INTEGRITY AND INTERNAL CONTROL IN INFORMATION SYSTEMS VI

Edited by
Sushil Jajodia
Leon Strous

IFIP

KLUWER ACADEMIC PUBLISHERS

INTEGRITY AND INTERNAL CONTROL IN INFORMATION SYSTEMS VI

INTEGRITY AND INTERNAL CONTROL IN INFORMATION SYSTEMS VI

IFIP TC11 / WG11.5 Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS) 13–14 November 2003, Lausanne, Switzerland

Edited by

Sushil Jajodia
George Mason University
Fairfax, Virginia, USA

Leon Strous
De Nederlandsche Bank NV
Amsterdam, The Netherlands

KLUWER ACADEMIC PUBLISHERS
NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 1-4020-7901-X
Print ISBN: 1-4020-7900-1

CONTENTS

PREFACE

The development and integration of integrity and internal control mechanisms into information system infrastructures is a challenge for researchers, IT personnel and auditors. Since its beginning in 1997, the IICIS international working conference has focused on the following questions:

- what precisely do business managers need in order to have confidence in the integrity of their information systems and their data and what are the challenges IT industry is facing in ensuring this integrity;
- what are the status and directions of research and development in the area of integrity and internal control;
- where are the gaps between business needs on the one hand and research / development on the other; what needs to be done to bridge these gaps.

This sixth volume of IICIS papers, like the previous ones, contains interesting and valuable contributions to finding the answers to the above questions. We want to recommend this book to security specialists, IT auditors and researchers who want to learn more about the business concerns related to integrity. Those same security specialists, IT auditors and researchers will also value this book for the papers presenting research into new techniques and methods for obtaining the desired level of integrity.

It is the hope of all who contributed to IICIS 2003 that these proceedings will inspire readers to join the organizers for the next conference on integrity and internal control in information systems. You are invited to take the opportunity to contribute to next year's debate with colleagues and submit a paper or attend the working conference. Check the websites given below regularly for the latest information.

We thank all those who have helped to develop these proceedings and the conference. First of all, we thank all the authors who submitted papers as well as the keynote and invited speakers, and those who presented papers and participated in the panel. Finally, we would like to thank all conference participants, IFIP and the sponsors and supporters of this conference.

January 2004

Sushil Jajodia
Leon Strous

Websites:

IFIP TC-11 Working group 11.5 IICIS 2004
http://www.cs.colostate.edu/~iicis04/

IFIP TC-11 Working group 11.5
http://csis.gmu.edu/faculty/Tc11_5.html

IFIP TC-11
http://www.ifip.tu-graz.ac.at/TC11

IFIP
http://www.ifip.org

Still available:

IICIS 2002:	Integrity and internal control in information systems V ed. Michael Gertz ISBN 1-4020-7473-5
IICIS 2001:	Integrity, internal control and security in information systems: Connecting governance and technology ed. Michael Gertz, Erik Guldentops, Leon Strous ISBN 1-4020-7005-5
IICIS 1999:	Integrity and internal control in information systems: Strategic views on the need for control ed. Margaret E. van Biene-Hershey, Leon Strous ISBN 0-7923-7821-0
IICIS 1998:	Integrity and internal control in information systems ed. Sushil Jajodia, William List, Graeme McGregor, Leon Strous ISBN 0-412-84770-1
IICIS 1997:	Integrity and internal control in information systems: Volume 1, Increasing the confidence in information systems ed. Sushil Jajodia, William List, Graeme McGregor, Leon Strous ISBN 0-412-82600-3

ACKNOWLEDGEMENTS

Conference chairs:

Stefano Spaccapietra, Swiss Federal Institute of Techn., Lausanne, CH
Serge Vaudenay, Swiss Federal Institute of Technology, Lausanne, CH

Programme Committee:

Co-Chairs:

Sushil Jajodia, George Mason University, USA
Leon Strous, De Nederlandsche Bank, The Netherlands

Members/reviewers:

