Optimized Trusted Information Sharing

Joseph M. D'Alessandro

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Tim Menzies, Ph.D., Chair
Katerina Goseva-Popstojanova, Ph.D.
Cynthia Tanner

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2010

Keywords: Information Sharing, Compliance Assurance, XPath Performance, Policy

Abstract

Optimized Trusted Information Sharing

Joseph M. D'Alessandro

As the digital world expands the building of trust and the retention of privacy in information sharing becomes paramount. A major impediment to information sharing is a lack of trust between the parties, based on security and privacy concerns, as well as information asymmetry. Several technological solutions have been proposed to solve this problem, including our's: a trusted enclave with a Continuous Compliance Assurance (CCA) mechanism. Of the work surrounding these proposed solutions, no attention has been directed toward studying the issues of performance surrounding processing of this nature. Studies have shown that ignoring the performance of a system can lead to ineffectiveness (i.e. disabling certain features), and can be severely detrimental to system adoption.

To ensure that our trusted enclave and CCA mechanism are viable solutions to the trusted information sharing problem, we have built a prototype CCA mechanism and a test bed. The test bed has allowed us to identify problem areas within our prototype. One such area is compliance verification, which utilizes the XPath language in order to test XML encoded information for compliance to regulatory and corporate policies. The compliance verification problem can be described as the answering of multiple queries over a single XML document. We proposed and tested multiple state-of-the-art algorithmic as well as system-based improvements to XPath evaluation, in order to better the overall performance of this aspect of our system. We integrated each of the improvements into our prototype mechanism and have observed the results. Our experiments have taught us much about the problem of compliance verification, and has led us in new directions as we continue to search for a solution.

Dedication

To my family, for their constant support.

Mom, Dad, Nicholas, and Lindsay.

I love you all!

Acknowledgments

Without the following people, this work would not have been possible.

Dr. Tim Menzies, for his constant enthusiasm and advice. He was always there to encourage me to look more deeply at the problem and to advise me to step away when I got to close. He taught me that there are many roads that research can take you down, and that you can learn something no matter which way you chose to go.

Mrs. Cynthia Tanner, for her wonderful support and encouragement. She provided me with a great opportunity to be part of her project as an undergraduate. Without that opportunity and her belief in my abilities, I may never have gone to graduate school. While working with her, she supplied me with fantastic advice and encouragement when I most needed it, helping to guide me through many of the difficult problems that I was presented with through out this work.

Dr. Katerina Goseva-Popstojanova, for her constant reminding of the nature of research and for all that she has done to feed my curiosities in computing. She taught me to see the beauty in an elegant solution, but to also look past the ugliness of a failed experiment. As she would come to teach me: no experiment is a failure if we learn something from it. Her insights and encouragement helped to shape me as both a student and a researcher.

To Ben Roberts, who must have grown tired of hearing my same explanation of why this or that happened, yet, was always there to listen. He always helped to bring me back to earth!

To Joe Falascino, who helped to put together our original test bed and rewrite the original prototype system. I wouldn't trade our marathon bug squashing sessions for anything in the world!

To Dr. Ed Jacobs, he taught me a new way to look at life and gave me the tools necessary to deal with myself.

To all the friends that I have made over the years. I am sure all of you, at some point, convinced me to step away from my work and enjoy life a little. Thank you, for teaching me one of life's most valuable lessons: don't take it too seriously!

Most of all, to my family: Mom, Dad, Nicholas, and Lindsay. When things weren't going well, you were there. When things were going exceptionally well, you were there. Throughout this process, I have occasionally needed to clear my mind, and each and every one of you were there to lend your ear. I can't put into words how much having you all by my side has meant!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Trust in Information Sharing

There are many situations in commercial supply chain operations [43], governmental operations [38], and heath care [2] [30] [44] [5], where the sharing of private information among several organizations could be mutually beneficial. Nevertheless, organizations are often reluctant to share even when all parties agree on how the shared data are to be used. This reluctance to share stems from concerns that the data recipients might not provide adequate security or enforce the agreed upon sharing policies.

In the commercial supply chain, information sharing can benefit companies, but it may also place them at risk if sensitive information is disclosed, thus trust plays a key role in this type of activity [43]. Organizations are often concerned about the disclosure of non-proprietary information about customers, vendors, or operations that could be used against the organization by competitors [26]. A study conducted by Albert Marcella et al [28], found that even with proper security assurances, trading partners involved in inter-organizational information sharing lack trust in each other. This is often due to the people involved in the transactions, making it clear that mechanisms must exist to allow for organizations to assess the "trustworthiness" of sharing partners (or potential

sharing partners). It has been demonstrated in research conducted by [43], that information sharing technology without a trust enabling element is insufficient for creating and maintaining a sharing relationship between business partners. According to this same research, businesses (involved in a supply chain) can surely benefit from the exchange of information.

According to a report issued by the National Security Council [10], the ability to share information amongst disjoint agencies is essential to the United States (US) National security strategy. To facilitate this need, the US government has begun to work toward establishing an environment in which agencies can share information [38] [41]. Essential to the success of this environment is trust between the agencies sharing information [10], and the trust of the public [10] [13].

Most of the issues pertaining to trust in the health care domain relate to the disclosure of sensitive information [9], leading many patients, physicians, and hospitals to hesitate in adopting the use of health IT systems and electronic medical records (EMR) as a standard means of data representation. From the patients perspective, Rohm and Milne [44], report that two concerns are central to the privacy concern: control of information being collected and the usage of that information. As for the physician and hospital, legal issues (lawsuits) from the disclosure of sensitive information are the chief concern [51]. According to McGraw et al [30] enhanced support for privacy will help in creating a higher level of trust in these systems, yet recent advances in health IT have failed to address this issue.

The central problem in the situations described above is that of information asymmetry. This refers to the inability of two information sharing parties to perfectly observe the actions of another party. A method of establishing trust in which this observational disparity can be reduced is to use a trusted enclave approach with a continuous compliance assurance mechanism (CCA). The trusted enclave provides a means of protecting highly sensitive and confidential information. The enclave would provide the physical and logical access controls to prevent unauthorized use. Included in the enclave would be data fusion applications (e.g. applications that identify data that matches across multiple sources, or collecting and combining data from multiple sources about an

entity of interest). These fusion applications execute within the enclave according to user specified policies, with the output subjected to monitoring for compliance to sharing rules and regulatory policy (both pre-specified). Testing for compliance is based on what leaves the enclave, not on data requests by individuals or applications. Information sharing policy enforcement, data fusion and analysis, and compliance assurance testing are done electronically according to pre-specified policies. An accessible audit log facilitates on-going ex post analysis of compliance. Central to the idea of the trusted enclave is the CCA mechanism, which provides an active process of continuously testing and verifying compliance with established policies (of regulatory nature). The CCA mechanism provides assurance through accountability by providing proof of the adherence to regulatory policies. The problem with this approach is that the verification of the adherence to the policies can be expensive and result in significant increase in the overall processing time. This performance cost is a major impediment to the adoption of continuous compliance assurance as a means of establishing and maintaining trust.

1.2 Trust and Engineering

In a world of digital information dissemination, trust can not simply be produced and sustained only through the use of a password and simple access control policies. The environment must be conducive to the protection of information being requested and its subsequent use, but we cannot simply eliminate the sharing of any sensitive information, as the sharing of this type of information can be beneficial in some case. In order to achieve this level of trust, systems must be built from more elaborate schemes, leading to more processor intensive activities. This causes users of such systems to turn off features [12] [22], and trust cannot be achieved if the features working to build the trust are not enabled. For this reason Hoffman et al [20], expanded the typical trust model to not only incorporate the psychological aspects of trust, but also the engineering requirements. The main claim of [20] is that the user's expectations about the system are important in building trust,

making the engineering of a system a necessary part of the trust model. In the words of [20]:

> "A thorough understanding of both the psychological and engineering aspects of trust is necessary to develop an appropriate trust model"

With this in mind, Hoffman et al expand the typical trust model to include: usability, reliability, and availability. Related to all three of these is the performance of the system. If a user of a trusted system cannot be sure that the overall performance of the system is not going to affect his or her business, then the system will surely fail.

Despite the warnings of Hoffman et al, there exists a surprising gap in the literature pertaining to the critical evaluation of the overall performance of systems proposed to establish and maintain trust. Much of the research conducted in the area of trusted information exchange has been directed toward the study of the psychological effects of trust. Of the systems discussed, an event monitoring mechanism (logging, policy enforcement, or both) seems to be a common aspect through which trust is ultimately built and maintained. This event monitoring mechanism exists to establish accountability (through proof of wrong doing) amongst the users within a sharing environment. If not implemented appropriately, these mechanism can significantly degrade the overall performance of the system, leading most operators to simply disable them. In order to prove that a trusted information sharing system may be viable in a real world situation, it is necessary to first study and understand the effects that these monitoring mechanisms have on the overall performance of such a system.

1.3 XML and Information Sharing

XML encoded data exchange is becoming more of a de facto standard in many domains , including government, business, and health care. Within the US government, XML is the representation language for the National Information Exchange Model (NIEM) [37]. NIEM will serve as the data model for the US government's Information Sharing Environment [41], and is currently used and

updated by a number of governmental organizations. Within the business domain, eXtensible Business Reporting Language (XBRL), an XML based model, is used to exchange information [21]. ACORD XML is used within the health care domain to exchange insurance related information [1].

XML is a data representation format that promotes readability and interoperability, but with these benefits comes a trade off: XML can degrade the performance of systems that rely on it [35]. In order to read and manipulate the XML encoded data, it must first be parsed. This step in the XML processing model can be very expensive (relative to performance) [24]. Several techniques exist to parse XML, some examples are: DOM [52], SAX [31], StAX [8], and VTD [42]. Even with the wide spread use of XML, little attention is paid to the comparison and study of the performance costs of these parsing techniques [24]. Each of these techniques are useful in different situations, and if not used properly can be costly to performance [24].

XML information alone, is not very useful without tools that can be used to explore the information. The standard language used to search XML documents is XPath [7]. XPath allows for simple specification of node sets within a document. A standard in memory representation of XML is a tree format, so naturally, XPath evaluation has been implemented through numerous recursive algorithms. A major problem that extends from evaluation of XPath in this recursive manner is the exponential evaluation time complexity (from the redundant evaluation of nodes on the evaluation path) [16]. While optimizations have been proposed to reduce the effects of this evaluation complexity, none of these consider anything beyond the processing of a single XPath query on a document.

Our CCA mechanism utilizes XPath as a means of verifying compliance to corporate or regulatory policies governing the data being exchanged. In this context, each of the policies (corporate or regulatory) is encoded as an XPath expression. The sole purpose of an XPath encoded policy is to determine if non-allowed fields exist within the information being exchanged (or verify that no exchanged information is in violation of any policy). For any given XML document, many XPath queries (policies) may exist. Thus, our problem is not that of efficiently evaluating a single

XPath on a document, but evaluating a (possibly large) set of XPath queries on a single document. This problem adds a non-trivial dimension to the exploration of appropriate strategies for XPath evaluation.

1.4 Statement of Thesis

Originally we set out to provide a recommendation for improving the evaluation process of XPath as a policy representation language. We evaluated multiple algorithmic improvements, and although we determined instances where some algorithms clearly operated more effectively than others, a common case arose: as the set of policies grew, the performance of all algorithms fell to a similar level. This begs the question: why are we concentrating on algorithmic improvements when the real problem is the policies? As a result of this discovery, we suggest multiple areas of exploration that work to reduce and merge policies into as few queries as possible. We will also suggest ways in which these policy reduction strategies can be incorporated with the algorithmic techniques to more effectively evaluate XPath in situations where multiple XPath queries must be executed.

1.5 Publications from this Thesis

Two publications have resulted from the research discussed in this thesis: [11] [33]. In [11], our unique problem of XML based processing (more specifically the processing of multiple of XPaths on a single document), is discussed, and possible solutions for overcoming these problems are suggested. In [33] the trusted enclave is explored more thoroughly, and our test bed is presented as means of studying our system's performance.

1.6 Structure of this Thesis

We begin this thesis by reviewing other work currently being conducted in the field of trusted information sharing, and how these studies relate to our work. Following this, we describe our trust model, our prototype mechanism, test bed, and our baseline results (gathered from the test bed). We next propose a set of solutions to overcome the problems we discovered through our test-bed. Following this we present results of the application of these proposed solutions and discuss the observations. We next discuss implications of these observations and present our future work. Finally, we summarize our observations and conclude the paper.

Chapter 2

Background and Related Work

In this chapter a review of related trust building technologies is provided. This discussion is meant to provide an understanding of the various approaches to building trust and why it is important for us to study the performance of certain features in these technologies. We first review a general architecture proposal to be incorporated into a governmental information sharing environment [27]. We next discuss a publish/subscribe based approach [50] [49], and follow that up with a discussion of a Hippocratic database approach [3] [22]. We finally present a brief overview of the Continuous Compliance Assurance approach and contrast this approach against the others.

2.1 Liu and Chetal

Liu and Chetal [27], propose an information sharing framework which makes use of an interest-based trust model as a solution to the problem of building trust among sharing partners. This work was motivated by the lack of communication and information sharing between government agencies. The authors claim that the lack of communication is related to the lack of trust between the agencies, stemming from two facts:

1. misunderstandings between government agencies
2. a lack of accountability

The misunderstandings between agencies (especially not understanding the intended use of the data), can lead to a the feeling of elevated risk which will decrease the likelihood (lower the level of trust) of sharing between the agencies. In the event of data misuse accusations, the lack of accountability makes it difficult to identify and verify the agency who misused the data (leading again to a lowering of the level of trust). Both of these combined make it difficult to create a culture of sharing among the government agencies.

Liu and Chetal, like most, view the problem of trust through the eyes of game theory, meaning that they are looking to persuade the actors involved in the interaction to both cooperate, by rewarding cooperation. In order for a model to assure that cooperation will occur and ultimately trust will be built, Liu and Chetal [27] describe four constraints that they feel must be met:

1. If neither agency feels slighted, mutual trust should increase
2. If one agency is slighted, the trust in the other agency should decrease
3. The interests of each of the agencies must be taken in to account (what does each agency value?)
4. There must be some way to assess the quality of the data being exchanged (check for validity and utility).

According to Liu and Chetal [27], if any of these constraints are not met, it is impossible for trust between two agencies to develop. As a result, they present a model which makes use of several different trust establishing strategies, in an effort to satisfy each of these constraints.

The model of trust described by [27] makes several assumptions. First, it is assumed that agencies involved in sharing have some sort of trusted central authority handling credentials. Next, they assume, that communication occurs in secure channels (free of denial-of-service attacks and eaves dropping). Finally, the infrastructure for inter-agency information sharing already exists ([27] is mearly interested in increasing the trust between the agencies).

The driving force of Liu and Chetal's trust framework (protocol) is what they describe as "information sharing policies." These policies fall into one of four categories:

Information release : an agency may explicitly specify situations in which information may be shared with other agencies. This type of policy is made up of two parts: a condition (a set of predicates separated by a conjunction or disjunction) and an action. Between zero (0) and four (4) predicates may be specified. Zero predicates implies that there exist no restriction limiting information exchange. If four predicates are specified, each predicate has a specific purpose. The first predicate checks for authorization (through an access control policy). the second and third predicates verify the validity of the information being shared, and its utility, respectively (it is assumed that a human being assigns ratings to these). The fourth predicate checks that the result of this action will increase the trust level between the sharing partners (using the trust level adjustment policy defined below). The action portion of the policy specifies the informational unit that may be release if the conditions in the first portion are fully satisfied.

Trust level adjustment : allows an agency to adjust its perceived quantitative trust level of other agencies. These policies are represented in a similar format as the information release policies described above. The main difference is that the action, in this case, does not grant the release of information, but increases the trust level one agency has for another.

Credential release : an agency may specify when and with whom sensitive credentials may be shared. Like the first two polices, this policy type consists of a condition (a set of predicates and an action. A credential release policy permits agency release its credentials to a sharing partner if the information is vitally important (negating the risk) or the trust level assigned to the other partner is high enough. The action aspect of this policy would perform the necessary steps to facilitate the release of the agencies credentials to its sharing partner (of course if the predicate conditions are completely satisfied).

Access control : allows an agency to limit which information units may be accessed by other outside agencies. These polices are of the same structure as the other three, a condition and an action. The action in this case grants access to the requesting agency if the condition is satisfied.

These policies are used during the negotiation phase of Liu and Chetal's information sharing process. The policies are used as the communication format between agencies. As the negotiation proceeds, the agencies adjust the level of trust in its partner, all while (hopefully) continuing to exchange information.

One of the goals of this work was to design a system to work with (or within) the Federal Enterprise Architecture (FEA) reference model [19]. This model describes the United States (US) government's plans for facilitating information sharing among federal agencies, and the citizens of the US. Among the constraints of the model is interoperability of agency and other private systems (all systems within federal environment should be able to communicate with each other). The FEA recommends information be shared in a predefined eXtensible Markup Language (XML) format (by making providing a common communication format, collaboration is more likely to occur [27]). Following this constraint, [27] developed a XML web services based implementation of their trust model.

2.2 Singh, Vargas, Bacon, and Moody

As health care moves closer to the full digitalization of patient records, dissemination of patient information will increase. With this spreading of information comes elevated concerns of privacy, and the desire to withhold information. Withholding information can cause problems in the treatment of patients and the study of diseases. At the heart of these concerns is a lack of patient trust within the health care providers to appropriately manage patient information. In order for patients to trust their health care providers, some level of assurance must be provided. Motivated by this

need for assurance, Singh et al describe a model architecture for facilitating sharing in a pervasive health care domain [50] [49].

Singh et al describe an architecture through which trusted information sharing can occur. Aspects of this architecture include: communication middleware, sharing policies, event monitoring (audit log), measures of "trustworthiness," and credential management.

The monitoring capabilities (through an audit) of this architecture are crucial to the maintanence of trust in the sharing environment. An audit log provides a means of accountability which will reduce the risk of information misuse. In a domain such as health care, an audit log is essential because of the sensitivity of the data and the large number of individuals interacting with the data.

Another aspect present in this architecture is a means of measuring (computing) trust. In Singh et al's environment, entities are assigned a trust value based on evidence gathered from the audit logs. Included in this evidence is: the accuracy/validity of the data being exchanged and data misuse. The accuracy/validity of the data is a qualitative measurement provided by feedback from other users and inference mechanisms provided by the architecture.

The architecture utilizes a publish/subscribe communication middleware (described in [50], therefore, all information is disseminated in the form of events. Events are posted (published) on topics, which are stored on nodes referred to as brokers. Topics typically host events of a certain type, for example, all prescriptions may be posted to a topic of pharameceutical requests. A user within the system can register interest in an event by subscribing to the topic on which that event type is posted. In this environment, all events (information) are encoded in XML.

In Singh et al's communication middleware, information sharing is limited by relevancy, in other words, an individual will only see what he or she needs to see (nothing more). This limited view of information sharing is controlled through sharing policies, and very closely resembles traditional access control. The sharing policies have one of two functions:

Transform Information : provides a means of altering an event (information) that meets some circumstances under which information can not be shared. The circumstances under which

the transformation will occur are encoded through a conditional clause. If the circumstances are met (the condition is fulfilled), a transformation function is fired and the event is altered to meet a previously agreed upon form, possibly removing fields and converting (or "bucketing") values. This function may also perform an event conversion (translate to another event type); this may include adding additional information, or possibly hiding sensitive information. There are two possible moments at which these transformation policies may be evaluated, on: publication and notification. On publication evaluation implies that anytime an event is published on a topic, it is first checked against the transformation policy (policies) tied to that topic. On notification transforms an event before it is forwarded to the subscribers of the topic.

