records and can include logical records as well as logical-record paths. For this testing, 'Subschema' is a hybrid approach that defines a collection of imports from the GJXDM and allows the XML parser, if supported, to choose which ones to use. The construct of this sub-schema is hybrid in the sense that it contains the ability for the XML instances to use both 'Aggregate'- and 'Full'-style constructs and is a better representation of the field. For the purposes of the GJXDM performance testing, it serves as a baseline comparison of the sub-schema to full schema performance. This variation tested thoroughly in the Incident Report test case and ran as a limited additional test with the rest of the test cases.
  - Some of the test cases may provide additional schemas. Information about these special schemas is included in the test case descriptions.
  - NOTE: the original test plan also specifies a 'Bare Bones Schema' as a schema type. The 'bare-bones schema' concept is not directly applicable to all test cases in the same sense as 'Aggregate' or 'Full' schema types and discussed only within the relevant test cases.
- **Data Size*:*** specifies the size of the GJXDM instance(s) used in the test run.
- **Network*:*** specifies the available bandwidth set on the router's packet filter. For most transactions, there are four network speeds used during the test run

– 100Mbps, 10Mbps, 56Kbps, and 9.6Kbps. If a transaction took extended time to complete and/or the probable transaction time was available based on existing data, the team saved time by omitting some of the network speed variations.

Collected metrics: the X axis

The transaction parameters immediately follow a number of fields collectively called 'General transaction statistics'. These include:

- **Total transaction time:** time elapsed from the moment the user request is registered on the front-end server till the XML data retrieved from the back-end server is transformed (if applicable) and transaction results are ready to be written out to the user's browser.
- **Total communication (commo):** time spent by activities such as packaging the XML data into SOAP format, establishing connection to the remote server, and sending/receiving data. Defined for the entire transaction and represents the total transaction time minus processing and overhead. Although this is a somewhat rough metric, it is a better representation of communication than the calculated or measured pure transmission time. While pure transmission time calculates knowing the network characteristics, this metric cannot because it includes the platform-dependent overhead.
- **Total overhead:** time spent preparing the test run (total for both the front-end and back-end server). Because of the difference in the testing platforms, the programmer may need to implement some testing-specific steps that would not be present in the live application and excluded from the final transaction time calculations. One example of this is the need to call the garbage collector in the beginning of a J2EE application to receive consistent RAM usage metrics. A live application would not be doing this and the testing protocol needs to separate this step from others.
- **Total processing:** the sum of all times spent in the instance generation, parsing, validation, and transformation steps (the composition may vary for different test cases).
- **FS-BS traffic** *(front-end to back-end):* total number of bytes sent from the front-end server to the back-end server during the TCP connection established for the exchange. This information is retrieved from examining the network "conversation" recorded using a standard packet sniffer and represents bytes sent/received on both sides, including the lower-level protocol information (IP headers, etc.). Note that this metric will be different for different network speeds because of different packet breakdown.
- **BS-FS traffic***:* same as FS-BS traffic except the direction of transmission
- **Total traffic***:* is the sum of the two previous metrics (BS-FS and FS-BS traffic).
- **TCP conversation***:* the number of bytes exchanged in the TCP payload; it is always the same for a given transaction because it does not include re-sent packets and packet header information.

The remainder of the transaction row consists of multiple 'Time/RAM' pairs recorded at different steps of the transaction. The steps display on the top of the sheet in the headings with alternating colors and follow the transaction sequence. The label for each step is followed by '(FS)' or '(BS)', specifying that the operation was performed on the front- or back-server, respectively.

- **Time**: the 'elapsed time' recorded for the action, expressed in milliseconds. The time recording native to the platform API is the basis on this metric.
- **RAM***:* the amount of RAM used by the CLR on .NET, or by JVM on J2EE, expressed in kilobytes. Since the testing suite is the only application running on the platform, the variations of this metric show the amount of memory a particular step requests from the host machine.