David Basin, ETH Zürich, Switzerland
Sabrina de Capitani di Vimercati, University of Milan, Italy
Michael Gertz, University of California at Davis, USA
Erik Guldentops, University of Antwerp, Belgium
Klaus Kursawe, IBM, Switzerland
Detlef Kraus, SRC, Germany
William List, William List & Co., UK
Refik Molva, Eurecom, France
David Naccache, GEMPLUS, France
Philippe Oechslin, EPF Lausanne, Switzerland
Indrakshi Ray, Colorado State University, USA
Arnie Rosenthal, The MITRE Corporation, USA
Adrian Spalka, University of Bonn, Germany
Bhavani Thuraisingham, NSF, USA

Organizing Committee

Christelle Vangenot (chair)
Marlyse Taric (secretariat)
Swiss Federal Institute of Technology, Lausanne, Switzerland

REMOTE INTEGRITY CHECKING

How to Trust Files Stored on Untrusted Servers

Yves Deswarte*, Jean-Jacques Quisquater**, Ayda Saïdane*
** LAAS-CNRS, France*
*** Université Catholique de Louvain, Belgium*

Abstract: This paper analyzes the problem of checking the integrity of files stored on remote servers. Since servers are prone to successful attacks by malicious hackers, the result of simple integrity checks run on the servers cannot be trusted. Conversely, downloading the files from the server to the verifying host is impractical. Two solutions are proposed, based on challenge-response protocols.

Key words: file integrity checking, intrusion detection, challenge-response protocols.

1. INTRODUCTION

The integrity of files stored on servers is crucial for many applications running on the Internet. A recent example has shown that Trojan Horses can be largely distributed in security critical software, due to a successful attack on one server [CERT 2002]. This kind of damage could have been hindered if this particular software distribution was secured by digital signatures (as is currently done for some other software distribution on the Internet). In other cases, damage may vary from web page defacing to circulation of false information, maliciously modified execution of remote services or diverse frauds in electronic commerce. The detection of such wrong behavior can be more difficult than for software distribution, since in most cases it is not possible to just check a signature on a content. The current state of the Internet security is such that most servers are vulnerable to dedicated attacks and in most cases the detection of such attacks happens several hours, days or weeks after the real attacks have occurred (one or two days in the case cited above). Indeed, new vulnerabilities are discovered frequently on most commercial operating systems and applications, and current intrusion detection systems do not detect or misidentify a too large proportion of attacks, in particular when the attacks are new and slow [Green et al. 1999].

It is thus of tremendous importance for system administrators to check frequently that no critical file has been modified on the servers they manage. The classical way to do so is to reboot the server from a secure storage (a CDROM for instance) and then to launch locally (from the secure storage) a program to compute cryptographic checksums (using one-way hash functions) on the critical files and compare the results with reference checksums stored on a secure storage. Tripwire[1] is a well-known example of such programs. It is important to reboot the system from a secure storage since a malicious hacker could have installed and launched a program on the server which, for instance, could modify the disk drive handler to return the original file content to the integrity checking program instead of a modified version actually stored on the disk. Some viruses and Trojan horses are using such stealth techniques. This is also why such a program has to be run locally, after a secure reboot, before any insecure application (which could have been modified by a hacker) is run.

Such a technique is impractical in most Internet server systems:

- This task requires a lot of time: halting operational applications; halting the system in a safe way; rebooting form a CDROM; running the integrity checking program on all critical files; restarting the operational applications. This is hardly compatible with 24/7 operation required from most Internet servers and, anyway, would be too costly if run frequently.
- Competent system administrators are a scarce resource, and thus most servers are managed remotely: it would be impractical for the administrators to go to each server to execute this routine task in a secure way.

Running remotely an integrity check program is inefficient, since it is impossible to be sure if the program that is run is the original one and not a fake, if the reference checksums are the correct ones[2], and if the operating system has not been modified for some stealth operation.

Conversely, it is impractical for the administrator to download all critical files to his local host, to compute locally the checksums and then to compare the results with reference checksums: this would cause too much overhead on the network and on the administrator host, as soon as the numbers of files

[1] Tripwire® is a registered trademark of Tripwire, Inc.

[2] The reference checksums can be signed, but then the integrity of the signature verification key must be also checked.

and of servers are high, the mean file length is large, and this task has to be run frequently.