Restrict Information : based on credentials of users (or other brokers) in the system, information can be withheld or fully denied through restriction policies. Restrictions are determined through imposed conditions, which can be viewed as a set of pre-requisites for information access. These conditions take precedence over all subscriber preferences. In order for information access to be granted, all of the imposed conditions must be met (all or none). In the event that all imposed conditions are met, data may be further restricted based on subscriber specific transformation policies. These policies are matched against the event type (after any notification transformations).

Although the policies are ultimately stored as rows in a database, they are specified in XML because XML is a simple and standard language through which policies can be sufficiently described. Storing the policies in a database allows the policies to reference functions loaded into the database. This can help the policies establish relationships between users, determine current conditional variables and system context, and access the event content. These policies are tied into the publish/subscribe architecture through what are referred to as "hook rules." The hook rules allow for policies to be checked at three different times, when a user: subscribes to a topic, publishes an

event on a topic, or is notified of an event on a topic. When an event is received by a broker, all applicable policies are looked up, and applied to the event.

Singh et al provide a model architecture for facilitating trusted information exchange. Though the model provides capabilities necessary for building and maintaining trust among its sharing partners, it neglects to address the fundamentals issues related to performance (reliability, availability, and usability) of such a system.

2.3 Hippocratic Database

The idea of the Hippocratic database (HDB) stems from the Hippocratic Oath, which has served as a guideline for ensuring appropriate handling of sensitive information among physicians [3]. The main purpose of the Hippocratic database is to enforce privacy policy (typically dictated through law, e.g. HIPAA) adherence, and monitor infractions [3]. Policies are managed through a method referred to as, Active Enforcement (AE).

AE is an "agnostic middleware solution" [2], comprised of three different aspects: policy creation interface, a preference negotiation mechanism, and a policy enforcement engine [22]. The policy creation interface allows for the definition of corporate or regulatory policies pertaining to information to be stored in the database [22]. An example of such a policy would be: The Health Insurance Portability and Accountability Act (HIPAA) [36]. The preference negotiation mechanism allows an individual whose information will be stored within the database to specify his or her preferences for the disclosure of his or her information [22]. During the negotiation phase the individual will be presented with a comparison of his or her preferences and applicable corporate and regulatory policies [22]. The individual will be notified of conflicts between the policies and provided an opportunity to withdraw his or her account (if he or she does not agree with the disclosure policies) [22]. The final aspect of the HDB is the policy enforcement engine, which is active during the information retrieval [22]. This engine will modify database queries to adhere to

individual preference, corporate and regulatory policies [22].

Along with AE, the HDB employs an audit mechanism to verify (ex post) whether or not compliance to privacy policies has been maintained [22]. The audit mechanism maintains a log of all information: updates, insertions, deletions, accesses, and retrievals within the database, and necessary contextual information (user, query time, query purpose, etc.) [22]. An administrator can search the log by fashioning a query that attempts to retrieve sensitive information; this query's results are then compared against queries currently held in the log, if any query's results contain a tuple that is in the administrator's query it is deemed "suspicious" [22].

2.4 The Enclave and Continuous Compliance Assurance (CCA)

As a possible solution to the trust problem, we have explored the use of a trusted enclave environment with an embedded continuous compliance assurance (CCA) mechanism [32]. The trusted enclave exists as an environment in which sensitive information can be stored, providing assurance that authorized access to the information will be maintained both physically and logically. Figure 2.1 depicts our model of the trusted enclave. We can see from the figure that the enclave encompasses many applications commonly associated with information sharing, including: analysis and fusion applications, data validation, data access, and audit logging. The analysis and fusion applications carry out transformations of data stored in the enclave. The output of these applications is subject to monitoring for compliance to user specified sharing rules, and domain specific regulatory rules. Only information leaving the enclave is subject to a compliance check (data requests are not). Compliance checks are performed electronically through the application of machine readable rules specified by the providers of the data. To assure that accountability exists in the enclave, all activity is logged. This audit log exists to provide ex post analysis of compliance.

Embedded within the enclave is the CCA mechanism, outlined by the dotted lines in Figure 2.1. This mechanism is responsible for verifying compliance to all policies applicable to a certain

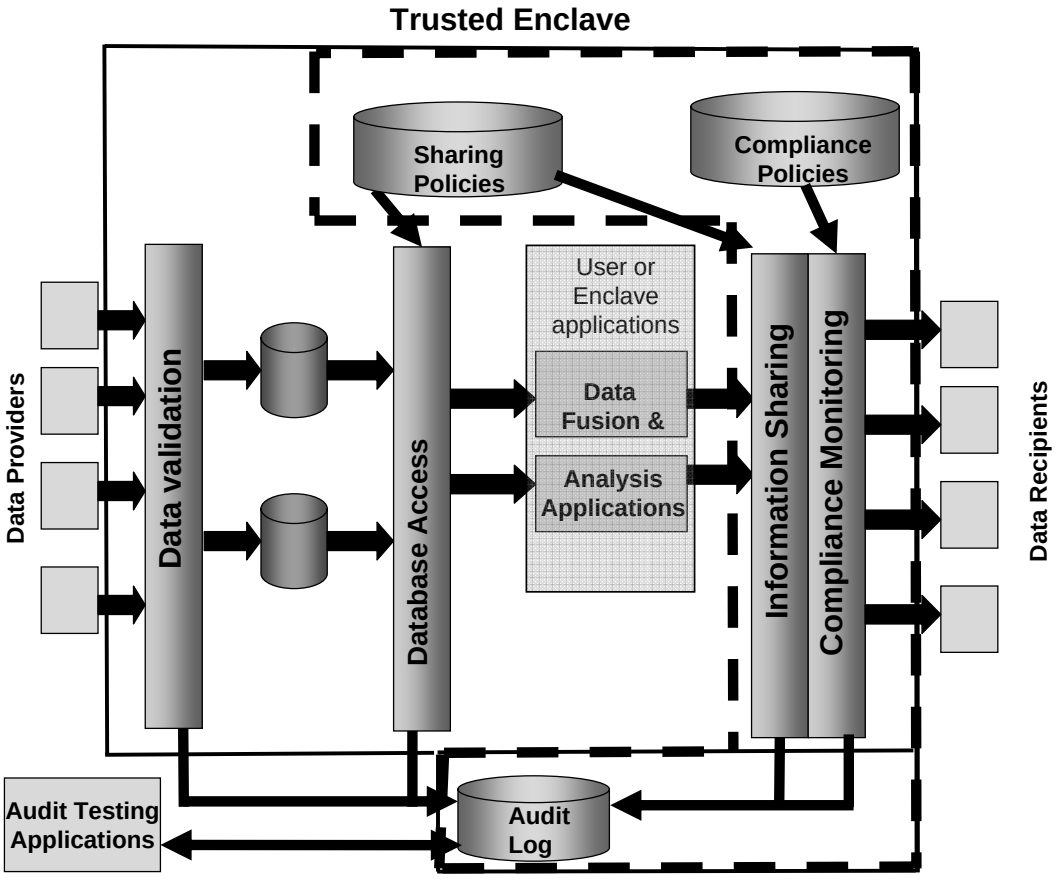

Figure 2.1: View of the Design of the Trusted Enclave

message. The CCA mechanism is designed to accept a pre-defined XML message (in Figure 2.1 the output comes from the fusion and analysis applications), this choice was made for reasons of interoperability. In our enclave environment, there exists four types of policies:

Validation : These policies are designed to verify that a message is of a correct structure/format (based on the type of message). These policies are expressed through the XML Schema Definition (XSD) language.

Sharing : These policies act as an agreement between the provider of the data and the recipient(s).

The policies "strip away" any data that is not allowed to be shared. These policies are expressed as eXstensible Stylesheet Language Transformations (XSLT).

Compliance : These policies represent the regulatory or corporate laws that govern the data being shared. Applicable policies are selected based on the content of the message. This policy selection can be determined in one of two ways: statically (through the message type) or dynamically (through the analysis of individual data elements in the message). The only concern of policies of this type is the existence of offending data. Therefore, these policies are appropriately encoded as XPath expressions.

Resolving : These policies exist as a measure of dealing with message failures in the system. They are designed to allow the user to express a set of steps to be carried out in the event of some message failure event occurring.

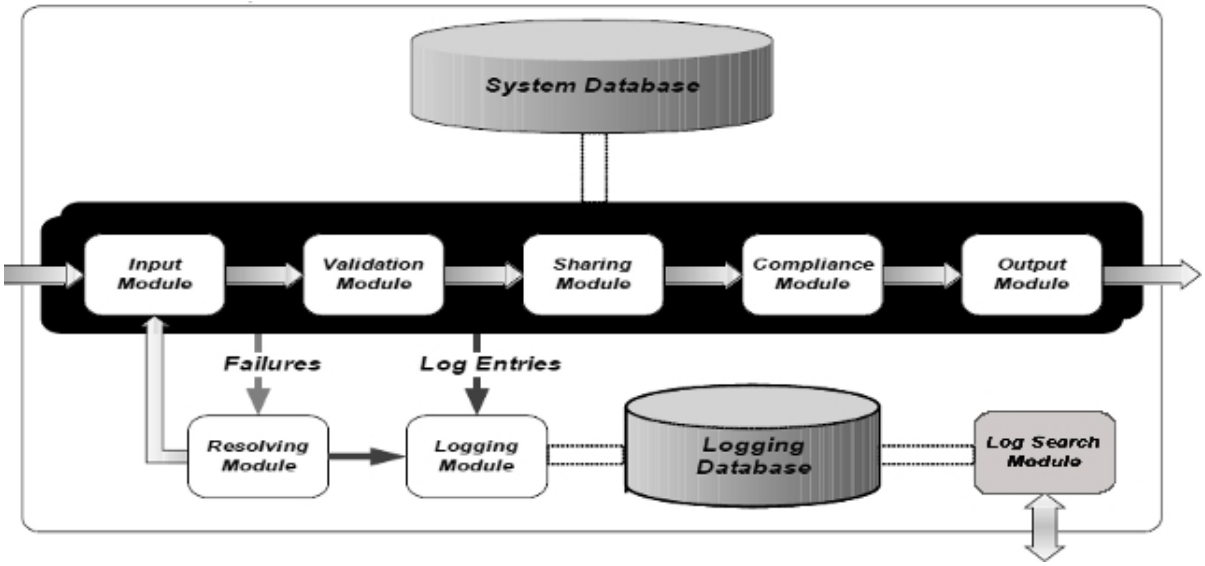

Figure 2.2: View of the Design of our Continuous Compliance Mechanism

The application of a user defined sharing policy results in message containing only the data elements that were allowed to be shared. The resulting data is checked against both corporate and regulatory policies; if the data is non-compliant to any of these policies, delivery is halted and the

offense is logged. Figure 2.2 displays a more finely grained view of the CCA mechanism. The mechanism can be viewed as a set of modules:

Input : Identifies the message type, each of the users (providing and receiving), and verifies the relationship between the users.

Validation : Verifies the XML message is of the proper format for its message type. This helps to remove the possibility of errant data being embedded into the message. It also ensures that the application of both the sharing and compliance policies will be possible.

Sharing : Applies intra- or inter-organizational policies pertaining to the data currently being exchanged.

Compliance : Verifies compliance to all international, national, and/or company regulatory policies.

Output : Forwards the data to its recipient(s).

Resolving : If at any point a message fails, it is immediately sent to resolving, where some user defined action may take place. This may be, for example, a notification sent to the receiving user.

Logging : Adds log entries to the enclave wide immutable audit log. In the CCA mechanism, messages are logged both before entering and after exiting a module.

2.4.1 What makes CCA different

The development of trusted information sharing will often have many common aspects among multiple different strategies. Among these commonalities are those found in typical access control mechanisms, which include:

Dictation of Sharing Requirements : the expression of desires in the sharing of information; limiting what can be retrieved by other users in the sharing environment.

Accountability to Sharing Requirements : proving that all users are in adherence to the sharing requirements of the environment.

Without these two aspects, trust cannot be built and ultimately maintained, but these two pieces alone are not enough. Unfortunately, Liu and Chetal [27] and Singh, Vargas, Bacon, and Moody [50] [49], concentrate only on these ideas. They both rely solely on sharing requirements and an ex post verification of the adherence. This means that violations can occur and will simply be caught afterward. This can lead to a lowering of confidence among the users of the system, and eventually abandonment. When it comes down to it, this provides no real protection beyond the proof that at sometime in the past someone had violated the sharing requirements of the system. The desire to violate a sharing rule is diminished by the accountability, but there is no verification that the sharing rules themselves are not in violation of any regulatory bodies. This lack of verification can, too, reduce the user confidence in the system, and eventually reduce user willingness to share because of fear of regulatory violations. CCA provides both of the aspects of these systems, but extends the verficiation process to also cover regulatory adherence. This extra verficiation step ensures that the user agreed upon sharing policies are not violating any of the numerous government or corporate regulatory bases. This also provides another step at which the sharing of information can be stopped due to violations. In CCA, any message that does not pass the verification to all applicable regulatory bases is stopped and handled appropriately. This means that violations are not simply caught after the violation, but during. The logging in the case of CCA verifies that the compliance verification is appropriately capturing violations, to provide further assurance to the users in the environment that no unlawful or inappropriate information sharing is occurring.

As for the Hippocratic Database (HDB), it differs from CCA in that it operates on the request for data, and not the data leaving the enclave. Allow me to elaborate, the HDB will analyze the

query being executed on some back end database and modify its structure to ensure that disclosure of sensitive information does not occur. CCA is part of the larger idea of a Trusted Enclave, and allows for more flexibility in the access of information, such as transformations through fusion applications. Though, initially, the requested data by the fusion application may violate some policy, the result of the fusion application may not (some sort of aggregation), thus it is absolutely necessary for CCA to verify data leaving the enclave.

Chapter 3

Prototype and Test Bed

In this chapter we present our CCA test bed. We will discuss the components that are included in the test bed and describe the layout and configuration of the machines used. Also discussed will be the implementation of the CCA mechanism. We end the chapter with a presentation of the baseline results.

3.1 CCA Test Bed

To study the effects that changes would have on our system, we have built a test bed (pictured in Figure 3.1) which consists of a(an):

- prototype CCA mechanism
- enclave database (small scale model)
- workload generator
- set of analysis tools
- logging facility

Each of these components is implemented in the Java programming language. Communication between the components is facilitated through the use of the Java Messaging Service (JMS) and a

Figure 3.1: The CCA Laboratory Test Bed

set of ActiveMQ message brokers. To eliminate the possibility of network inconsistencies affecting our experiments, all of the machines hosting these components are connected through a 10/100 switch. The topology of this test bed is displayed in Figure 3.2.

Table 3.1: Machine Specifications(machine numbers refer to those found in Figure 3.2

Machine Number	CPU	HD RPM	Memory	O.S.
1	Core2Duo (2.53 GHz)	7200	2 GB DDR 800	Linux 2.6.28-11
2	P4 HT (3.00 GHz)	10000	1 GB DDR 400	Linux 2.6.28-11
3	P4 HT (3.00 GHz)	7200	1 GB DDR 400	Linux 2.6.28-11
4	P4 HT (3.00 GHz)	7200	1 GB DDR 400	Linux 2.6.28-11

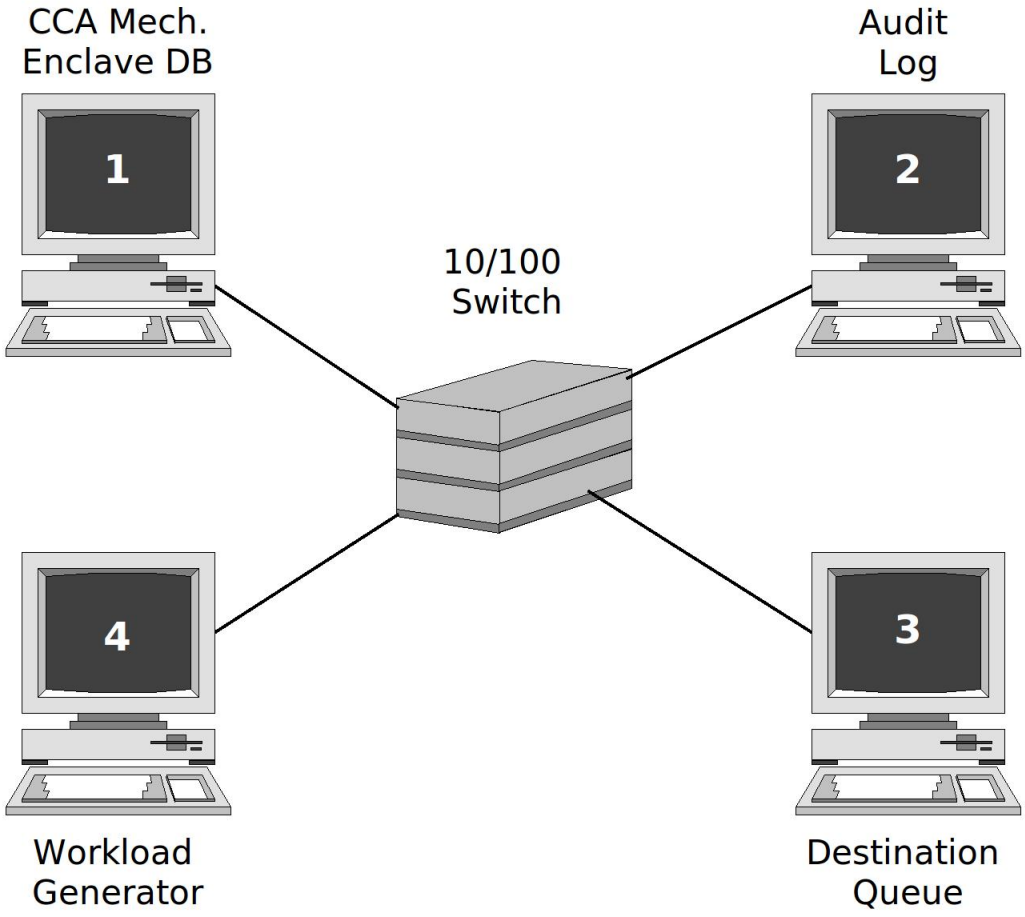

Figure 3.2: Topology of CCA test bed

Decoupling of the test bed components in this manner has allowed us to experiment with different configurations and easily isolate components to study them individually. This could be accomplished because the components would always communicate in the same way (through the JMS interface). Table 3.1 describes the configuration used to collect results for the experiments discussed in this thesis (unless otherwise stated).

3.1.1 CCA Mechanism

This mechanism is written in the Java programming language and can be decomposed into the following modules: Input, Validation, Sharing, Compliance, Output, Logging, and Resolving.

Referring back to Figure 2.2, the core modules must be executed in the order depicted (from left to right). For this reason, the system was designed in a monolithic style dictating that each of the modules operate in a single thread of execution (and implying that no two processes may operate in parallel). Messages are fed to the CCA mechanism through a queue residing on an ActiveMQ message broker, referred to as the input queue. A brief overview of each of the modules follows.

Input Module

Input is the module that first receives a message after it is pulled from the input queue. Utilizing the Java Database Connection (JDBC) API, input connects to the Enclave database and retrieves user credential information (verifying that both the providing and receiving users are valid users of the system and that a relationship exists between the two). Using this credential information, the input module then queries the enclave database for both the validation policy and user sharing policy defined for this relationship. These are both stored for later use by the validation and sharing modules respectively. Using the providing user, the receiving user, and the message type, a set of applicable compliance policies are also gathered and stored for later use. If any of the queries to the enclave database fail to return a result processing is stopped, an exception is raised, and the message is forwarded to the resolving module.

Validation Module

After input has completed its processing, the message is next processed by the validation module. Using the validation policy (encoded as an XSD) retrieved by the input module, the validation module will verify that the message fits a predefined agreed upon form. This is achieved through the use of the Java API for XML Processing (JAXP). JAXP provides a state of the art in memory schema validation tool, which is used to apply the XSD to the XML message. This validation is necessary to ensure that both the sharing and compliance policies (evaluated in the next two modules) can be applied correctly. If a message is deemed to not be valid according to the XSD,

an exception is raised and the message is forwarded to the resolving module.