The steps may include the following:

- **Setup/GC**: the overhead caused by application activities necessary for testing, but not necessary for normal operation. This metric is higher on the J2EE platform because the test application needs to call garbage collection, which always takes several hundred milliseconds.
- **Response/request generation**: retrieves a string of XML data (instance) for further manipulation. The test plan specifies that the transaction start with an existing instance provided in the test case, and this is the time for reading the XML from a file. In most cases, this metric is low, because of the buffering done by the platforms – the read() call returns the String object reference before reading it all into memory and proceeds to read the data into memory in the background.
- **Response/request validation**: provides a complete validation the XML data. Only successful validation times are included in the result data because in the negative validation case, the validation subroutine may exit at any point. This step includes both parsing the schema (possibly with imported schemas) and document parsing.
- **DOM parsing**: includes reading and parsing the XML string, creating the in-memory DOM representation.
- **DOM manipulation**: adds/modifies a number of nodes in the DOM tree.
- **Transformation:** applies an XSLT stylesheet to the XML data.
- **Serialization and writing to file**: in case the transaction includes an update to the in-memory DOM tree, this step converts the DOM tree into a string of XML data and writes it onto the hard drive.

Individual test case information

This section describes in detail the implementation of the test cases. Whenever possible, mentioning the data/schema/stylesheet files used and position them in the test data package also made available online.

Baseline

This special test case captures the overhead of starting the environment itself and using the XML APIs. All sub-cases represent a simple "RAP Sheet"-like transaction, which sends a request XML data stream to the back server and receives XML data stream in return. The request and response validates when it sends or receives, for all four validations, and the response transforms on the front-end server. For simplicity, the testing used the same XML file for request and response. Five baseline results reported for each of the two platforms:

- **Non-JXDD**: This test represents the smallest possible transaction. It is based on an extremely small non-JXDD schema (i.e., two elements in own namespace) and uses a minimal XML transformed by a minimal XSLT stylesheet.
- **Importing Full Release**: The schema imports the full 3.0 release of GJXDM, but does not actually use elements from the imported namespace.
- **Importing Pre-release**: The schema imports the Pre-release version of the GJXDM, but does not actually use elements from the imported namespace.
- **Importing/using Full Release**: The schema imports the full 3.0 version of the GJXDM, and the XML data file uses elements from the imported namespace. This sub-case tries to determine whether the XML processing metrics change based on the actual usage of the elements.
- **Importing/using Pre-release**: The schema imports the Pre-release version of the GJXDM, and the XML data file uses elements from the imported namespace.

Inmate Record test case

The test divided into a number of sub-cases, or scenarios: Alias Add, Alias Address Add, Address Update Scenario, Detention Add, Detention Update, Person Update, Basic Query, and Full Query. Similar scenarios existed and omitting the redundant scenarios left Alias Add, Address Update, Basic Query, and Full Query. The first two modify an XML record on the back-end server, and the other two request data from the back-end server.

All original test runs used the Pre-release version of the GJXDM schema and conform to either 'Aggregate' or 'Full' schema. Three additional tests ran using the full 3.0 release schema: 'Aggregate', 'Full', and 'SubSchema'. All three used average data size and 100 Mbps network speed to determine the time difference on the validation step, dominated by schema parse time previously.

Inmate Record test case scenarios that modify an XML record

Alias Add and Address Update scenarios involve sending a short Inmate Record request to the back-end server, which uses the information from the request and makes changes to the XML file designated as the 'storage'. The XML data used as storage is one of the full Inmate Record files taken from the Full Query scenario. The 'storage' file (of the requested size and conforming to the requested schema) reads into memory, modified based on the request, and written back to the file system as a temporary file.

Alias Add:

On the front-end server:

1. Create an Inmate Alias instance from file to be used as a request (an Inmate node with one Subject, always the same size, but conforming to one of the two schemas)
2. Validate the request instance XML
3. Send the request XML to the back-end server in a SOAP request envelope (the request envelope contains information about the schema and size of the storage file to be used)

On the back-end server:

4. Validate the request XML
5. Create the in-memory DOM structure from the request XML
6. Read in the storage XML file
7. Create the in-memory DOM structure from the storage XML
8. Extract the ClientID value from the request using an XPath expression
9. Extract from the storage the Inmate node with matching ClientID
10. Extract from the request the Subject node
11. Add the Subject node from request to the Inmate node from storage
12. Extract the IDs from the first Subject in the Inmate storage and the Subject in the request XML using XPath.
13. Add a SameAsRelationship node with object and subject attributes set to these IDs.
14. Serialize the storage XML and write it to the file system as a new file.