This paper proposes two solutions to this problem. The first one, described in Section 2, is a challenge-response protocol using conventional methods, which lead to some engineering trade-offs. Section 3 presents another protocol, based on a modular exponentiation cryptographic technique, to solve this problem in a more elegant way, but at the price of more complex computations. These solutions are then compared to alternative solutions and related work.

2. A CONVENTIONAL CHALLENGE-RESPONSE PROTOCOL

2.1 General approach

In this solution, the administrator's host (called hereafter the *verifier*) sends periodically a request to the server for it to compute a checksum on a file (specified as a request parameter) and return the result to the verifier. The verifier then compares the returned result with a locally-stored reference checksum for the same file.

A naïve implementation of this protocol would be inefficient: a malicious attacker could precompute the checksums on all files he intends to modify, store these checksums, and then modify the checksum computation program to retrieve the original checksum rather than compute it. The attacker can then modify any file, while remaining able to return the expected checksums when requested by the verifier. This protocol has to be modified to guarantee the freshness of the checksum computation.

This can be achieved by adding a *challenge C* in the request parameters. With this new protocol, the server has to compute a *response R* depending on the challenge. More precisely, instead of a checksum computed as the result of a one-way hash function on the content of the file, the response must be computed as the hash of the challenge concatenated with the file content:

$$R = \mathcal{H}(C|File) \tag{1}$$

Of course, the challenge must be difficult to guess for the attacker. In particular it must be changed at each request. But then the verifier cannot simply compare the response with a reference checksum. A solution would be to maintain a copy of all the original files on the verifier and run the same response computation on the verifier as on the server. But this is impractical if the numbers of files and of servers are high and the mean file length is large.

A better solution would be for the verifier to use two functions f and $\mathcal{H}'$, of which at least one of them is kept secret[3], such that $\mathcal{H}'$ is a one-way hash function, and f is such that:

$$f(C, \mathcal{H}'(File)) = \mathcal{H}(C|File) = R \qquad (2)$$

Unfortunately, we have not found (yet) functions f, $\mathcal{H}$ and $\mathcal{H}'$ satisfying this property.

To workaround this problem, a finite number N of random challenges can be generated off-line for each file to be checked, and the corresponding responses computed off-line too. The results are then stored on the verifier. At each integrity check period, one of the N challenges is sent to the server and the response is compared with the precomputed response stored on the verifier. In order to guarantee that a different challenge is issued at each request, and thus that an attacker cannot predict which will be the next challenge(s), the server has to be rebooted periodically[4] so that:

$$N > \textbf{(frequency of challenge-response protocol for the same file) / (reboot frequency)} \qquad (3)$$

A possible way for the attacker to circumvent this freshness checking could be to keep copies of both the original and the modified files. But this should be easy to detect, either by checking the integrity of the concerned directories and system tables, or by intrusion detection sensors tuned to detect this specific abnormal behavior.

The table of precomputed responses, stored on the verifier, is thus composed of N entries with, for each entry, a challenge and the expected corresponding response. It is possible to reduce the size of the table, by exploiting a technique presented in [Lamport 1981]: rather than generating a specific random number as a challenge for each entry, only one random number C_N is generated for a file, and each challenge C_i is computed as $\mathcal{H}(C_{i+1})$ for each i from (N-1) to 1 (by step of -1). The precomputed response table contains only the N expected responses, the last challenge C_N and the

3 At least $\mathcal{H}'$ or f must be kept secret, because if both were public, it would be easy for the attacker to precompute all needed $\mathcal{H}'(File)$ and then dynamically compute the expected response $f(C, \mathcal{H}'(File))$.

4 Our current implementation exploits the fact that the servers are periodically rebooted (e.g., once a day), as a measure for software rejuvenation [Huang et al. 1995]: at reboot all files and programs are restored from a secure copy. This would erase any Trojan horse implemented previously by a malicious hacker, as well as any table of precomputed responses he could have built with previous challenges.

number *N*. The challenges are sent in the increasing order from C_1 to C_N, each challenge C_i being dynamically computed by the verifier:

$$C_i = \mathcal{H}^{(N-i)}(C_N), \text{ where } \mathcal{H}^k(X) = \mathcal{H}(\mathcal{H}^{k-1}(X)) \text{ and } \mathcal{H}^1(X) = \mathcal{H}(X) \qquad (4)$$

2.2 A practical example: integrity checking for a distributed web server

The above protocol has been implemented in a distributed, intrusion-tolerant web server [Valdes et al. 2002]. The system consists of a set of *verifiers* managing and monitoring a bank of web *servers* (see Figure 1).