Sharing Module

Once the message is validated, the next module to operate on it is the sharing module. This module ensures that the message data adheres to the information sharing agreement between the providing and receiving users. Using the sharing policy (XSLT) retrieved in the input module, this module will guarantee that only sharable (according to the agreement) data items remain in the message. This is achieved through the use of the Java API for XML Processing (JAXP)'s XSLT support libraries. Essentially, this process "copies" the data elements within the message that are allowed to be shared, which results in a "new" (new in the sense that it does not contain the non-sharable data) message. This "new" message is then returned.

Compliance Module

After the application of the sharing policies, the message returned from the sharing module is next processed by the compliance module. Using the set of compliance policies gathered at input, the compliance module will verify that the sharing agreement between the providing and receiving users is not in violation of any of the corporate or regulatory policies governing the data being exchanged. The policies in this case are encoded in the XPath language. XPaths are expressions that always return a set (including the empty set). Our compliance policies are written as a description of the non-compliant data that could possibly be in the document. Each policy is designed to return either a $true$ or $false$ answer. A policy evaluates to $false$ if an XPath expression returns a non-empty set, meaning, that data was detected (within the document) which matched the policy description (indicating that the message is in violation of the policy). A policy evaluates to $true$ if the XPath expression returns the empty set, meaning that no data elements were found to be in violation of the policy. Due to the fact that all policies return a boolean result, compliance to a set of policies can be determined by the conjunction of the policy evaluations (i.e. if one policy

is found to be non-compliant the conjunction of that policy and any other policy will always be $false$ and the message would deemed to be non-compliant). Evaluation of the XPath expressions is carried out using the Saxon 8.8 XPath library. This library makes use of a DOM-based XML tree representation. If a message is deemed to be non-compliant, an exception is raised and the message is sent to the resolving module.

Output Module

A message that is found compliant to all applicable policies is next handled by the output module. The output module uses the user credentials gathered at input to query the enclave database for a destination queue to which the message should be sent. Once the destination is determined, the output module forwards the message.

Resolving Module

If at any point during the process, a message fails, the message is immediately forwarded to a special module: Resolving. Failures include: no defined user relationship (detected at Input), invalid message form (detected at Validation), or non-compliance to a corporate or regulatory policy (detected at Compliance). Currently, Resolving only acts as a forwarding mechanism for failed messages.

3.1.2 Enclave Representation

Another aspect of our test bed is the mimicking of the trusted enclave environment described above, specifically the relationships between sharing partners. To achieve this we created a model enclave database containing prospective users of the system, validation policies, user sharing policies, compliance policies, and resolving policies. An entity relationship diagram describing the relationships of data elements in our model enclave database is displayed in Figure 3.3. It is important that relationships are properly defined and user related policies are assigned appropriately.

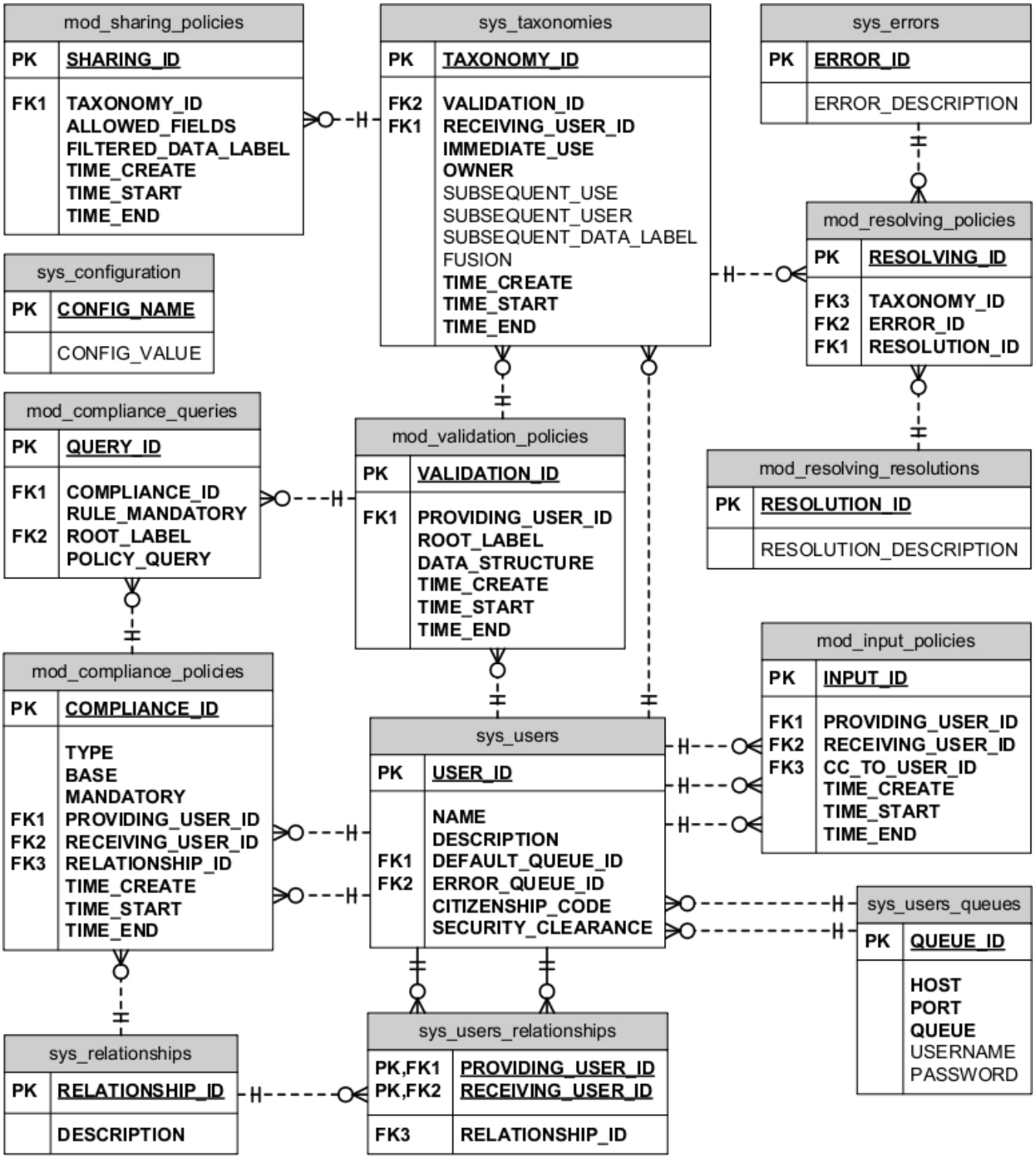

Figure 3.3: Enclave System Database Entity Relationship Diagram

To ensure that this is the case, we developed the following simple procedure for creating the model enclave database:

1. Load users and credentials into the database
2. From this set of users, randomly select two users and register this relationship within the database
3. From the set of available message types in the current scenario, randomly select a message type and assign it to this relationship. This also implies that a message validation policy (XSD) will need to be assigned to this relationship.
4. Each of the message types will have a set of predefined sharing policies. Of these predefined policies, some are designed to remove any non-compliant data others are designed to not remove non-compliant data. This allows us to test that compliance check is working appropriately. From these policies, randomly select one and assign that policy (XSLT) to the user relationship.

Following all these steps ensures that any message passed between users that have a defined relationship will not fail at input. Compliance policies are not directly linked to any particular user, so the procedure for creating and loading these policies follows a different procedure. In an effort to make our simulations as real as possible, we collected a set of real regulatory policies (both U.S. and International) that apply to the data contained in our messages. To complement these regulatory policies, we devised a set of corporate policies. These policies are referred to as our **Real Policy** set. Having realistic policies like the ones listed in Figures 3.4 and 3.5 serve two purposes in testing:

1. It verified that our compliance mechanism was working correctly
2. It allowed us to observe the systems performance in evaluating a set of real policies

```
not(boolean(//RAAR/CrewDiscrepancies/CrewDiscrepancy/CrewPersons/CrewPerson
              [CitizenshipCode = "US"        and
               (boolean(SID)                 or
                boolean(DateOfBirth)         or
                boolean(PlaceOfBirth)        or
                boolean(Height)              or
                boolean(Weight)              or
                boolean(HairColor)           or
                boolean(EyeColor)            or
                boolean(DistinguishingMarks) or
                boolean(Sex)                 or
                boolean(VesselName)          or
                boolean(Status)              or
                boolean(CurrentLocation))]))
```

Figure 3.4: U.S. Privacy Act of 1974

```
not(boolean(//ShipManifest/CargoManifest/NonContainerItems/NonContainerItem[
              (Consignee = "DoD" or
               Owner = "DoD"     or
               MarksAndNumbers/MarkAndNumber/Mark = "DoD Restricted") and
               boolean(Weight)]))
```

Figure 3.5: Non-disclosure of Department of Defense contracted cargo

Although, this was a great start to our policy definition, we felt that it was necessary to ramp up our efforts so that we could better study the performance of our system. To do this we added a set of **Synthetic** policies that would be evaluated but whose outcome would not affect the compliance of the message (i.e. only the real world policies can deem a message compliant or not). This set of policies consists of XPaths representing the enumeration of all the distinct paths from the root of an XML document to all of its leaves. To ensure that the evaluation of these XPath statements is always true (i.e. always compliant), we added a disjunction of the boolean evaluation of the path with a simple boolean evaluation of the descendent-or-self axis (which for any non-empty document will always return true).

3.1.3 Workload

The purpose of the workload generator was to provide our CCA mechanism with a continuous stream of messages to process. These messages are encoded in XML. In our test bed, scenarios were defined by the possible messages that users could send. The scenario used for these experiments was devised over the months of September and October of 2006 through conversations with representatives from our funding source, VIACK corporation. This scenario was meant to mimic a situation in which time sensitive information is to be shared. The information in this case is related to the monitoring of suspicious activity occurring in the international freighting industry. To model communication in this scenario we defined four basic message types:

Risk Assessment and Analysis Report (RAAR): One of the aspects of our proposed trusted enclave environment is the fusion applications. The RAAR represents an example of a message that is generated from one of these fusion applications. The data in this message includes anomalies related to cargo items and crew members. For example, a crew member may be listed as active on two boats, which can never happen.

Ship Manifest: Another use of our system would be the exchange of supply chain information,

which is modelled in this scenario through the Ship Manifest. This message contains a list of crew members and a listing of the cargo currently on board.

Alert: Another fusion mechanism might be alerting users of the enclave of new information. This messages represents a commonly exchange simple message.

Request: The requesting of information from the enclave is another activity that would occur frequently in day to day operations. This activity is modelled through the request.

	Number of Elements			Size (kB)		
Component	Mean	Standard Dev.	Median	Mean	Standard Dev.	Median
RAAR	2918	658	2267	144.35	115.37	112.07
Ship Manifest	920	651	694	40.26	31.72	30.43
Alert	3	0	3	0.559	0.0	0.553
Request	3	0	3	0.564	0.0	0.567

Table 3.2: Message type descriptions

XSDs representing each of these messages can be found at: `http://thejoeshow.net/cca/xsd`. Table 3.2 displays statistics related to the size and structure of each of the message types, gathered from the generation of 20,000 messages. As we can see, the **Requests** and **Alerts** are the smallest of the messages, and they do not fluctuate much in size. This decision was made to model real life situations where we expect to see a larger number of smaller messages (having two equally small messages increases the likelihood that we will select these types of messages more often). These smaller messages indicate quick message banter between users of the system and the enclave. We also see that the **RAAR** and **ShipManifests** are typically much larger messages and fluctuate more widely than do the **Requests** and **Alerts**. These messages were designed to represent the reactions to the **Requests** and **Alerts**. These messages are intended to model the requestable content of the enclave.

Message generation must follow a certain procedure to ensure that messages intended to be successfully shared make it through the system, and those designed to fail, fail at the right time (mainly input or compliance). This means that we must select appropriate providing and receiving users for the situations we wish to create. For example, we select only users who have a predefined relationship for those messages that we wish to pass successfully through the input, validation, and sharing modules. In order to sufficiently model cases of non-compliance in our scenario, we had to incorporate sharing policies that were known to be in violation of at least one compliance policy. By including these, we were able to guarantee that we tested whether or not the system appropriately failed non-compliant messages.

We use our generator to create a set of messages, and save them on the hard drive. A tool referred to as the workload runner is then used to actually "run" the workload. The workload runner grabs a message from the hard drive, creates a JMS message (stamping the appropriate properties), and sends that message to the CCA input queue, where it waits to be processed by the CCA mechanism. This workload runner can send messages at a maximum rate of 80 messages per second, but can be slowed to more appropriate rates. For our purposes, we send at a rate of approximately 4 messages per second. This is done so that we do not overload the input queue and cause the ActiveMQ queue manager to slow down and affect the performance of the CCA mechanism.

3.1.4 Logging Facility

An important aspect of the CCA Mechanism is the Logging module which maintains an up-to-date log of the activity occurring throughout the processing of a message. This is achieved by sending a copy of the message to the Logging module before and after each module operates on it. This module operates in parallel to the CCA mechanism (residing on a separate machine), and listens on a queue hosted on an ActiveMQ queue manager. Once a message is received by the logging module, it is broken down and characteristics of these messages are then selected and stored into

an audit log. The entity relationship diagram of this audit log is depicted in Figure 3.6. Some of the characteristics logged for a message would include:

- providing and receiving users of the message
- data specific constraints (quality, usage, owner, etc.)
- error details
- the time of the message failure
- module in which the failure occurred

Capturing and storing this information allows for both external independent ex post auditing and examination of data access and exchange. This facility provides our system with another form of accountability (necessary for building trust).

3.1.5 Gathering Statistics

Our test bed is designed to capture throughput performance statistics for our CCA mechanism. To make this possible without negatively affecting the performance of our system, we store our measurements as JMS properties of the message. The cost of recording a measurement is equivalent to assigning a key value pair to a hash table (constant time). These properties are only visible through the JMS interface and in no way affect the form of the XML document being processed. Leveraging the use of these properties, we are able to record timestamps of significant events in the processing cycle. Some events that we record are:

- Timestamps for when a message enters the CCA mechanism and exits the CCA mechanism. The difference in this case indicates the the time it takes a message to be fully processed by the CCA mechanism. These timestamps are used to calculate the **system throughput time**

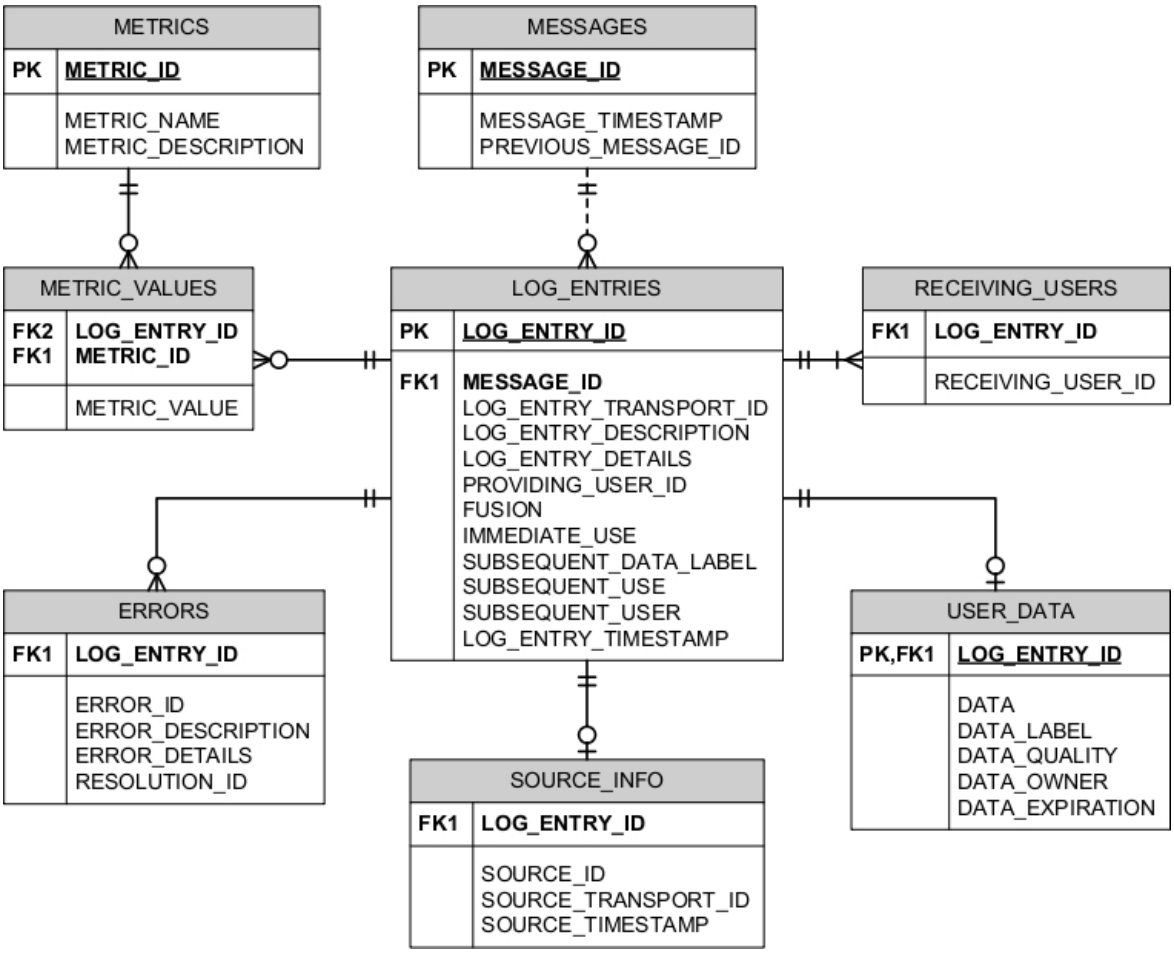

Figure 3.6: Logging Database Entity Relationship Diagram

- Timestamps for a message entering a module and exiting a module. The difference of course giving us an idea of how long a message spends in each of the modules. Giving us a finer granularity to work from. These timestamps are used in our calculation of the individual **module throughput times**.

- The last module to operate on the message. This provides a necessary means of determining where in our processing cycle messages are failing.

An added benefit of collecting statistics in this way is that if the system structure changes in any way, this capture mechanism does not need to. This means that we could explore the performance of our system in the same way if our modules were spread over multiple machines instead of being

contained in the monolithic fashion that we currently utilize.

Messages that have passed through the CCA mechanism are collected onto a final queue. Our test bed includes a tool for automatically retrieving and assessing the messages residing on this queue. The assessment includes the calculation of average throughput time, average messages per second, average bytes per second, and average time spent in each of our modules, among other statistics.