Address Update:

On the front-end server:

1. Create an Inmate Alias instance from file to be used as a request (an Inmate node with one Subject, always the same size, but conforming to one of the two schemas, depending on user input)
2. Validate the request instance XML
3. Send the request XML to the back-end server in a SOAP request envelope (the request envelope contains information about the schema and size of the storage file to be used)

On the back-end server:

4. Validate the request instance XML
5. Create the in-memory DOM structure from the request XML
6. Read in the storage XML file (Inmate Record of the requested size and conforming to the requested schema).
7. Create the in-memory DOM structure from the storage XML
8. Extract the ClientID value from the request using an XPath expression
9. Extract from the storage the Residence node belonging to the first Subject within the Inmate record with matching ClientID using XPath
10. Extract the Residence node from the request using XPath
11. Traverse the two node trees starting at the Residence node, comparing the element values and if they differ, setting the values in storage Residence equal to the ones in the request Residence. Before this is completed, the application ensures that the two node trees are structurally the same and normalizes the XML data. The procedure results in 15 updates to the Residence node tree.
12. Serialize the storage XML and write it to the file system as a new file.

Inmate Record test scenarios that retrieve an XML record

Full Query and Basic Query scenarios involve sending a short Request record (a minimal Inmate Record) to the back-end server, which responds with a Full or Basic Inmate Record XML data of varying sizes and conforming to either 'Aggregate' or 'Full' schema. The response processes on the front-end server using the matching XSLT stylesheet and then sent to the user. Full Query responses of different sizes contain a varying number of Detention records. Basic Query responses are always one size (Average).

Full Query / Basic Query:

On the front-end server:

1. Create a Request instance from file (Inmate element with one Subject, always the same size, but conforming to one of the two schemas, depending on user input)
2. Validate the request instance XML
3. Send the request to the back-end server in a SOAP envelope

On the back-end server

4. Validate the request XML
5. Create a full response Inmate Record from file (this is an Inmate Record with multiple Subject elements (aliases) and Detention elements)
6. Validate the response XML
7. Send the response back to the front-end server

On the front-end server:

8. Validate the response XML
9. Transform the XML data using the matching XSLT stylesheet
10. Write the transformed HTML content to user's browser

Field Report test case

This test case was included in order to evaluate the performance characteristics of an extremely short transaction. The Field Report entry transaction was selected due to the minimum number of fields and relatively small size of the transaction.

RAP Sheet test case

This test case involves requesting the RAP Sheet information from the back server and displaying it on the user's screen. The SOAP envelope sent by the front-end server includes a special small 'Request' XML string which contains information such as <jdd:PersonFBIID />.

All original tests ran with the Pre-release version of the GJXDM schema and conform to either 'Aggregate' or 'Full' schema. Three additional tests ran using full 3.0 release schema: 'Aggregate', 'Full', and 'SubSchema'. All three use average data size and 100 Mbps network speed to determine the time differences on the validation step, dominated by schema parse time previously.

On the front-end server:

1. Create a Request instance from file (i.e. a RAPSheet with one element, Introduction.) The request is always the same size, but conforms to one of the two schemas, depending on user input.
2. Validate the request instance XML

3. Send the request to the back-end server in a SOAP envelope

On the back-end server

4. Validate the request XML
5. Create a response RAP Sheet from file (e.g. this is a RAPSheet element with multiple Subject, Organization, and Cycle elements in it)
6. Validate the response XML
7. Send the response back to the front-end server

On the front-end server:

8. Validate the response XML
9. Transform the XML data using the matching XSLT stylesheet
10. Write the transformed HTML content to user's browser

AMBER Alert test case

This test case implements the AMBER Alert Get transaction, during which the front-end server requests an AMBER Alert message from the back-end. There is no data size variation within the case, and all data are 'Average' sized.

All original tests ran with the 3.0 release version of the GJXDM schema and conform to either 'Aggregate' or 'Full' schema. Two additional tests ran using full 3.0 release-based 'SubSchema', and one with a 'Full' Pre-release version for performance comparison.