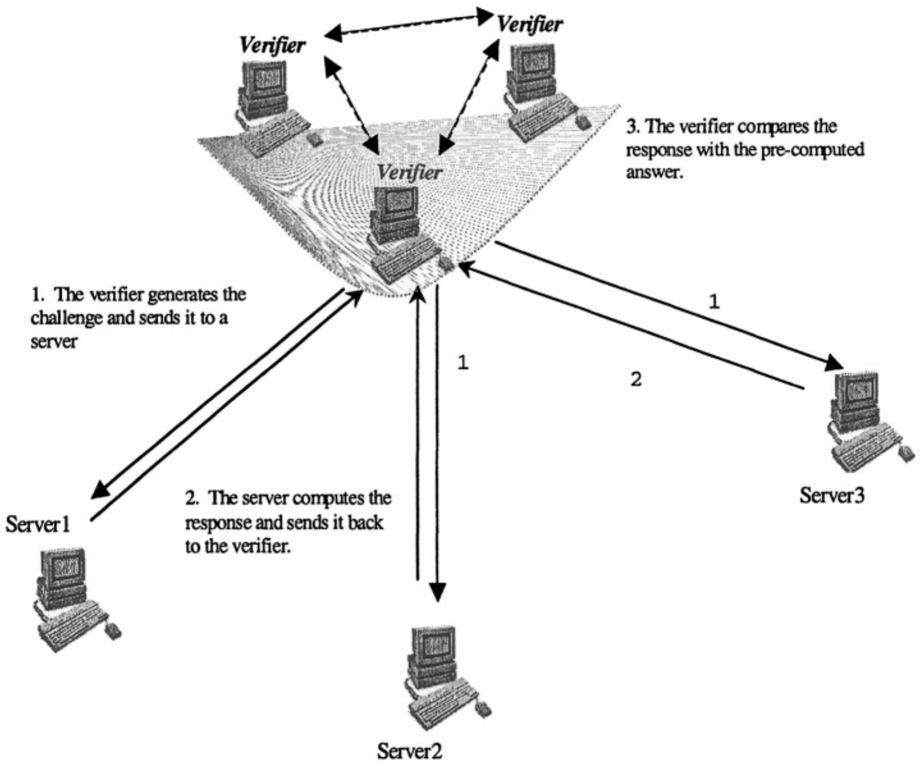

Verifier
Verifier
Verifier
3. The verifier compares the response with the pre-computed answer.
1. The verifier generates the challenge and sends it to a server
1
1
2
2. The server computes the response and sends it back to the verifier.
Server1
Server2
Server3

Figure 1. **Intrusion-tolerant web server architecture**

The challenge-response protocol is launched periodically by each verifier to check each server and each other verifier. This protocol is used for three purposes:

- As a *heart beat*: since this protocol is launched periodically on each proxy, if a proxy does not receive a challenge from another proxy for a time greater than the period, it can raise an alarm.
- To check the liveness of the servers and other proxies: if, after emitting a challenge, a proxy does not receive a response within some delay, it can raise an alarm.
- To check the integrity of some files, directories or tables located on remote servers and proxies.

The challenge-response protocol (CRP) was created to check the integrity of some files which are not modified during normal operation, such as sensitive system files (e.g., boot files and OS code files) or security-critical files (e.g., /etc/passwd on a UNIX system). The role of a web server is to produces HTML documents that could be static files (web pages) or dynamically produced (by CGI or ASP scripts) from static data (e.g., a read-only database). So CRP checks important HTML files, scripts and system and security files. It can also check the identity of sensitive active processes on the machine (e.g., httpd, security processes, etc.).