3.2 Baseline Results

Using our test bed, we began to observe the behavior of our prototype CCA mechanism. Our approach was to observe the system on a large scale (overall system throughput) and on a smaller scale (each of the modules individually) to determine where most of the processing time was being spent. Figure 3.7 displays the execution time per module of our initial system prototype. Dis-

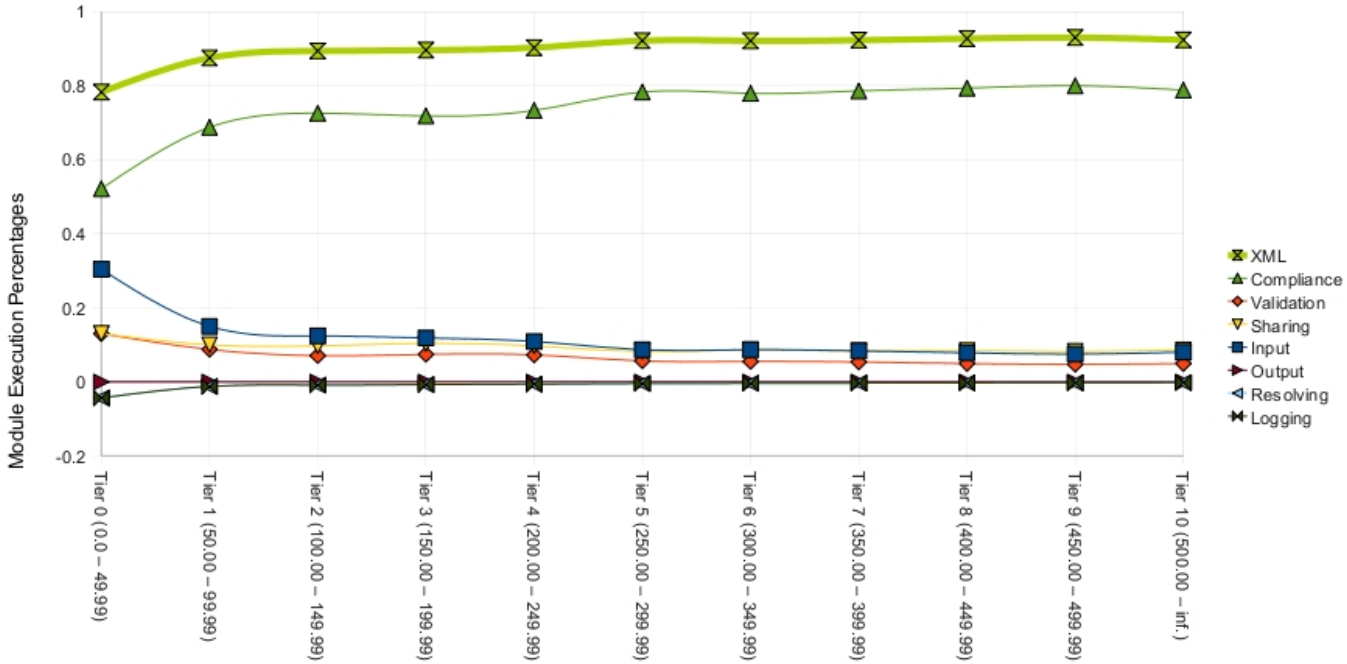

Figure 3.7: Average Execution Time per Module (as a percent of the total time), broken down by message size

cussed in the last chapter, Validation, Sharing, and Compliance all perform some form of XML based processing. Figure 3.7 displays the total processing time spent in these three modules (labelled: "XML"). For larger messages this time approaches nearly 95% of the total time. Looking specifically at the individual modules, we can see that in this situation, the compliance module accounts for nearly all of this time. For smaller messages, compliance accounts for approximately 50-60% of the total processing time. As the messages get larger, compliance approaches (on average) 80% of the total processing time per message. Throughput results for our baseline CCA mechanism are displayed in Figure 3.8 and Table 3.4. We can see in Figure 3.8 a rather steep drop from the smallest message tier to the next (from 33.3 to 9.43), and from that point on, throughput performance falls steadily to nearly 2 message per second for the larger tiers. This drop seems to correspond to the dominant processing time of the compliance module, indicating that if we were to reduce the processing time of the compliance module, that should increase the overall system throughput.

Method	Class	Type	Time (% of total execution time)
compress	CompressedWhitespace	XML Processing	36.16%
charAt	CharSlice	XML Processing	29.62%
flush	ReceivingContentHandler	XML Processing	2.24%
characters	XMLIndenter	XML Processing	2.04%
characters	TinyBuilder	XML Processing	1.75%
startElement	ReceivingContentHandler	XML Processing	1.74%
sendSAXSource	Sender	XML Processing	1.54%
length	CharSlice	XML Processing	1.53%
addNode	TinyTree	XML Processing	1.06%

Table 3.3: Top 10 Methods by Execution Time

To further inspect the problem, we ran a profiler on our on our prototype CCA mechanism. For this we used Eclipse's Test and Performance Tools Platform (TPTP). With this tool, we were able to observe execution time at the method level, class level, and package level. The top 10 most

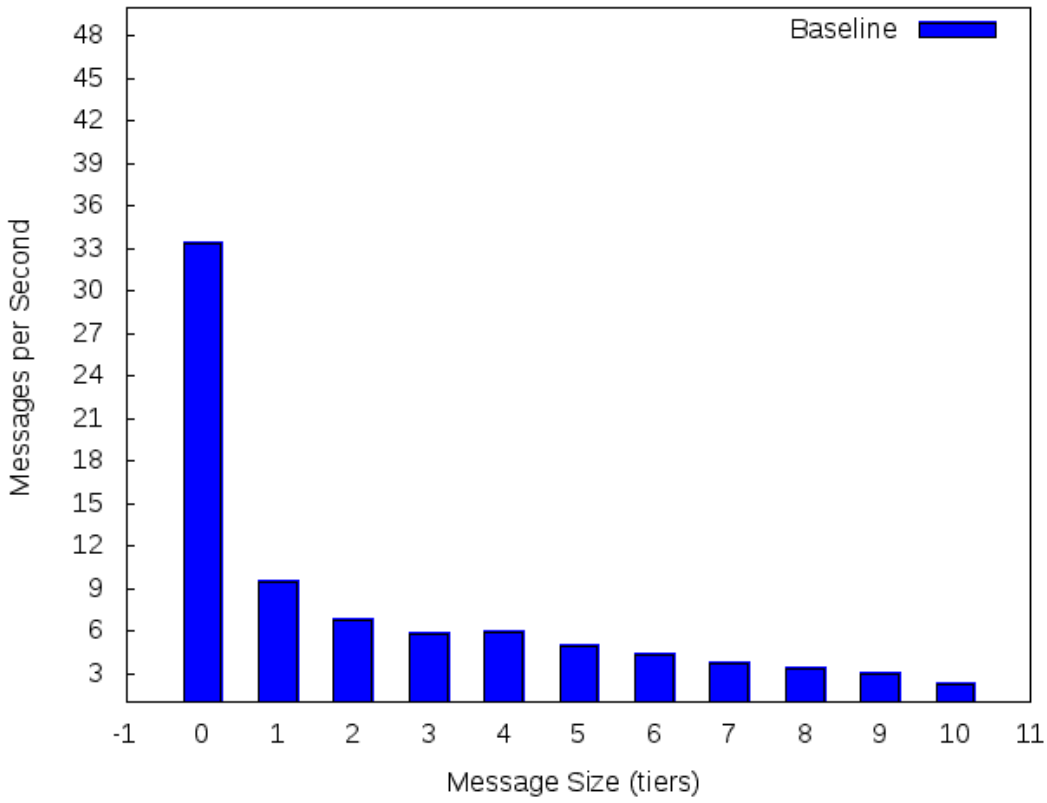

Figure 3.8: Message Throughputs for Baseline CCA Mechanism, broken down by message size

costly methods are displayed in Table 3.3. These methods all are related to XML processing. The top method (*compress*) is only called during the construction of the in memory XML data model used by the XPath processor (this is called before the evaluation of each XPath query). The next method (*charAt*) is called during the traversal of the data model occurring during query execution. Combined, the methods in Table 3.3 account for about 77.68% of the total execution time of the system. Finding a way to reduce the effect of these methods, should have a great impact on overall performance.

Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	33.33	9.43	6.80	5.78	5.88	4.93	4.26	3.66	3.37	2.99	2.18
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 3.4: Throughput Results per Message Size (Baseline)

CCA is an XML based process requiring several steps involving XML based processing. Modules related to this type of processing encompass approximately 80% of the total processing time of the CCA mechanism. This fact is further validated by the profiling of the system where it is clear that most (77.10%) of the systems processing time is related to the processing the XML. Each activity performed by these modules is a required step in checking for regulatory compliance, and thus the feasibility of in-stream audit mechanism (like ours) relies on the time cost reduction of the XML based processing. From Figure 3.7 we can see that the compliance module is the most costly (as far as processing time) of all modules and accounts for 50%-75% of the XML related processing time. Efforts to better system throughput should, therefore, be directed toward the improvement of performance of operations within the compliance module.

From the literature review [35] [24] and our own observations, we conclude that the performance problems extend from our XPath evaluation model. Currently, we parse XML into a DOM tree, store it in memory, and execute statements over this tree. Due to the recursive nature of XPath (in the standard evaluation methods), the current models of DOM based XPath evaluation may require redundant evaluations (per path). Also related to this is the cost of DOM based XML parsing. According to Lam et al [35], DOM based parsing is the most expensive in terms of both memory and CPU usage. Being that we currently employ a DOM parsing technique, we have seen this expense in action. Thus, it would be in our best interest to explore alternative patterns of parsing. From our initial experiments, we have seen that the compliance module utilizes the largest chunk of processing time. This module is solely dedicated to executing XPath queries to determine compliance. We have seen that as the messages grow larger the execution time grows to about 80% of the total. If we are able to possibly reduce the search space for the policies operating on these larger documents, we can maybe observe better performance in these situations. Thus, our research has directed us toward non-standard evaluation methods of XPath. We will explore to improvement approaches: pre-process (reduce the search space) and inter-process (determine an

algorithmic or system based improvement).

Chapter 4

Policy Encoding

One of the most important aspect of our CCA mechanism is the encoding of policy bases in XPath. In this chapter we will discuss a few key policy bases, and then, through examples, outline the procedure for encoding these bases in XPath.

4.1 U.S. Privacy Act of 1974

The U.S. Privacy Act of 1974 provides protections against disclosures of personally identifiable information (PII) of U.S. citizens [17]. Some of the data items that are considered PII are: social security number, name, address, phone number, etc.

4.1.1 U.S. Privacy Act Encoding

Assume that we are interested in verifying that an instance of a RAAR message is in compliance with the U.S. Privacy Act of 1974. Figure 4.1 displays the XPath encoding of this policy. The U.S. Privacy Act is only concerned with U.S. citizens, therefore this policy only checks for the existence of PII for U.S. citizens. Compliance policies are encoded in such a way that they look for elements that are not permitted, reporting a boolean value of $false$, if any offending data is discovered. In this policy the offending data is listed as a set of disjunctions (i.e. SID (or Social

```
not(boolean(//RAAR/CrewDiscrepancies/CrewDiscrepancy/CrewPersons/CrewPerson
              [CitizenshipCode = "US"        and
               (boolean(SID)                 or
                boolean(DateOfBirth)         or
                boolean(PlaceOfBirth)        or
                boolean(Height)              or
                boolean(Weight)              or
                boolean(HairColor)           or
                boolean(EyeColor)            or
                boolean(DistinguishingMarks) or
                boolean(Sex)                 or
                boolean(VesselName)          or
                boolean(Status)              or
                boolean(CurrentLocation))]))
```

Figure 4.1: The encoding of the U.S. Privacy Act of 1974 in XPath

Security Number). The *boolean*() function in XPath returns *true* if expression it wraps returns a non-empty set. and *false* otherwise. So the set of disjunctions in the policy below returns *false* only if none of the data items are found.

4.2 Defense Federal Acquisition Regulation

The set of policies relating to Defense Federal Acquisition Regulation (DFAR) is a body of regulatory policies that govern how defense contracts are conducted and carried out. These policies regulate areas such as: service, research, and business opportunities, limiting the disclosure of information related to these types of projects. To ensure compliance to these regulations, certain corporations may limit information flow to a need to know basis.

4.2.1 DFAR Encoding

The DFAR policy base is large and open to some interpretation, to ensure compliance, contractors may develop internal policies limiting the disclosure of information. Figure 4.2 displays an ex-

```
not(boolean(//ShipManifest/CargoManifest/NonContainerItems/NonContainerItem[
              (Consignee = "DoD" or
               Owner = "DoD"     or
               MarksAndNumbers/MarkAndNumber/Mark = "DoD Restricted") and
               boolean(Weight)]))
```

Figure 4.2: Non-disclosure of Department of Defense contracted cargo

ample of the encoding of one such policy. This policy relates to the freight shipping industry and limits what individuals can learn about crates belonging to the Department of Defense. The weight of a crate can be used to infer possible contents and therefore could be considered sensitive, thus a contractor would likely not share this information to be sure that it is not in violation of any of the DFAR regulations. This particular policy looks for crates that belong to the Department of Defense or subcontractors. The policy verifies that each of these crates does not have a weight attribute, if it does have a weight attribute, the policy will return a boolean $false$.

4.3 HIPAA

According to George Annas, at the heart of the concerns over medical information (more accurately its disclosure) is the need for trust between the patient and the physisician or care-giver [4]. A basic privacy doctrine has been in existence between patient and physician, in Anna's words, states:

> ...no one should have access to private health care information without the patient's authorization and that the patient should have access to records containing his or her own information, be able to obtain a copy of the records, and have the opportunity to correct mistakes in them.

The Health Insurance and Portability and Accountability Act (HIPAA) is regulatory policy base which is indented to provide the a regulatory framing of these ideas [4]. It is purposed with providing guidelines for fair disclosure of senisitve personally identifiable health information [36].

The HIPAA law basically designates that the "minimum necessary" information be shared, which essentially means that only a patient's physician should have access to the entire medical record [4]. Although it may seem that this policy base is pretty cut-and-dry, there is left plenty of room for interpretation. As such, different individuals may have different access rights to the patients information. For example, a patients physician would likely have access to all of the patients medical history, although, interpretations of the law do suggest that non-relevant procedures and medication may be withheld [15].

As the health care industry has grown, so to has the shift to electronic medical records (EMR) [34]. This movement has provide physicians with more information which ultimately allows them to make more informed decisions and provide better care [34]. A key concern associated with EMRs is associated with privacy and the disclosure of personally identifiable information (PII) [34] [4] [48]. According to a 2004 study, conducted by the California Health care Foundation, 75% of the respondents in the survey were not comfortable with maintaining EMRs with a third party. The key proponent in this discomfort is the lack of trust between the consumers and the EMR service providers. Trust is absolutely essential in this information sharing domain, and can be built and maintained through patient assurance that their medical records are being handled appropriately, and by providing the patient control over the disclosure of his or her record [48] [30]. CCA is a mechanism that can provide assurance and control of this level. Assurance can be built by verifying that information being shared is not in violation of any of the regulatory policies in place to protect its disclosure (for instance HIPAA). This requires an encoding of the policy bases (like HIPAA) in a machine readable format. The following sections discuss the policy base of HIPAA and describe the machine readable XPath representations of the policy base.

4.3.1 HIPAA Policy Encoding

HIPAA is made up of several parts. The piece that is most interesting to us is the portion dedicated to avoiding disclosure of sensitive private information. This regulatory base places responsibility

on parties handling patient data, this includes the hospital, the insurance companies, and any research groups. The following is an XPath encoding of this privacy portion of HIPAA as it pertains to our messages.

Assume we have the following actors:

Hospital Staff : processes the patient paper work

Insurance Group : handles the patients insurance claims

Research Group : search patient records for information related to illness

Policy for Actor: Hospital Staff

The hospital staff needs to be able to record patient visits and submit patient billing information to the insurance company, but the hospital staff does not need access to the patients medical history nor specifics about any procedures that the patient has been through. This represents an exception as dictated by the rules of the HIPAA legislation. The policy that applies to this particular situation is displayed in Figure 4.3

```
boolean(not(//PatientRecord[not(patient_informationrelease) or
            patient_informationrelease[not(node())]]
   [patient_health or
                   patient_personalinformation]))
```

Figure 4.3: HIPAA privacy policy for the hospital staff's use of the patient record

Policy for Actor: Insurance Group

The insurance group requires similar information as the hospital staff. The difference here, is that the insurance company cannot see anything unless an information release exists. Also, the insurance company cannot see any information related to any other insurance company associated

with this patient. Figure 4.4 displays the XPath encoding of this policy (we assume in this case that we are referring to the primary insurance company).

```
boolean(not(//PatientRecord[not(patient_informationrelease) or
patient_informationrelease[not(node())]]
   [patient_information/sex
            patient_information/patient_secondaryinusurance
    patient_health or
            patient_personalinformation]))
```

Figure 4.4: HIPAA privacy policy for the primary insurance company use of the patient record

Policy for Actor: Research Group

Research groups on the other hand, should be given no clue as to the identity of the patient (this includes name, address, phone number, SSN, etc.), but they should be provided access to patient medical history. The removal of all personally identifiable information from the records should guarantee that the research group has no ability to correlate an illness to a particular patient. In this case, the patient's information release is not required because no personally identifiable information is being released. Figure 4.5 displays the disclosure policy for the research group.

```
boolean(not(//PatientRecord[patient_information or
    patient_personalinformation
    patient_health/age
    patient_health/weight
    patient_health/height]))
```

Figure 4.5: HIPAA privacy policy for the research group's use of the patient record

Chapter 5

Approach

]

In this chapter we explore four approaches for enhancing the evaluation of our XPath policies. Methods we have explored include: XML Document Projection [29], systematic reduction of parsing, parallelization strategies [25], and stream based approach in SPEX [39]. For each of these strategies, we discuss how we incorporated them into the CCA mechanism.

5.1 XML Document Projection

A problem with traditional (in memory) XML processors is that they consume an excess of system resources (main memory). As a result of this limitation, exceedingly large documents are unable to be processed, while other larger documents suffer from the extreme overhead of creating in memory representations of the document. As a step toward a solution to this problem, XML document projection [29] was introduced. Document projection reduces the size of an XML document based on what is "needed." This need is extracted from XPath expressions that are to be evaluated over the document. A side effect of this search space reduction is the shorter evaluation time of XPath expressions. This effect is what peeked our interest in this technique.

The projection, D_p, of an XML document, D, is defined as the set of nodes within D, such that, a query, Q, executed on D_p returns the same results as Q executed on D. In other words D_p contains

```
SimplePath ::= Axis NodeTest
Axis       ::= child::
               | self::
               | descendent::
               | descendent-or-self::
               | attribute::
NodeTest   ::= ((NodeName | *):) ? (NodeName|*)
               | node()
               | text()
NodeName   ::= any combination of literals
```

Figure 5.1: Projection Path Grammar

all of the nodes necessary to properly evaluate query Q. The techniques proposed by Marian and Siméon rely on static analysis of the query to build a set of required elements in a document.

5.1.1 Static Path Analysis

The authors of [29] intended this technique to be used in conjunction with the XQuery language, of which XPath is a subset. The most important aspect of the static analysis is observing the XPath expressions. As a result, we ignore the discussion on handling XQuery exclusive language constructs and concentrate only on the portion of the literature dedicated to XPath analysis.

The purpose of the static path analysis is to determine a set of paths that are relevant to the query. This set of paths is referred to as the *projectionpaths*. As a simple example, consider Example 5.1, an XPath statement (which is a valid XQuery as well).

$$//RAAR/Ship[NameOfShip =' Posiden' or NameOfMaster =' DAlessandro'] \quad (5.1)$$

Example 5.2 shows the *projectionpath* set that would be derived from Example 5.1.

$$\{//RAAR/Ship/, //RAAR/Ship/NameOfShip, //RAAR/Ship/NameOfMaster\} \quad (5.2)$$

As can be seen, a *projectionpath* is a simple XPath representation of the enumeration of paths from the root node of the document to nodes necessary for full evaluation of the query. The grammar describing *projectionpaths* is displayed in Figure 5.1. The language generated from this grammar contains expressions that only use forward axes, and are of the form of the path displayed in Example 5.2. Two sets of paths are defined in [29]: *returned* and *used*. The union of these two path sets is the *projectionpath* set. *Returned* paths are those paths that select and return a set of nodes, while, *used* paths are those that are necessary to the evaluation of the expression, but which do not return a node set. Referring back to Example 5.1, the set of *returned* paths is:

$$\{//RAAR/Ship\} \tag{5.3}$$

while the set *used* paths is:

$$\{//RAAR/Ship/NameOfShip, //RAAR/Ship/NameOfMaster\} \tag{5.4}$$

The paths in Example 5.4 are necessary for filtering the nodes that will belong to the set of nodes in Example 5.3, but do not return any nodes themselves to the result of the query, so are therefore part of the *used* set of paths.