There is also a special schema used: 'Aggregate Flat'. This schema is not "flat" in the full sense, but rather a special kind of schema derived from GJXDM. "Flattening" GJXDM (i.e. eliminating the object inheritance and making all objects "self-contained") is impossible because GJXDM is an indeterminate schema. Instead, the sub-schema author created a "sectioned data model", which uses 'groups' and 'attributeGroups' to eliminate the duplicate elements that would emerge from flattening an indeterminate schema. Including this schema into the tests provides data that evaluates the impacts of abstraction.

Based on the AMBER Alert test case, the Reduced Tags Name test case determines the impact of GJXDM's long tag names and presents its results on the same tab as a sub-case. The instances of AMBER Alert Reduced Tags have the same structure and processing, but have all tag names replaced with the 'smallest possible' substitute, generated automatically during the test case development (see the test plan for more information on this methodology). The Excel tab contains the detailed information about the size of the instances of AMBER Alert and AMBER Alert – Reduced Tags.

On the front-end server:

1. Create a SOAP request packet with the requested schema type and data size; there is no XML data sent in the request
2. Send the request to the back-end server

On the back-end server:

3. Create a response AMBER Alert from file
4. Validate the response XML
5. Send the response back to the front-end server

On the front-end server:

6. Validate the response XML
7. Transform the XML data using the matching XSLT stylesheet
8. Write the transformed HTML content to user's browser

Arrest Incident Report test case

This test case implements a "push" transaction during which the front-end server sends a report to the back-end. The report transforms on the front-end before sending and the back-end acknowledges the receipt of the message by responding with a 'no errors' SOAP envelope containing the ID of the report that was received.

The three data sizes are Minimal, Average, and Large and three schemas used in this test case: 'Aggregate', 'Full', and 'SubSchema'.

The 'Full' sub-case divides into 'Local' and 'External', which obtains the GJXDM from the server file system or OJP's Web site, respectively. This test aims to determine the difference caused by varying the full schema location.

On the front-end server:

1. Create a Report instance from file (an IncidentReport with IncidentSubject, IncidentVictim elements, etc.) The instance can be one of three different sizes and conforms to one of the three schemas, depending on user input.)
2. Validate the report instance XML
3. Transform the report using the matching XSLT stylesheet
4. Send the report to the back-end server in a SOAP envelope

On the back-end server

5. Validate the report XML

6. Acknowledge receipt by getting the ID from the SOAP envelope and sending it back in an otherwise empty SOAP envelope, meaning 'received'.

On the front-end server:

7. Notify the user upon receiving report.

References

Title	URL
[XSTF report] "Structure and Design Issues for Developing, Implementing, and Maintaining a Justice XML Data Dictionary", report of the Justice XML Structure Task Force (XSTF), 13 April 2004	http://justicexml.gtri.gatech.edu/JusticeXMLStructureTaskForceReport_a.pdf
OpenBSD FAQ, PF manual	http://www.openbsd.org/faq/pf

APPENDIX C – Acronym List

Acronym	Name or Term
.NET	Microsoft product, a language-neutral platform for enterprise and Web development
AMBER (alert)	America's Missing: Broadcast Emergency Response
API	Application Program Interface
BJA	Bureau of Justice Assistance
BS	Back-end Server
CADS	Center for Advanced Defense Studies
CPU	Central Processing Unit
CSPRI	Cyber Security Policy and Research Institute
DOJ	Department of Justice
DOM	Document Object Model
DTD	Document Type Definition
FS	Front-end Server
GJXDD	Global Justice XML Data Dictionary
GJXDM	Global Justice XML Data Model
GTRI	Georgia Tech Research Institute
GWU	George Washington University
HTML	Hypertext Markup Language
IIS	Internet Information Server
IJIS	Integrated Justice Information Systems Institute
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
JAXP	Java API for XML Processing
JWSDP	Java Web Services Developer Pack
JXDD	Justice XML Data Dictionary
Kb	Kilobyte
Mb	Megabyte
ms	Millisecond or 1/1000 second
NCIC	National Crime Information Center
OJP	Office of Justice Programs
OpenBSD	Open source variant of the Berkeley System Distribution UNIX
RAM	Random-Access Memory
RAP	Record of Arrest and Prosecution (Sheet)
SAIC	Science Applications International Corporation
SDK	Software Developer's Kit
XDR	XML-Data Reduced
XML	eXtensible Markup Language

Acronym	Name or Term
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformations

For more in-depth information about the IJIS Institute or a copy of this report, visit www.ijis.org