5.1.2 Application of Projection Paths

Using the *projectionpaths* inferred from the query, an XML source document can be filtered to only contain relevant information. To do this, the source XML document, *D*, must be parsed using the Simple API for XML (SAX) [31], an event based XML parser. The only SAX events of use to the projection algorithm are:

- CHARACTERS (String)
- OPENING_TAG (QName)

- CLOSING_TAG

The SAX parser will allow for a traversal of the abstract source tree in a left-deep recursive manner. Parsing the document in this manner allows for the creation of *contextpaths* which can be compared to the *projectionpaths* gathered from the analysis of the query. While parsing the document (observing the events returned by the SAX parser), the algorithm will select one of four possible actions:

1. **SKIP**: do not copy this node over to the projected document tree
2. **KEEP_SUB_TREE**: copy this node and its subtree to the projected document tree
3. **KEEP**: just copy this node (without its subtree) to the projected document tree
4. **MOVE**: more information is needed, so take no action right now and move to the next event (node)

Which action to apply is determined by comparing the *projectionpaths* to the current *contextpath*, and copying the node (and possibly subtree) to the projected document only if a *projectedpath* is satisfied.

To better explain how the projection paths affect the form of the projected document, we will go through a simple example. Consider the XML document displayed in Figure 5.2. Assume that the query we wish to evaluate over this document results in the set of projection paths displayed in Example 5.5

$$\{//RAAR/Source/AgencyID, //RAAR/Ship/NameOfShip\} \tag{5.5}$$

This document will be processed in a depth first traversal (SAX-based parse), while a set of currently open paths is maintained. The projection algorithm processes one SAX event (described previously) at a time. A node may only be loaded once a CLOSING_TAG event is received (at which point a decision is made as to whether that node belongs in the projected document). Referring back to our example, a SAX parse of our document will begin with the root node, *RAAR*. The

```
<RAAR>
   <Source>
      <AgencyID>10035</AgencyID>
      <AgencyName>Semico Devices</AgencyName>
      <Contact>
         <Name>Eleanor Crawford</Name>
         <Phone>(780) 176-8942</Phone>
         <Email>ecrawford@erat.ca</Email>
      </Contact>
   </Source>
   <Ship>
      <NameOfShip>Island Time</NameOfShip>
      <NameOfMaster>Rhona F. Cruz</NameOfMaster>
      <NationalityOfShip>IRQ</NationalityOfShip>
   </Ship>
</RAAR>
```

Figure 5.2: Document before Projection

root node, of course, will always appear in the projected document. Next, we receive an OPENNING_TAG event, for *Source*. Looking at our projection paths in Example 5.5, we can see that source is part of the first path, so we will ultimately need to keep this node, the algorithm will then take a MOVE action, because more information is needed. The next event emitted by the SAX parser will be the OPENNING_TAG for the *AgencyID* node. Again, this node is part of the first projection path (a child of the *Source* node). At this time, we can copy over both the *Source* node and the *AgencyID* node (it is implied that the text, "10035" within *AgencyID* will also be copied) The next siblings of the *AgencyID* will all be skipped (including their subtrees). Next we process the node labelled *Ship*. Since this is a node that is captured by the second projection path in our set (Example 5.5), we know that this node will eventually be copied to the projected document tree. At this time, though, more information is needed, so the algorithm takes the MOVE action. The algorithm moves on to the next event emitted from the parser, the OPENNING_TAG for the *NameOfShip* node. This node is, again, part of the second projection path, and should be copied over. The algorithm holds off until we receive the CLOSING_TAG event from the parser, which

```
<RAAR>
   <Source>
      <AgencyID>10035</AgencyID>
   </Source>
   <Ship>
      <NameOfShip>Island Time</NameOfShip>
   </Ship>
</RAAR>
```

Figure 5.3: Document after projection

indicates that we must copy over the node, and its parent. The siblings of *NameOfShip* are not part of any of the projection paths, and therefore are skipped. When we reach the end of the document we will have created the projected document, displayed in Figure 5.3.

5.1.3 Applications to CCA

The main desire to explore this mechanism was the search space reduction that it provided. In the paper that originally discussed the document projection process, a reduction in query time is displayed for most of the queries in the XMark [45] XML benchmark. Another part of the argument was that the overhead of including path analysis in the parsing and loading stages of XML processing is fairly negligible. It goes on to discuss that the situations in which overhead is observed are those in which projection is not very well suited. These situations include paths that select most of the nodes in the document.

We expected most real world policies to not rely on the full examination of a message. As a result, we fully expected this approach to provide some benefit, if it could be incorporated with as little overhead as possible. According to the authors of [29], this method could be incorporated into the early stages of XML parsing and loading. This was where we would incorporate the process.

The query is compiled (parsed and validated) as normal, and the result of this is stored. Before evaluating the policy over the document, the projection paths are extracted from the policy, and a document projector is created from these paths. This projector will become a stage in the document

processing procedure. While the internal representation of the document is built, each element is sent to the document projector where it is determined to be necessary or not. After the document is fully processed, only the elements necessary for evaluation remain. This projected document is what the policy is evaluated over.

5.2 Limited Parsing

The simplest and most minor of all our strategies, this approach was derived from our initial explorations of operation of CCA mechanism. During these initial observations, we observed a glaring problem. From the profiling of our system it became quite obvious that one main issue was the parsing of the XML before XPath evaluation.

From some simple investigation into the Compliance Module code, we could see that we were re-parsing the document at times that were unnecessary. While it is necessary to reset and rebuild some internal data structures before each XPath evaluation, it is not necessary to fully re-parse the document.

The modification that we made was to simply parse the document once, create a reference to this in-memory representation (object), and use this object to rebuild the necessary data structures before each XPath evaluation. This method works in our situation because we do not anticipate documents that will be larger than those that can be stored in memory.

5.3 Parallelization of Evaluation

Another systems based approach we explored was to parallelize the XPath evaluation. To accomplish this, we adopted an approach similar to the Fork/Join threading pattern outlined by Doug Lea in Concurrent Programming in Java [25]. The Fork/Join threading pattern provides a means of breaking down a problem into subproblems (forking), solving the subproblems in parallel, and

sharing the solutions to the subproblems (joining).

A lightweight framework implementation of this pattern is described in [25]. The framework defines two main classes of objects: *FJTask* and *FJTaskGroup*.

FJTask: this is a lightweight implementation of the Java *Thread*. It ignores the use of typical Java *Thread* synchronization techniques and monitor methods (e.g. no suspension methods). Instead synchronization responsibility of these tasks is left up to the *FJTaskGroup*.

FJTaskGroup: this class is tasked with managing the FJTasks. All monitor controls is handled within this class. New *FJTasks* can only be created by this class (often limiting the number of concurrent threads to the available CPUs).

Lea goes on to suggest some basic steps for defining the pieces to solve a problem using the fork/join thread pattern. These steps include:

Define Task Class fields should be included in the task to hold both arguments to the task and results generated from the execution of the class. It is best to make these fields local to the task, to avoid issues of synchronization. This class should include a constructor to initialize the necessary class fields, and a *run* method that will execute the task's workload.

Define a Coordinator code should be created to monitor and coordinate the tasks. This includes: creating the subtask objects, forking the task (make it execute in parallel), join the tasks when they complete their execution, and combine the results of the tasks.

5.3.1 Applications to CCA

Although, the Fork/Join strategy was designed with parallelization of recursive divide-and-conquer algorithms in mind, with some slight rethinking, the same basic algorithm could be used to solve our problem as well. The main allure of this approach is that we could cheaply thread off workers to evaluate policies, and subsequently verify the result of the policy. Figure 5.4 displays a typical

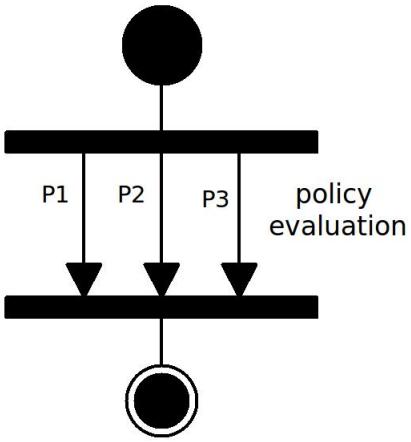

Figure 5.4: A view of the Fork/Join Operation in CCA

fork/join parallel process. In this figure, each of the processes represents a policy. The policies either evaluate to $true$ or $false$. Once a policy completes its evaluation, its result is collected and a conjunction is made between its result and the results of previous policies. If the result of this conjunction is $false$, execution stops, currently executing threads are interrupted and the message is deemed non-compliant. In the case of our implementation, the number of policies allowed to execute in parallel is limited to the number of available processors.

Taking the advice of Lea, we have created two separate classes: *ComplianceCoordinator* and *ComplianceWorker*. The *ComplianceCoordinator* contains all of the code necessary to manage the *ComplianceWorker* threads. The *ComplianceWorker* class is an internal class of the *ComplianceCoordinator*. One example of the classic fork/join pattern not fitting our problem exactly is that our tasks necessary to solve a problem can differ greatly in both number and time necessary to complete. An assumption of the classic fork/join problem is that the tasks typically take close to the same amount of time. Lea discusses this situation and explains how the classic fork/join pattern can be expanded to use call backs.

These call backs allow for the accommodation of asynchronous tasks. This solution will only work for problems in which there is no necessary ordering for the sub tasks completion, which fits our problem particularly well. The call back extension makes use of Java's monitor mechanisms,

```
if FAIL then
   interrupt policies
   terminated ← TRUE
   NOTIFY coordinator that we are done
else
   if POLICY_QUEUE = EMPTY then
      terminated ← TRUE
      NOTIFY coordinator that we are done
   else
      POLICY_QUEUE.dequeue()
   end if
end if
```

Figure 5.5: pseudo-code for termination of parallel policy evaluation

and waits on some condition. The condition in our case is the complete evaluation of all policies (indicating compliance to all) or any policy finding some non-allowable information (indicating non-compliance).

To solve our particular problem we employed the fork/join pattern and the call back extension. To aid in our implementation of these ideas, we also employed a queuing mechanism which allows us to easily manage the operations on our tasks. These operations include: retrieving the next policy (task), checking if any policies remain to be evaluated, and interrupting any currently evaluating policies. Our *ComplianceCoordinator* waits on a *terminated* condition to become true, once it is true, the *ComplianceCoordinator* takes over execution. Figure 5.5 describes the criteria we use to determine whether or not we must terminate. This particular block of code is run after each policy completes evaluation (it is synchronized and guarantees mutual exclusion). We first check if the policy that just completed failed, if so, we interrupt any currently running policies, set our terminated flag to true, and notify the coordinator (yielding execution to that thread of execution) that it can check the terminated flag. If the policy did not fail its evaluation, we check if any other policies are waiting to execute. If no policies are waiting, we set the *terminated* flag to true and inform the coordinator that it can check the flag, otherwise, we grab the next policy and trigger its

evaluation.

Inside of each *ComplianceWorker* thread, XPath evaluation is carried out in exactly the same manner as it was in the baseline scheme. This form of evaluation was not immediately lent to parallel execution as some of the internal structures of evaluation were not thread safe and caused deadlock issues under certain situations. To remedy this problem, we made copies of the message to be evaluated and dispersed these copies across the *ComplianceWorker* threads. This eliminated the synchronization issues, but added extra overhead. With this in mind, we are not necessarily ruling out this strategy as a possible area of further exploration in the future.

5.4 Streamed and Progressive Evaluation of XPath

Streamed and progressive evaluation of XPath (SPEX) is a stream based query processor. The goal of this particular processing strategy is to return a result as soon as possible, while not reducing the expected data throughput. Unlike traditional XPath evaluation techniques, SPEX does not rely on a DOM-tree based representation of an XML document, instead, SPEX operates on a stream of XML (generated through a parsing technique like SAX).

5.4.1 XML Streams and Annotations

These streams of XML are generated through a depth-first, left-to-right, preorder traversal of the abstract XML tree. In this stream nodes are represented by an opening tag and closing tag (its opening tag is first appended to the stream upon discovering that node, and the closing tag is appended once that node is out of scope). Opening tags in the stream may also be followed by annotations. These annotations are used to mark nodes that could possibly be included in the result of an XPath query evaluation. Annotations appear as a sequence of integers in ascending order, for example, [1,2,3]. There also exists two special annotations: [] (empty annotation) and [0] (full annotation, which indicates the total sequence of available integers). A set of operations exist for

```
query: '/' path | path | query 'union' query | query 'except' query
path:  step( '/' step )*
step:  axis '::' nodetest ( '[' pred ']' )?
pred:  pred 'and' pred | pred 'or' pred | 'not' '(' pred ')' |
       '(' pred ')' | path
```

Figure 5.6: SPEX XPath Fragment in EBNF

manipulating and comparing these annotation sets:

- union (symbol): inclusion of all (without duplication) from all sets
- intersect (symbol): only common items
- inclusion (symbol): subset

5.4.2 SPEX XPath Fragment

SPEX utilizes a fragment (subset) of XPath that only considers forward axes. To clarify a forward axis is one that does not rely on previously traversed (in document order) nodes (e.g. descendent or child), while a backward axis does (e.g. parent or ancestor). The omission of these backwards axes is not unreasonable because as is shown in [40], any query containing backwards axes can be rewritten as one containing only forward axes. The XPath fragment used by SPEX in EBNF is displayed in Figure 5.6.

5.4.3 Query Processing

SPEX maintains two compact data structures that serve as the current state of discovery for an XPath query, these two structures are:

Matchings: set of lists, one for each location step in query being evaluated. Items in a list represent nodes that match the currently evaluated location step.

Candidates: a buffer of possible answers to the query (consists of nodes that match the location step immediately before a predicate in the XPath query.

The matching list for location step, L_i, is used as the context node set for location step L_{i+1}. By this logic, if there exists n location steps leading up to a predicate, p, then, the candidate set of predicate p should be equivalent matching list associated with location step, L_n. Upon reviewing a predicate, an entry, e_i may be replaced by $true$ if the predicate evaluates to true for that particular node (if no predicate exists, e_i existing in the list implies $true$). The moment it is determined that a node does not satisfy a predicate, it is immediately removed from the candidate list. As candidates are verified as final answers to the query, they are output to a separate buffer which will be returned upon reaching the end of the stream input.

5.4.4 SPEX Transducer

SPEX utilizes a network pushdown automata with output tape (referred to as a transducer) to verify a node's membership to the XPath query result set. Each location step is defined as its own transducer in the network (the full network representing the entire XPath query). The list discussed in the previous section (matchings and candidates), and communication between transducers is facilitated through the input and output tapes for each of the transducers. Consider two transducers, T_i and T_{i+1}, the output tape of T_i will be in the the input tape of T_{i+1}. This form of communication is absolutely necessary due to the way XPath is specified. Each location step represents a set of nodes relative to a certain context, which is derived from the set of nodes selected by the previous location step. With this in mind location step S_i will output its candidates to its output tape, these candidates then become the input (or context set) for location step S_{i+1}.

A SPEX transducer is a simplification of the classic pushdown automata transducer. It is a single-state deterministic pushdown automata, whose input and output alphabet is the set of available node tags (both opening and closing as well as possible annotations) in the XML document

being processed. The stack alphabet consists of the set of possible annotations in the XML document. A special SPEX transducer exists at the end of the network called the *HEAD* transducer. Its purpose is to mark non-empty annotations with a flag, indicating that this node is to be included in the query answer. Although a transducer exists for each of the forward axes defined in the XPath specification [7], we will only go through the two most widely used: child and descendant. Each of these single-state transducers are defined by a set of transition rules. To avoid getting bogged down by the notation, these transition rules will be presented as English descriptions. For each of the following descriptions, assume that we are processing a stream of XML tags ($< t >$),annotations ($[c_i]$) and stored annotations ($[s_j]$).

child This transducer moves annotations of current context nodes to their children. The child axis transducer is defined by the following transition rules:

1. upon receiving annotation $[c]$ from the input stream, annotation $[c]$ is pushed onto the stack and nothing is sent to the output tape
2. upon receiving an opening tag, $< t >$, the tag is sent to the output tape, followed by the annotation currently on top of the stack
3. upon receiving a closing tag, $< /t >$, the tag is sent to the output tape, and the annotation on the top of the stack is popped

descendant This transducer moves annotations of the current context nodes to all of their descendents (nodes below it in document order). The only difference between this transition rule set and that of child is the first rule. This first rule is responsible for maintaining the current context node's annotation. The descendant axis must consistently push annotations down the line so that all sub nodes are appropriately marked. The descendant axis transducer is defined by the following transition rules:

1. upon receiving an annotation $[c_i]$ from the input stream, union annotation $[c_i]$ with

$[c_{i-1}]$ and push this annotation onto the stack. Nothing is sent to the output tape.

2. upon receiving an opening tag, $< t >$, the tag is sent to the output tape, followed by the annotation currently on top of the stack
3. upon receiving a closing tag, $< /t >$, the tag is sent to the output tape, and the annotation on the top of the stack is popped

nodetest This transducer filters out nodes not matching the specific nodetest. This is done by outputting an empty annotation for nodes not matching the node test. The following four transition rules are defined to accomplish this task:

1. upon receiving an opening tag, if the tag matches the node test, write the node and its annotation (unmodified) to the output tape.
2. upon receiving an opening tag, if the node does not match the node test, write the node and an empty annotation (committing its annotation) to the output tape.
3. upon receiving a closing tag (associated with the matching node), write that node to the output tape
4. upon receiving a closing tag (associated with the non-matching tag), write that no to the output tape

To better understand the processing of SPEX, let us consider an example processing scenario. The input stream for this scenario is presented in Example 5.6, and query over this input in Example 5.7.

$$< a > [1] < a > [2] < b > [3] < /b >< /a >< /a > \tag{5.6}$$

$$/descendant :: b \tag{5.7}$$

The input stream in Example 5.6 serves as the input tape for the descendant transducer. Assume that the stack is initialized to an empty annotation ([]). The descendant transducer will take the

following steps:

process node $< a >$ since $< a >$ is an opening tag, we follow second transition rule defined above: the tag $< a >$ is sent to the output tape, as well as the annotation on the top of the stack, which is the special annotation, [].

process annotation 1 here we follow the second transition rule for the descendant axis. In this case, the annotation is pushed onto the stack and a union is performed on the current stack top and the newly pushed annotation. Thus the new stack top is: $[1]$.

process node $< a >$ According to the transition rules we write the node $< a >$ followed by the current top of the stack (which is $[1]$ right now) to the output tape.

process annotation 2 according to the first transition rule, we must union the new annotation with the top of the stack. Now, the current top of the stack is: $[1,2]$.

process node $< b >$ here the transition rule two is once again applicable. We write the node $< b >$ and the stack top ($[1,2)$ to the output tape.

process annotation 3 the first transition rule is applicable here. The new annotation ($[3]$) is combined with the current stack top ($[1,2]$), this union ($[1,2,3]$) is then pushed to the stack.

process node $< /b >$ the closing tag indicates we should use the third transition rule. This indicates that we write the closing tag to the the output tape and the top of the stack is then popped. This makes the new stack top: $[1,2]$.

process node $< /a >$ once again the third transition rule applies. The tag is written to the output tape, and the stack top is popped. The current stack top is now: $[1]$.

process node $< /a >$ again, the third transition rule applies. The tag is written to the output tape and the stack top is popped. The top of the stack is now: [].

end of stream The output tape generated from these processing steps will become the input for the node test transducer.

The output tape generated by this descendant transducer is displayed in Example 5.8. This tape will become the input for the next transducer in the network, the node test transducer.

$$< a > [] < a > [1] < b > [1,2] < /b >< /a >< /a > \tag{5.8}$$

The node test transducer will examine each of the nodes in the input stream displayed in Example 5.8. The node test transducer in this case is only looking for nodes that match the *b* tag. According to the transition rules discussed above, a node is written to the output tape regardless of whether it matches or not. The only difference is that if the node matches, its annotation is written as well, an empty annotation is written otherwise. The output tape of this transducer is displayed in Example 5.9

$$< a > [] < a > [] < b > [1,2] < /b >< /a >< /a > \tag{5.9}$$

This output tape then becomes the input of the final *HEAD* transducer. This transducer will simply collect the nodes whose annotations are not empty. In this case the node $< b >$ is the only node with a non-empty annotation. Thus, the set $\{< b >\}$ is our result.

5.5 Applications to CCA

From the literature review [24] [35], it was determined that selecting the appropriate XML parsing strategy is essential for maintaining a quality level of performance. Lam et al [24], cite instances when each of the different XML parsing strategies work most effectively. According to their studies, stream based parsing (SAX for example) works most effectively if the application does not need to update, modify, or perform some kind of random access on the document [24]. Based on this fact and our own observations on the effect that parsing has on performance, we were much

more inclined to discovering a stream based processing strategy.

There are a few of aspects of our problem that require a unique strategy for XPath evaluation. First, we must be able to handle a possibly large set of XPath expressions (policies). Second, we only require that the queries determine the existence of some data (not required to collect or return any set). Last, we must fail as quickly as possible if non-compliance is determined. Of the streaming XPath processors that we have studied, SPEX displayed the highest level of maturity, covering all of the aspects that we require for our problem.

Although, the current description of SPEX does not include a means of evaluating multiple queries over one stream, the ability to integrate this in the future is present. Due to the way that SPEX compiles the queries (into transducer networks) it would not be unreasonable to believe that this compilation step could be exploited to create a transducer network representing the joining of multiple queries (policies). Although this has not been implemented yet in our system, the promise of this opportunity indicated that SPEX represents an appropriate solution to our problem.

Another advantage of SPEX over traditional DOM and even the other stream based strategies is the progressive nature of its evaluation. By this we mean that SPEX will perform the query while parsing the document. This allows for a quick determination of the fulfillment of the query, which is extremely useful in cases of non-compliance is determined. In this case a traditional DOM based query processor would have already fully parsed the document into a tree before determining the query result. With SPEX, the query evaluation occurs progressively and therefore will not parse the entire document if the query is determined to be non-compliant (i.e. it finds a single element that is not allowable).

The final attractive advantage of SPEX was that it could be optimized to handle “true” or “false” query evaluations. This meant we could completely eliminate the overhead (however small) of building and returning a (possibly large) result set. This also meant that, as previously discussed, we could short circuit evaluation if we have seen sufficient evidence to draw a conclusion.

Chapter 6

Real Policy Results and Discussion

In this chapter, we present observations gathered from testing the proposed modifications for XPath evaluation using our Real policy set.

6.1 Evaluation

Using our test-bed, we tested the proposed evaluation strategies and observed the results. Each strategy was subjected to a message load of 100,000 messages with the same characteristics as described in the baseline results section. Below, we report the results for each of these strategies.

We will rank our strategies based on their average message throughput numbers and relative speedup from the baseline. The formula for the **average throughput time (ATT)** is displayed in Equation 6.1 and the overall throughput in **messages per second (MPS)** is described in Equation 6.2. The **speedup** over the baseline is mathematically described in Equation 6.3.

$$ATT = \frac{TOTAL\_MESSAGES}{SUM\_OF\_MESSAGE\_TIMES} \tag{6.1}$$

$$MPS = \frac{1}{ATT} \tag{6.2}$$

$$SPEEDUP = \frac{MPS_{Strategy}}{MPS_{Baseline}} \tag{6.3}$$

In our evaluation of the strategies, we will also consider the simplicity of implementation and room for future improvements. In each section we report throughput numbers broken down by message size (each size range is referred to as a tier). The organization of messages in this manner allows us to better observe situations in which certain strategies work best.

6.2 Document Projection

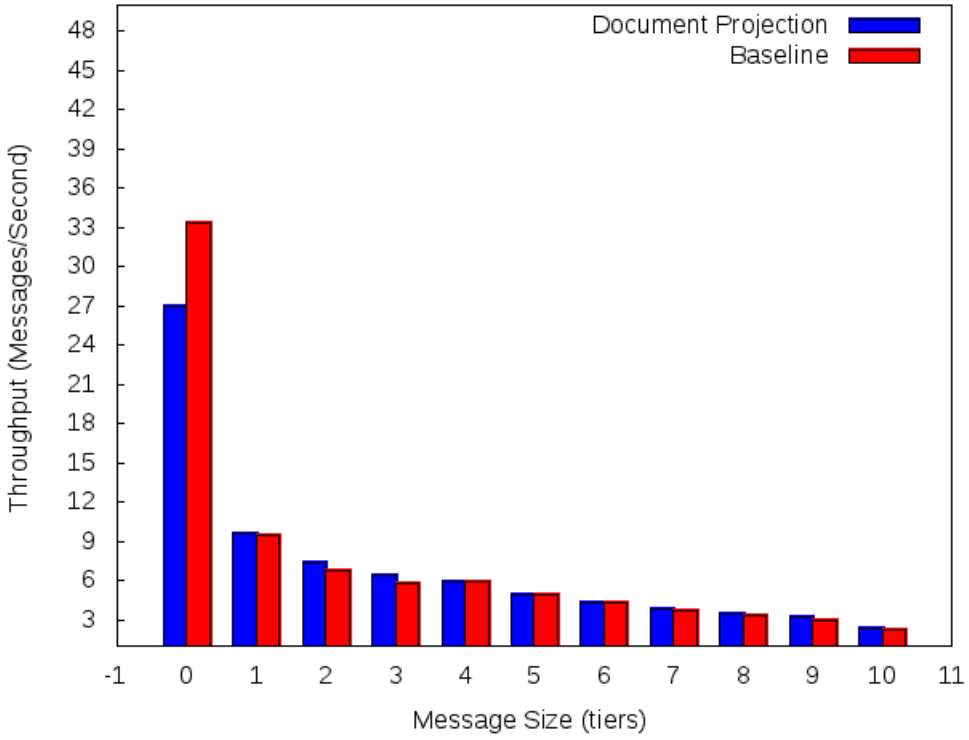

Figure 6.1: Message throughputs for Document Projection using the real policy set

The results for the Projection strategy are displayed in Table 6.1. A comparison of the speed-up results for this strategy are displayed in Figure 6.1. For our real policy set, document projection provided us an average throughput of: **14.71** which is actually lower than the baseline. We can see

Real Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	27.03	9.62	7.41	6.33	5.88	4.93	4.33	3.83	3.45	3.2	2.32
Speed-up	0.81	1.02	1.09	1.1	1.0	1.0	1.02	1.05	1.02	1.07	1.06
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 6.1: Compliant Throughput Results per Message Size, using the Projection Strategy

from the results from the real policy set, that for smaller messages, document projection does not provide much of an improvement. In fact for a smallest tier (where we expect to see most of our messages fall), it performs much worse than the baseline. This is to be expected, and is likely why the overall average throughput is much lower than the baseline. Although, document projection does not provide much in the way of overhead, if the documents are small enough, the search space reduction is not enough to warrant a benefit greater than the overhead of the projection. We can see for larger messages that document projection always performs better than the baseline, although it is a fairly modest improvement.

6.3 Limited Parsing

Real Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	40.0	11.24	7.81	6.45	6.06	5.13	4.5	3.83	3.41	2.99	2.19
Speed-up	1.2	1.19	1.15	1.12	1.03	1.04	1.06	1.05	1.01	1.0	1.0
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 6.2: Compliant Throughput Results per Message Size, using the Limited Parsing Strategy

The results gathered for the limited parsing strategy are displayed in Table 6.2. A comparison of the peformance of the limited parsing strategy to the baseline is displayed in Figure 6.2 For the real policy set, this strategy yielded an average message throughput of: **17.5439** which is a relative speed-up of: **1.12** over the baseline. The throughput numbers for this strategy on are not

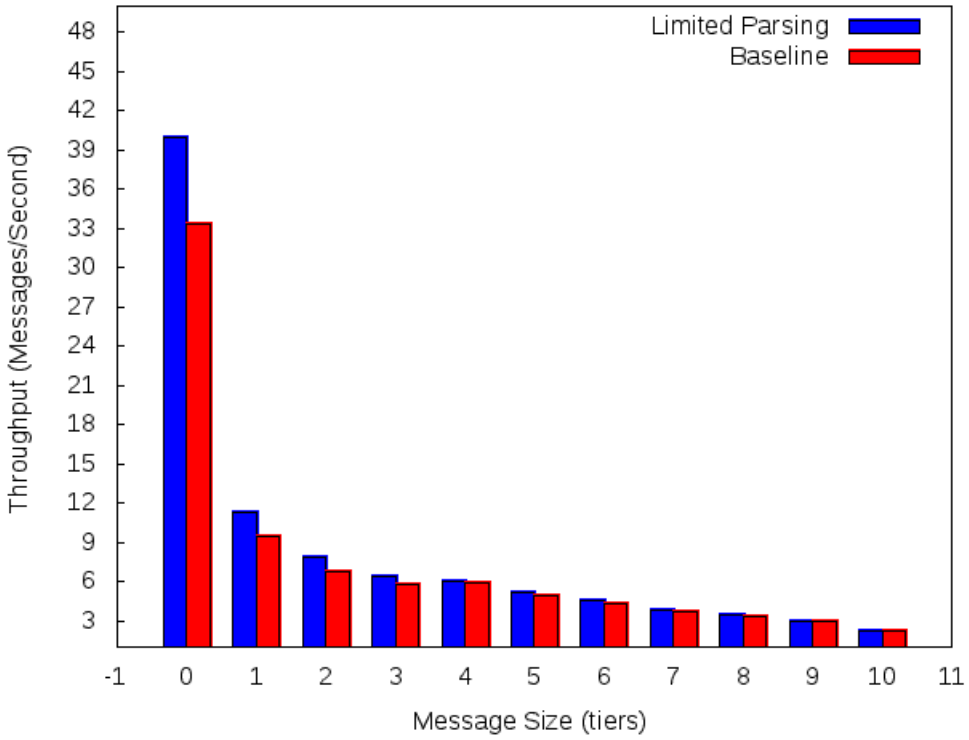

Figure 6.2: Message throughputs for Limited Parsing using the real policy set

overwhelmingly better than the baseline. This strategy seems to work best under small documents, and falls apart as we begin to deal with larger documents.

6.4 Parallelization

Real Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	37.0	10.64	7.58	6.25	5.92	4.98	4.27	3.75	3.37	2.95	2.16
Speed-up	1.11	1.13	1.11	1.08	1.01	1.01	1.0	1.02	1.0	0.99	0.99
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 6.3: Compliant Throughput Results per Message Size, using the Parallelization Strategy

The results for our parallelization strategy are reported in Table 6.3. A comparison of the

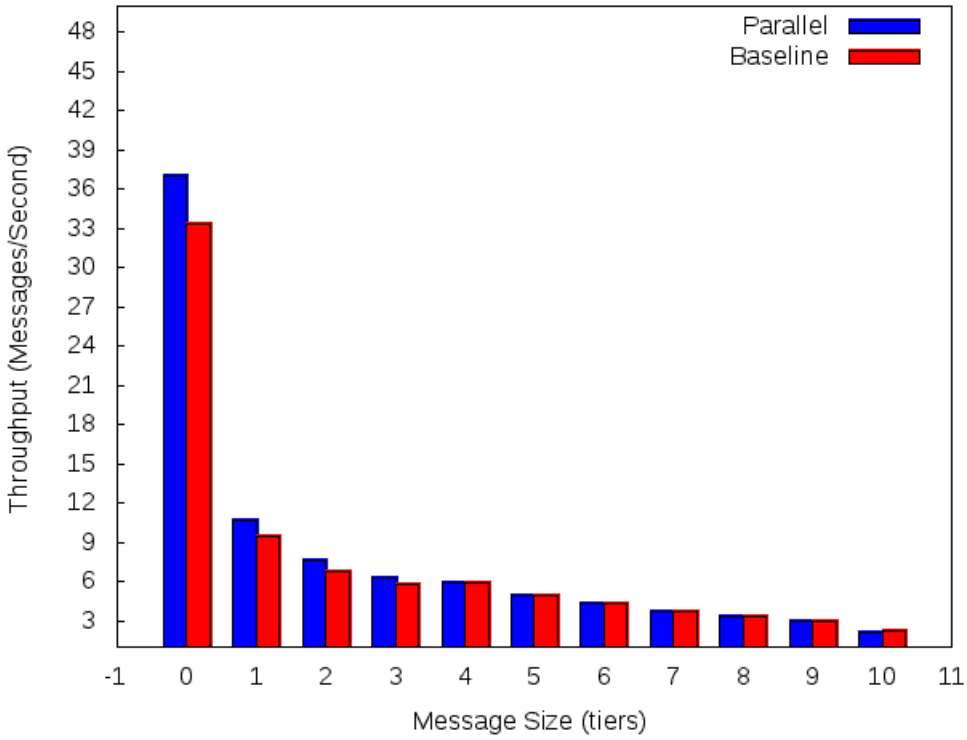

Figure 6.3: Message throughputs for the Parallelization Strategy using the real policy set

performance of the parallelization strategy against the baseline is displayed in Figure 6.3. This strategy yielded an average message throughput of: **16.6667** which is a relative speedup of: **1.06** over the baseline. Like the limited parsing strategy, parallelization performs around 10% better than the baseline for smaller messages, but falls apart for larger messages (for the largest, performing worse than the baseline).

6.5 Stream Based Evaluation (SPEX)

The results for our stream based evaluation strategy (using SPEX) are reported in Table 6.4. A comparison of results for SPEX compared against the baseline is displayed in Figure 6.4. This strategy yielded an average message throughput of: **18.18** which is a relative speed-up of: **1.16**

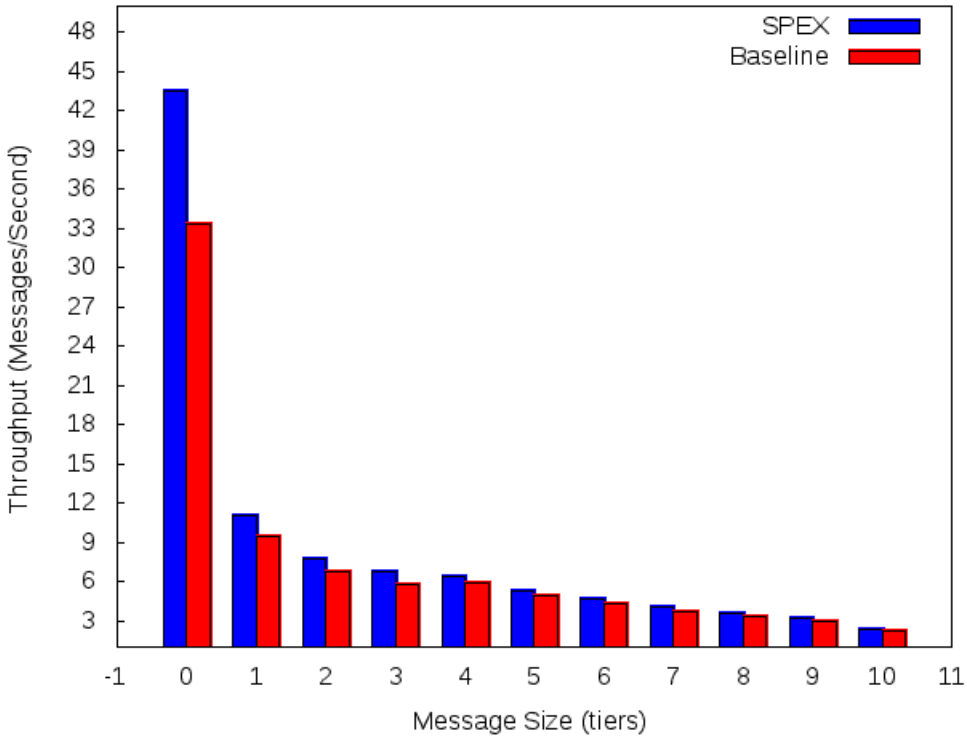

Figure 6.4: Message throughputs for SPEX using the real policy set

Real Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	43.48	10.99	7.75	6.76	6.45	5.32	4.67	4.1	3.56	3.22	2.3
Speed-up	1.3	1.17	1.14	1.17	1.1	1.08	1.1	1.12	1.06	1.08	1.06
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 6.4: Compliant Throughput Results per Message Size, using the Stream Based Strategy

over the baseline. We can see in the results that SPEX overall performs better than any of the previous explored strategies. For the smallest tier it performs 30% better than the baseline and throughput numbers of at least 6% better for all remaining tiers. For the larger tiers (where we expected document projection perform the best), SPEX performs better than all other strategies.

6.6 Strategies Compared

Comparing all of the employed strategies, we begin to see certain patterns emerge. Figure 6.5 displays message throughput speed-up (relative to the baseline strategy) results for all of the strategies using the real policy set.

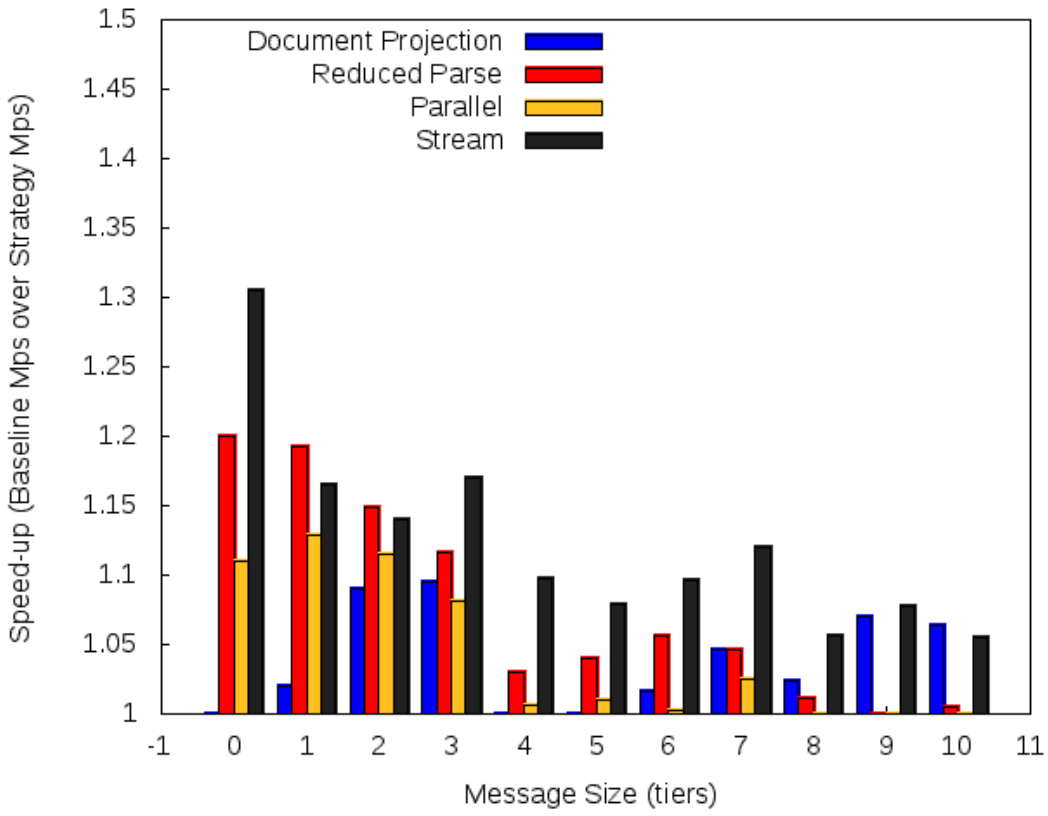

Figure 6.5: Message throughput speed-up (relative to baseline) for all strategies using real policy set

Looking at Figure 6.5, it is clear for the larger messages that both stream-based evaluation (SPEX) and document projection provided the greatest throughput performance, with a slight edge to SPEX. We can also see that SPEX, out performs all other strategies across the board, performing exceptionally better in the smallest group of messages, where the majority of our messages are contained. For this reason, we conclude that SPEX works best under the real policy set conditions.

Chapter 7

Synthetic Policy Results and Discussion

In this chapter we discuss the results we gathered from the synthetic policy set. We provide an explanation as to why this is not necessarily an accurate representation of policy sets that we are likely to encounter, but yet, we still may be able to learn something from these observations.

7.1 Synthetic Instance Generation

At an attempt to observe the scalability of our proposed algorithmic and systematic improvements, we constructed a policy generation tool. This tool was used to create policy sets of a large number of simple policies (consisting of only location paths). The union of all sets collected by these policies consisted of every element in the document. These instance sets were meant to provide an absolute worst case in the number of policies to be evaluated.

Instance generation, no matter what the domain, is difficult. An example of an extreme failure of instance generation is discussed in Selman, Livesque, and Mitchell's paper "A New Method for Solving Hard Satisfiability Problems" [47]. In this paper we learn of an instance generation procedure proposed by Allen Goldberg. This procedure took years to create and proved to not be very representative of the hard problems that they were supposed to be instances of. The lesson from all this is that instance generation is a difficult process that may never yield ideal results.

Thus, our hastily built policy generation tools may not be very representative of our worst case policy set. Nevertheless we feel that the observations presented here may provide an indication of a need for future exploration in the area of policy representation.

7.2 Document Projection

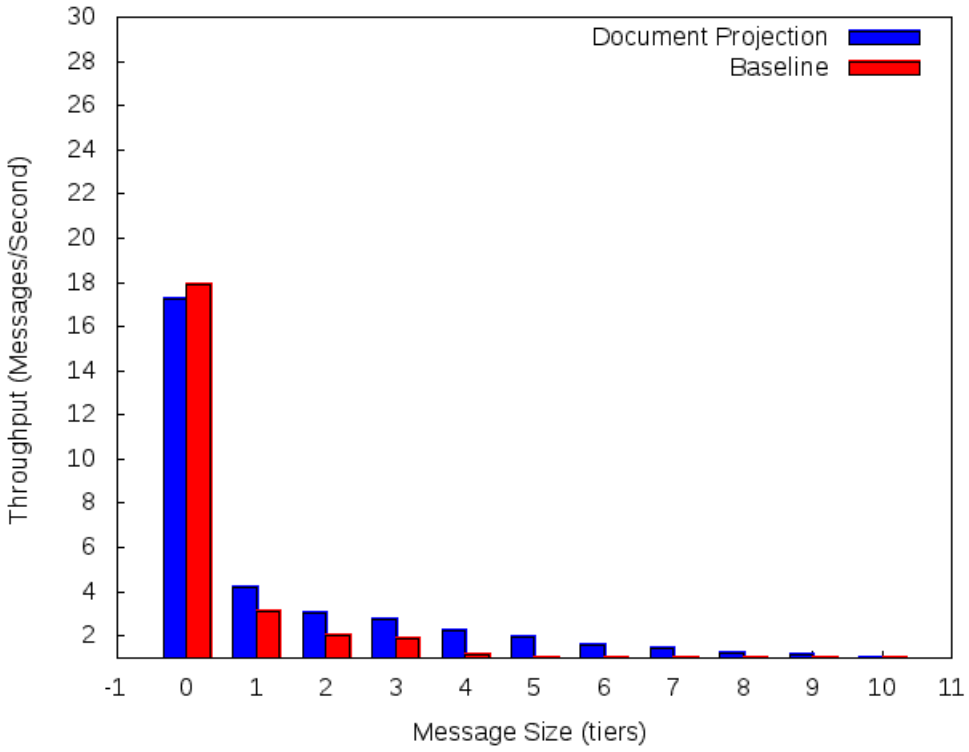

Figure 7.1: Message throughputs for Document Projection using the synthetic policy set

Synthetic Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	17.24	4.2	3.01	2.75	2.23	1.95	1.6	1.45	1.2	1.13	0.86
Speed-up	0.97	1.36	1.51	1.48	1.91	2.41	1.6	2.59	1.96	1.78	1.98
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 7.1: Compliant Throughput Results per Message Size, using the Projection Strategy

The results for Document Projection using the synthetic policy set are displayed in Figure 7.1 and Table 7.1. For the synthetic policy set, document projection yields message throughput numbers of: **4.74** which is a relative speed-up of: **1.6** over the baseline. Once again, it is evident in the results that document projection provides a benefit in the evaluation of larger messages. The advantage of document projection under these conditions is much more pronounced over what we observed in the real policy set experiments. Another difference in these results is almost all message sizes see an increase over the baseline. This is consistent with what we would expect from document projection. All but one of the policies in the synthetic policy set select only a subset of the nodes in a message. Most of these select a relatively small subset, thus, document projection would be expected to significantly reduce the search space under these conditions.

7.3 Limited Parsing

Synthetic Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	25.0	2.75	2.09	2.09	1.58	1.54	1.96	1.29	0.88	1.17	0.76
Speed-up	1.4	0.89	1.04	1.13	1.36	1.9	1.96	2.31	1.44	1.84	1.76
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 7.2: Compliant Throughput Results per Message Size, using the Limited Parsing Strategy

The results for the Limited Parsing strategy using the synthetic policy set are displayed in Figure 7.2 and Table 7.2. For the synthetic policy set, this strategy yielded an average message throughput of: **4.37** which is a relative speed-up of: **1.48** over the baseline. The limiting parsing strategy performs slightly better than the real policy set, relative to the baseline. In this case, the large number of policies causes the baseline strategy to perform exceptionally poorly due to the fact that it is re-parsing the document for each of the policies. In the case of this strategy,

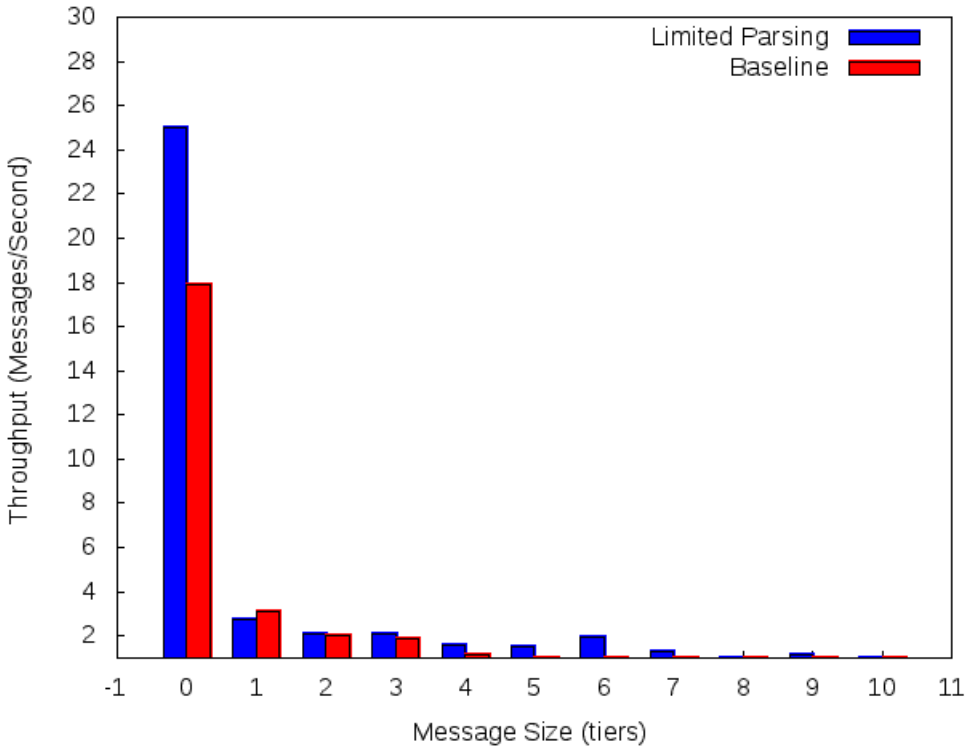

Figure 7.2: Message throughputs for the Limited Parsing Strategy using the synthetic policy set

the simple reduction of the parsing frequency allows for slightly better performance under the increased number of policies.

7.4 Parallelization

Synthetic Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	16.39	2.98	2.14	1.6	1.13	1.07	1.48	0.84	0.9	0.68	0.58
Speed-up	0.92	0.96	1.07	0.86	0.97	1.32	1.47	1.51	1.48	1.07	1.34
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 7.3: Compliant Throughput Results per Message Size, using the Parallelization Strategy

The results for the Parallelization strategy using the synthetic policy set are displayed in Figure

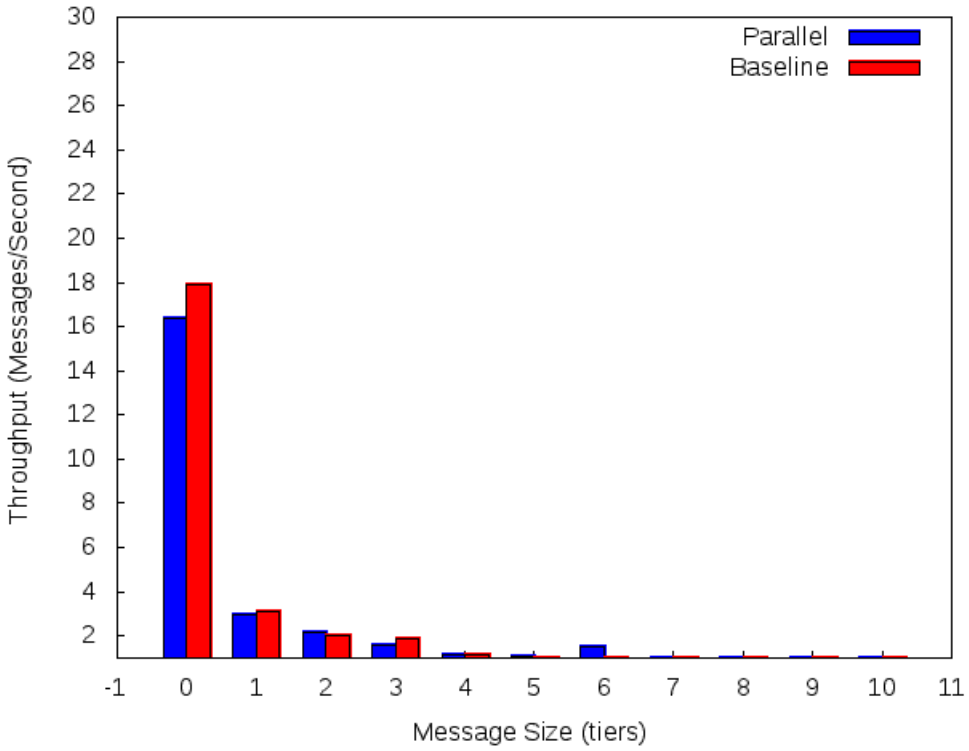

Figure 7.3: Message throughputs for the Parallelization strategy using the synthetic policy set

7.3 and Table 7.3. Under the synthetic policy set, the paralellization strategy yielded an average message throughput of: **3.1949** which is a relative speed-up of: **1.01** over the baseline. Our parallelization strategy is practically no better than the baseline under these conditions.

7.5 SPEX

Synthetic Policy Set											
Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Msg/Sec	23.81	4.05	2.53	2.11	1.62	1.41	1.23	0.77	0.81	0.7	0.61
Speed-up	1.33	1.31	1.26	1.14	1.39	1.74	1.23	1.37	1.32	1.11	1.42
% of Msgs	65%	8%	7.5%	5.6%	2.4%	1.2%	1.3%	1.4%	1.3%	1.2%	5.1%

Table 7.4: Compliant Throughput Results per Message Size, using the Stream Based Strategy

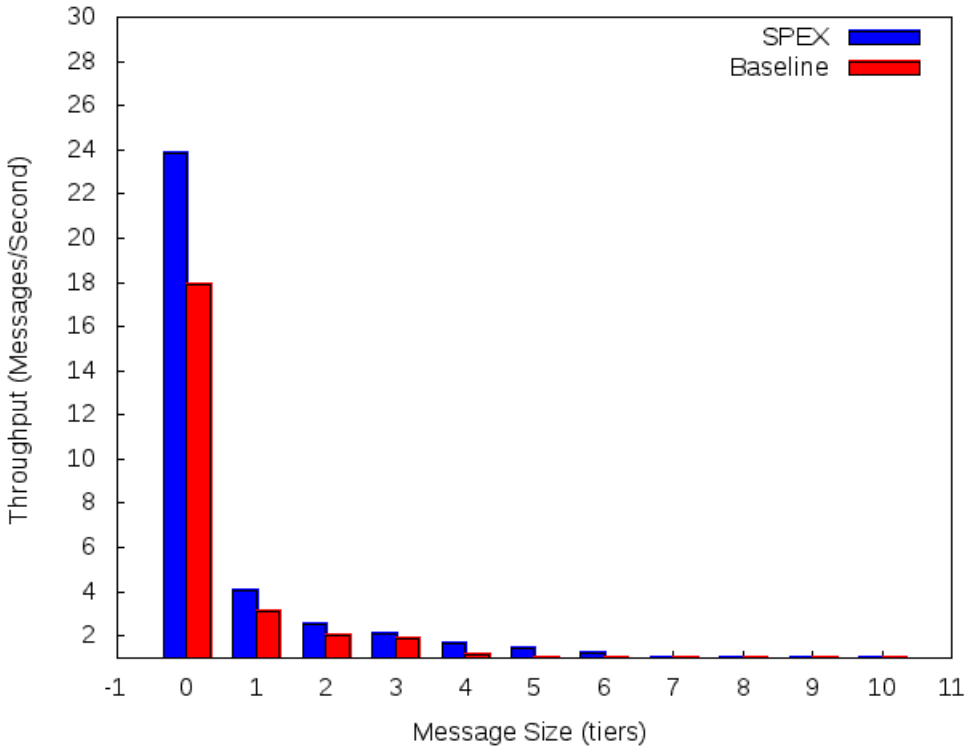

Figure 7.4: Message throughputs for SPEX using the synthetic policy set

The results for SPEX using the synthetic policy set are displayed in Figure 7.4 and Table 7.4. For the synthetic policy set, this strategy yielded an average throughput of: **3.76** which is a relative speed-up **1.27** over the baseline. This result is somewhat surprising as we expected that SPEX would scale much better than the others. SPEX is a "progressive" meaning that it will cease its search as soon as enough information is gained to make a decision about the query result. Thus, we expected SPEX to come to quick conclusions on most of the "smaller" queries (those that select only a few nodes).

7.6 Strategies Compared

The results displayed in Figure 7.5 are not as conclusive as the results for our real policy set. For the most part none of the strategies seem to present a significant advantage over the others.

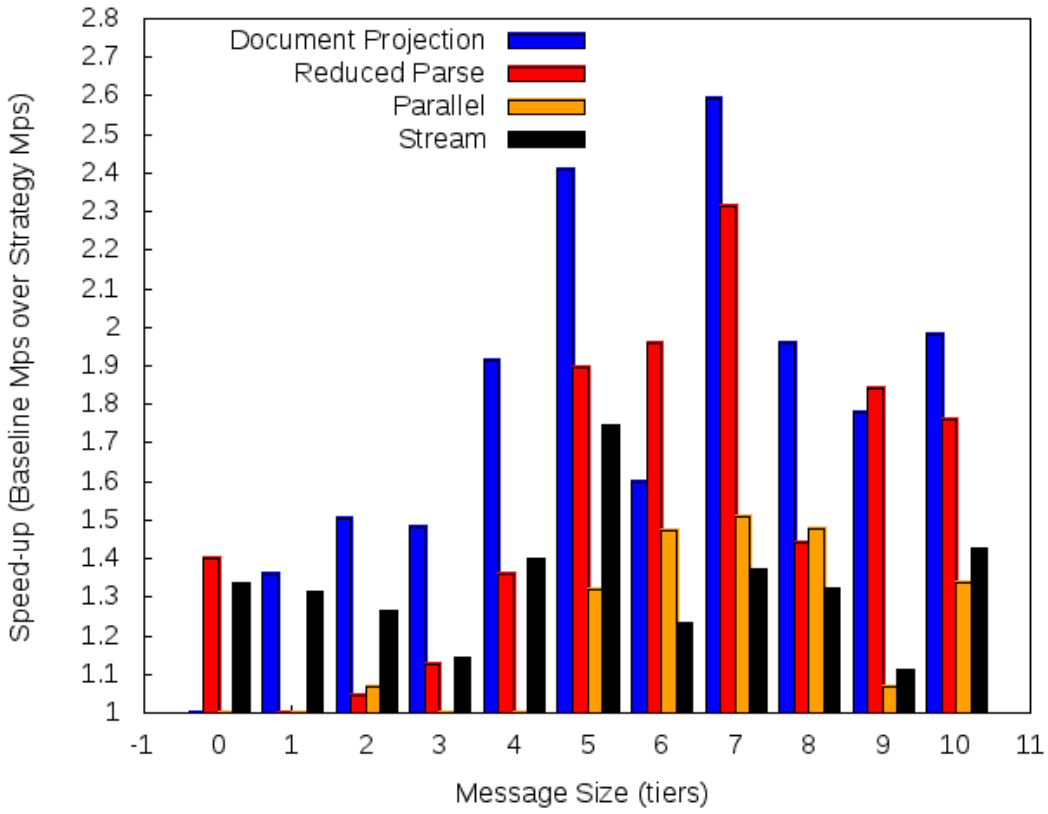

Figure 7.5: Message throughput speed-up (relative to baseline) for all strategies using synthetic policy set

The first observation is how much better the SPEX, limited parsing, and document projection strategies scale, compared to the baseline evaluation process. The performance gap between these strategies and the baseline has increased in this situation. With this said, the performance of each of these strategies under these conditions is still not where we would like it.

The next observation of interest in Figure 7.5 is the performance of SPEX compared to the other strategies. As was mentioned previously, we expected SPEX to scale well in this situation and as we can see it was out performed by both the document projection and limited parsing strategies. Although SPEX still provides a performance increase over the baseline, it did not provide the results that we expected. The improvement over the baseline in this situation is relatively minor an average speedup of 30% over the baseline.

Another observation of interest is that, for the most part, document projection provides a speed-up of over 30% over the baseline for almost all message sizes (except the smallest, which we expected), and an average speedup of 60% over the baseline. We can say with some certainty that under the synthetic policy set, the document projection strategy seems to provide the best improvement over the baseline.

Although we cannot take these results to be a clear indicator of the expected performance under a worst case policy set, they do present a possible problem: as the policy set increases in size, all our algorithms begin to under-perform. This leads us to question whether we are in need of an algorithmic improvement or a better way of representing our policies. The ideas surrounding this will be left as an area of future exploration.

Chapter 8

Implications and Future Work

In this chapter we will discuss some ideas for improving policy evaluation within our continuous compliance assurance mechanism. These strategies are directed toward reducing the effect that policy set size might have on the evaluation time.

8.1 Policy Optimization

We observed that algorithmic improvements to query evaluation alone may not be enough to solve our problem. We have suggested along the way different strategies for possibly improving performance in the future. Of these strategies, many of them involve the idea of combining or collapsing the queries (policies) themselves. This would indicate that the solution likely resides in reducing the load being placed on the various evaluation strategies, rather than improving the evaluation itself.

We have already shown the situations in which certain strategies work best. We could leverage this performance by ensuring that the minimal policy set is always presented to the processor. Using query rewriting [18] [40] and minimization strategies [14] [23], we can structure the policies in some kind of canonical form, and ensure that each policy query does not contain redundant evaluations because by definition, minimized queries do not contain any redundant steps [14] [23].

By ensuring the policy queries are all in a canonical form, we may check for containment of similar paths, using a strategy such as [46]. This would allow us to avoid the possibility of redundant evaluation. According to Schwentick [46], these ideas are all fundamental to the problem of query optimization.

Over the past several years most of the study of improving the performance of XPath query processing has been directed toward the evaluation of the queries (which the main reason that we directed our attention to these types of improvements). More recently, the discussion has begun to move toward alternative strategies for overcoming the performance issues related to XPath query processing, by lending more attention toward optimizing the queries themselves [6]. Of the research currently being conducted in this area, much of it pertains to the efficient evaluation of queries over large databases of XML data. This often implies the evaluation of single and disjoint query over a large set of XML encoded data. This differs from our problem in that we are very much concerned with effectively evaluating multiple queries over an XML encoded document (which are likely to be much smaller than an XML database). Although, these problems differ, there are enough similarities for us to borrow many of the ideas that have come out of this work, and move forward from there. For example, like our situation, the database query evaluation problem must evaluate the query on the fly. The main focus of future work in this area (for our concerns) should be directed toward the determination of effective algorithms for minimizing and combining the set of XPath queries that are to be evaluated. In the coming sections, we will lay out some of the areas that need to be studied, laying out specifically the areas that are of most importance to our problem.

8.1.1 XML Structural Knowledge

XML is a well structured and a relatively complex data model. Che et al [6] argue that these structural properties of XML provide opportunities for query optimization. These extend from the semantic knowledge that is implicitly provided by the structure of XML. For example, if we know

from an XSD the structure of an XML document, we can assert that certain elements only appear as children of one particular element. This type of knowledge could be exploited to better optimize queries, unfortunately, according to Che et al [6], these types of inferences are rarely used.

As helpful as the semantic knowledge provided by XML can be for query optimization, it can just as detrimental. According to Che et al [6], the structure of XML adds extra levels of complexity to the data model. This complexity in the data model increases the optimization search space leading to the need for more complex evaluation strategies (in runtime space).

CCA's requirement for each of documents to be in a known and valid form implies that each document will have an associated XSD. Studying methods for leveraging this structural knowledge could prove to be helpful in optimizing the policies.

8.1.2 Canonical Formulization

For purposes of examining and comparing the policies, it is important that they be in some kind of canonical form. One helpful aspect of our problem is that we control the encoding of the policies in XPath. For example the auditor overseeing the data in the enclave determines that a policy must exist limiting disclosure of data item X to user Y. He or she would provide some high level description of this policy, from which the system would then generate all appropriate XPath encoded policies. This control would allow us to formulate the policies in any way that we felt would be most appropriate. This certainly adds an helpful simplification to the problem, but there is still much studying that must be done to determine the appropriate strategies for formulating these policies.

Much research has already been conducted in the field of XPath minimization (essentially formulating an expression into some kind of canonical form). Although, this research has been directed toward the reduction of an existing path expression into this minimal form, it is not at all unreasonable to adopt these strategies for our problem of XPath policy generation. Below is a listing of some rewriting strategies that should be further explored:

Remove Excepts Groppe et al [18], discuss rules for eliminating *intersect* and *except* operators from any XPath statement. They claim evaluation improvements of up 350 times that of the original XPath, as a result of their proposed minimizations.

No Backwards Axes Olteanu et al [40] discuss techniques for representing any XPath expression containing backwards axes as an expression containing only forward axes. The authors of this paper were also the ones behind the SPEX XPath processor used in our experiments. XPath expressions containing only forward axis are more easily evaluated over streaming XML, and thus this was a requirement of that particular XPath processor.

Ensure only child/descendant Axes As far as our knowledge, there is no work outlining rules for reducing queries to only *child* and *descendent* axes. According to Shwentick [46], the reduction of queries into this subset of XPath will ensure polynomial complexity bound on minimimization and containment detection.

Minimize or Eliminate Wildcards From the discussion of complexity of various XPath subsets in [46]. The removal of wildcards would place the complexity of minimization and containment in polynomial space.

Beyond rewriting the policies, we must also ensure that each is in a minimal form. A good survey of the problem of minimization is discussed in [14]. This problem is closely related to the issue of query containment [14]. It can be treated as a special case of the query containment problem, dealing only with conditions located in one query. If, for example, one condition in a query implies another, then the second condition does not need to be evaluated.

8.1.3 Query Containment

Query containment would be useful to us as it might significantly reduce the need to evaluate redundant policies. Imagine two policies, p and q, it might be the case that if some document

satisfies p, then it also satisfies q [46]. This being the case, there would be no need to evaluate q as p has already been satisfied. This type of inference will be helpful in our situation to reduce the effects of an increasing policy set.

To fully understand query containment, we must first briefly review XPath and what a path expression represents. Very simply put, an XPath expression represents a set of nodes in an XML document that match the properties specified in the expression. With this understanding, we can now explore the three different types of XPath query containment. Given two queries q and p, and an XML tree, t:

Binary Relation Containment p is contained (through binary relation containment) by q if for every t, the relation specified in p returns a subset of the results returned by the relation contained by q. This simply implies that the expression p evaluates to a subset of q, and is evaluated from some arbitrary context.

Absolute Expression p is contained (through absolute expression containment) by q if for every t, relative to the root, the relation specified in p returns a subset of the results returned by the relation contained by q. This type of containment is only concerned with subsets whose context is the root of the document.

Boolean Containment This is the most general case, and considers only whether one query is at all contained in another. As either binary or absolute containment.

All of these containment notions are related, the use of each depends on which subset of XPath we are operating within.

Using the rewriting and minimization strategies discussed above, we can ensure that our XPath policies are in some canonical form. This is necessary in order to statically check for containment. If the queries were not expressed using the same minimal subset of XPath, we could not be certain of what the intended results were to be for one query relative to the other, and therefore could not statically check for containment

Essential to the detection efficiency of containment is the selection of an appropriate XPath subset. Discussed earlier, was the notion that XML provides a large search space to operate within (this includes the XPath data model). This becomes a problem when evaluation time efficiency is a chief concern. Shwentick [46] provides a good survey of various XPath notation sets and their related time complexity. From these, we should chose a notation set that provides us all of the expressiveness that we require while minimizing the time complexity.

8.2 Multiple Strategies

A very prevalent observation from our experiments is that no strategy performed well in every situation. SPEX performed well under the real policy set, but completely fell apart when subjected to the synthetic policy load. Document projection appeared to show plenty of promise in handling the larger synthetic policy sets, but performed poorly when operating on smaller documents. With this in mind, we feel that making use of multiple strategies (dynamically choosing, at runtime) could prove to be helpful, but the issue is: how do we determine which strategy to use? As we see it, the problem can be broken down as: Given a policy set, a document size, and a document structure, can we choose the appropriate strategy. Early signs indicate that each of these strategies exhibit certain characteristics that could be inferred from these three items.

Chapter 9

Conclusions

0

In this chapter we will present a summary of this paper, and discuss our observations. We will present the lessons we have learned from our experiments and tie these observations to plans for future work.

9.1 Summary

In this paper we have introduced ideas of the trusted enclave and continuous compliance assurance (CCA) as a solution for enabling trust in a sharing environment. We demonstrated that there is a need for solution like CCA in domains, including: commercial supply chain, government, and health care. We discussed the importance of performance of a system in building and maintaining trust. If the system does not perform adequately, it will likely be disabled in some form, thus removing any ability to enable trust. We presented other approaches to enabling trust in an information sharing environment and contrasted that work to that of our enclave and CCA solution. Through this discussion, we demonstrated that our solution stands alone in comparison to the other strategies.

We also discussed issues related to our trusted enclave and CCA solution. These problems are

related to the compliance verification mechanism that is employed in CCA. Specifically, the problem extends from the evaluation of multiple policies (XPath expressions) over a single document. This type of evaluation adds a non-trivial nature to our problem. In order to over come this we would explore multiple evaluation strategies.

To observe our systems performance and assess the improvements made by changes to the evaluation strategy of XPath, we put together a test bed. This test bed incorporated a prototype CCA mechanism, a model enclave database, a workload generator, and analysis tools. We discussed the process of generating the model enclave, describing all of the types of policies that were used and how they related to the data in the enclave. Our methods for creating a realistic workload were described, including the scenario we were modeling and the messages that were involved in that scenario.

Also discussed, was the policy sets that we used in our testing: real and synthetic. The real policy sets were used as an indicator of how we would expect our system to perform in a real world situation, while the synthetic policy set was designed to test the bounds of the evaluation strategies (i.e. how would they perform under a worst case policy load). We admit that the synthetic policy set certainly may not best represent the best example of a worst case. Nevertheless we have learned a great deal from the process of assembling the policies and the observations made from applying them.

We discussed our baseline results and suggested areas that needed improvement, particularly identifying the compliance module as the area of execution that required a further exploration. We pointed out that the observed problems reside within the evaluation of XPath (the compliance module). We suggested approaches for reducing the cost of XPath evaluation.

These approaches included: document projection, limiting the amount of necessary parsing, parallelizing the evaluation, and making use of a streaming XPath evaluation process. We integrated each of these into our prototype and observed the results. We were concerned with the effect that these alternative evaluation strategies had on the overall system throughput, but were

also interested in observing the strategies from the perspective of message size.

9.2 What We Learned

The results we gathered, from our test bed allowed us to draw some conclusions about the performance of our proposed evaluation strategies.

We expected that document projection would provide improvement under the evaluation of larger documents. For the real policy set, this was true as it performed better than parallelization and limited parsing for larger documents, but only slightly better. Under the synthetic policy set we could better observe the improvement provided by document projection as it performed better than all of the policies for larger documents. Our conclusion here is that document projection does provide improvements for the evaluation of larger documents, but only under heavier policy loads. Also tied to these observations is the universally poor performance of document projection for smaller documents. Thus, document projection should not be used to evaluate smaller documents as it will always provide poor performance.

We expected SPEX to provide a boost in performance as it was utilizing a less expensive (in terms of performance) parsing technique (SAX). Under the real policy set, our observations supported this expectation. SPEX performed the best under all document sizes, averaging a throughput improvement of approximately 20% over the baseline. We observed under the synthetic policy set that SPEX was out performed by document projection. This can be explained by the fact that document projection is more tightly tied to the policies. Meaning that document projection can adjust its environment to better fit the needs of the policy, where as SPEX is ignorant to the policy and performs the evaluation in the same manor for all policies. This likely leads to a larger search space for SPEX and a smaller search space for document projection. From the observations gained through the real policy set, we conclude that SPEX provides the best solution, but this comes with a need for future exploration. In order for SPEX to be viable, we must concentrate on techniques

for reducing the policies. As the experiments with the synthetic policies have shown, under heavier policy loads, SPEX tends to fall apart, but if we are able to reduce the number of policies by eliminating redundancy, we could ensure fewer searches over the document.

The process of building the test bed for this study has been a wonderful learning process both about the construction of a test bed and of the studying of a system such as ours. Throughout its, construction, we were faced with several problems, many of which are still being explored. The first issue was related to creating a realistic scenario. This included creating an appropriate policy set and a representative document set (including distributions of the document sizes). Although we feel that we have put together an excellent base from which to work, there is still plenty of room for improvement that must be done to move our test bed ever closer to a more real and representative model.

One key area of improvement is in expanding our scenarios beyond what we have. Work is currently under way in adding two new scenarios. For each of these scenarios, we will have to determine first if our message size distributions are truly representative, and second what is an accurate policy set size? Though the work discussed here does not provide a definitive answer to these questions, it has done enough to bring these questions to our attention.

Currently, there is hardly any discussion about the performance of trust enabling systems, and even less demonstrating the viability of particular solutions. Through our work, we have discussed concerns with performance of our CCA system, we have suggested and tested possible improvements, and have laid out a plan to further improve the performance. Our work should serve as a launching point for plenty of future exploration into the related performance problems with continuous compliance assurance, and further the discussion of the study of these types of systems.

Bibliography

[1] ACORD. Acord xml standard. http://www.acord.org/Standards/propertyxml.aspx, October 2009.

[2] Rakesh Agrawal, Tyrone Grandison, Christopher Johnson, and Jerry Kiernan. Enabling the 21st century health care information technology revolution. *Commun. ACM*, 50(2):34–42, 2007.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 143–154. VLDB Endowment, 2002.

[4] George Annas. A new era of medical-record privacy. *The New England Journal of Medicine*, 348(15):1486–1490, 2003.

[5] Bharat Bhargava. Innovative ideas in privacy research (keynote talk). In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 677–681, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Dunren Che, Karl Aberer, and Tamer Özsu. Query optimization in xml structured-document databases. *The VLDB Journal*, 15(3):263–289, 2006.

[7] James Clark and Steve DeRose. Xml path language version 1.0. http://www.w3.org/TR/xpath/, March 2010.

[8] Codehaus. Stax. http://stax.codehaus.org/Home, May 2010.

[9] President's Information Technology Advisory Committee. *Revolutionizing Health Care Through Information Technology*. President's Information Technology Advisory Committee, 2004.

[10] United States National Security Council. *National Strategy for Information Sharing*. National Security Council, 2007.

[11] Joseph D'Alessandro, Cynthia Tanner, Bonnie Morris, and Tim Menzies. Is continuous compliance assurance possible? In *Sixth International Conference on Information Technology: New Generations*, 2009.

[12] Debreceny, Gray, Ng, Lee, and Yau. Embedded audit modules in enterprise resource planning systems: Implementation and functionality. *Journal of Information Systems*, 19(2):7–28, 2005.

[13] United States Information Sharing Environment. *Guidelines to Ensure that the Information Privacy and Other Legal Rights of Americans are Protected in the Development and Use of the Information Sharing Environment*. United States Information Sharing Environment, 2006.

[14] S. Flesca, F. Furfaro, and E. Masciari. On the minimization of xpath queries. *J. ACM*, 55(1):1–46, 2008.

[15] Lawerence Gottlieb, Elliot Stone, Diane Stone, Lynne Dunbrack, and John Calladine. Regultory policy barriers to effective clinical data exchange: Lessons learned from medsinfo-ed. *Health Affairs*, 24(5):1197–1204, 2005.

[16] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing xpath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[17] United States Government. U.s. privacy act of 1974. `http://www.ssa.gov/privacyact.htm`, April 2010.

[18] Sven Groppe, Stefan Bottcher, and Jinghua Groppe. Xpath query simplification with regard to the elimination of intersect and except operators. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*, 2006.

[19] FEA Working Group and Federal CIO Council. E-gov enterprise architecture guidance (common reference model). In *FEA Working Group*, 2002.

[20] Lance J. Hoffman, Kim Lawson-Jenkins, and Jeremy Blum. Trust beyond security: an expanded trust model. *Commun. ACM*, 49(7):94–101, 2006.

[21] XBRL International. Xbrl xml standard. `http://www.xbrl.org/Home`, October 2009.

[22] Grandison Johnson. Compliance with data protection laws using hippocratic database active enforcement and auditing. *IBM Systems Journal*, 46(2), 2007.

[23] Benny Kimelfeld and Yehoshua Sagiv. Revisiting redundancy and minimization in an xpath fragment. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 61–72, New York, NY, USA, 2008. ACM.

[24] Tak Cheung (Brian) Lam, Jianxun Jason Ding, and Jyh-Charn Liu. Xml document parsing: Operational and performance characteristics. *Computer*, 41(9):30–37, 2008.

[25] Doug Lea. *Concurrent Programming in Java, Second Edition: Design Principles and Patterns*. Addison-Wesley, 2000.

[26] Hau Lee and Seungjin Whang. Information sharing in a supply chain. *International Journal of Manufacturing Technology and Management*, 1(1):79–93, 2000.

[27] Peng Liu and Amit Chetal. Trust-based secure information sharing between federal government agencies: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 56(3):283–298, 2005.

[28] Albert J. Marcella, William J. Sampias, and Larry Stone. *Electronic Commerce: Control Issues for Securing Virtual Enterprises*. Institute of Internal Auditors, Incorporated, 1998.

[29] Amélie Marian and Jérôme Siméon. Projecting xml documents. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 213–224. VLDB Endowment, 2003.

[30] Deven McGraw, James Dempsey, Leslie Harris, and Janlori Goldman. Privacy as an enabler, not an impediment: Building trust into health information exchange. *Health Affairs*, 28(02):416–427, 2009.

[31] David Megginson. Sax. `http://www.saxproject.org/`, May 2010.

[32] Bonnie Morris, Geoffrey Shaw, Cynthia Tanner, and George Trapp. Continuous compliance assurance for trusted information sharing: A research framework. VIACK Corp. Grant, Septmber 2007.

[33] Bonnie Morris, Cynthia Tanner, and Joseph DAlessandro. Enabling trust through continuous compliance assurance. In *Seventh International Conference on Information Technology: New Generations*, 2010.

[34] Avinandan Mukherjee and John McGinnis. E-healthcare: an analysis of key themes in research. *International Journal of Pharmaceutical and Healthcare*, 1(4):349–363, 2007.

[35] Matthias Nicola and Jasmi John. Xml parsing: a threat to database performance. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 175–178, New York, NY, USA, 2003. ACM.

[36] Office of Civil Rights. Summary of the hipaa privacy rule. Technical report, 2003.

[37] US Department of Justice Office of Justice Programs. National information exchange (niem) user guide. Technical report, 2008.

[38] United States Government Accountability Office. Information sharing: The federal government needs to establish policies and processes for sharing terrorism-related and sensitive but unclassified information. Technical report, 2006.

[39] Dan Olteanu. Spex: Streamed and progressive evaluation of xpath. *IEEE Trans. on Knowl. and Data Eng.*, 19(7):934–949, 2007.

[40] Dan Olteanu, Holger Meuss, Tim Furche, and Francois Bry. Xpath: Looking forward. *XML-Based Data Management and Multimedia Engineering*, 2490/2002:892–896, 2002.

[41] Information Sharing Environment Program Manager. Information sharing environment enterprise architecture framework. Technical report, 2008.

[42] VTD Project. Vtd. http://vtd-xml.sourceforge.net/persistence.html, May 2010.

[43] Pauline Ratnasingham and Kuldeep Kumar. Trading partner trust in electronic commerce participation. In *ICIS '00: Proceedings of the twenty first international conference on Information systems*, pages 544–552, Atlanta, GA, USA, 2000. Association for Information Systems.

[44] Andrew Rohm and George Milne. Just what the doctor ordered: The role of information sensitivity and trust in reducing medical information privacy concern. *Journal of Business Research*, 57(9):1000–1011, 2004.

[45] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. Xmark: a benchmark for xml data management. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 974–985. VLDB Endowment, 2002.

[46] Thomas Schwentick. Xpath query containment. *SIGMOD Rec.*, 33(1):101–109, 2004.

[47] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

[48] Edward Shortliffe. Strategic action in health information technology: Why the obvious has taken so long. *Health Affairs*, 24(5):1222–1223, 2005.

[49] Jatinder Singh, Jean Bacon, and Ken Moody. Dynamic trust domains for secure, private, technology-assisted living. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 27–34, Washington, DC, USA, 2007. IEEE Computer Society.

[50] Jatinder Singh, Luis Vargas, Jean Bacon, and Ken Moody. Policy-based information sharing in publish/subscribe middleware. In *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pages 137–144, Washington, DC, USA, 2008. IEEE Computer Society.

[51] Nicolas P. Terry and Leslie P. Francis. Ensuring the Privacy and Confidentiality of Electronic Health Records. *University of Illinois Law Review, Vol. 2007, pp. 681-735, 2007*, 2007.

[52] W3C. Dom. http://www.w3.org/DOM/, May 2010.