If the server is rebooted periodically for "software rejuvenation", there is a relation between the frequency of the CRP, the duration of one CRP exchange, the number of files to be checked, and the number of servers:

If the server is rebooted periodically for "software rejuvenation", there is a relation between the frequency of the CRP, the duration of one CRP exchange, the number of files to be checked, and the number of servers:

Considering n the number of the checked files, f the frequency of CRP ($f < 1/d$, d being the duration of one CRP exchange), and N the number of challenges per file, the relation is:

$$(n \times N \times \#\text{servers} / f) >= \text{time between reboots} \qquad (5)$$

There is a minimal value of $1/f$ that corresponds to the maximal value of the duration of an execution of CRP which is related to a request on the biggest checked file. The next table gives examples of the performance of CRP (when using MD5 as hash function, on Pentium III, 600 Mhz):

Table 1. Execution duration

File size	Duration of one execution of CRP
2,2 Mbytes	~ 0,66 s
13,5 Kbytes	~ 0,008 s

So if we consider that the biggest file to check is about 2 Mbytes, we must choose a value of $1/f > 0{,}66$ sec.

The following table gives the size of the precomputed response table for different values of n, $1/f$ and N, considering 4 servers and a reboot frequency of one per 24 hours.

Table 2. Frequency and size

n	$1/f$	N	Table size
50	5 s	87	~578 Kbytes
500	1 s	44	~2. 92 Mbytes
5000	0 ,7 s	7	~4.76 Mbytes

3. A SOLUTION BASED ON THE PROTOCOL OF DIFFIE-HELLMAN

We here describe a generic solution based on the well-known cryptographic protocol of Diffie-Hellman for key exchange [Diffie & Hellman 1976].

Let:

- m denotes the value of the file to be remotely verified on a server; it is an integer,
- N, a RSA modulus, with two prime factors or more, of length of around 1024 bits; this value is public, that is, considered as known by everybody including any malicious hacker with a lot of computing power,
- $phi(N) = L$ is secret and only known by the verifier; this function is the Euler function (if $N=pq$, then $L = (p-1)(q-1)$),
- a, an element between 2 and N-2, randomly chosen and public.

The protocol is the following one:

- the verifier stores the following precomputed value

$$a^m \bmod N = M, \tag{6}$$

this computation is easy thanks to the knowledge of L (the theorem of Euler allows us to replace the exponent m by the short value (m mod L) of length around 1024 bits, independent of the length of the protected file) and using, if necessary, the Chinese remainder theorem using the knowledge of the prime factors;
- the verifier chooses a random value r (the domain is the same as a) and sends the following value A as a challenge to the server with the file to be verified:

$$a^r \bmod N = A \tag{7}$$

- the server computes

$$A^m \bmod N = B \tag{8}$$

  and sends *B* to the verifier,
- the verifier computes in parallel

$$M^r \bmod N = C \tag{9}$$

  and verifies if $B = C$ thanks to the equation (10).

It is easy to see that the next equation (10) is correct by using the equations (6) and (9),

$$B = A^m \bmod N = a^{rm} \bmod N = M^r \bmod N = C \tag{10}$$

The security of the protocol follows from the security of the Diffie-Hellman protocol. The freshness of computation (8) on the whole file is guaranteed by the random selection of *r* by the server. Another paper will describe a lot of optimisations of this generic protocol.

4. DISCUSSION

In this section, we discuss how an attacker can defeat the proposed solutions and compare these solutions with conventional signature schemes.

To defeat our solutions, a hacker could save each file before modifying them. In that case, the hacked server would serve modified files to innocent users while still being able to compute fresh responses by using the saved file copies. But counter-measures can easily prevent and/or detect the saving of critical files:

- To prevent the hacker to copy the files, the server file system can be dimensioned in such a way that there would be no room for critical file copies.
- It is easy for a host-based intrusion detection system to discriminate the file copying from the normal server behavior. Moreover, the challenge-response protocol can be applied not only to data files, but also to directories, and even system tables, which stay mostly static on, dedicated servers. This would make the hacker's job much more complex.

An alternative, conventional way to check file integrity consists in signing every file by using a private owner's key, while each user would be able to check the file integrity by retrieving the corresponding public key through a public key infrastructure. But this solution, while well adapted for software distribution, presents many drawbacks for other applications:

- It is not directly applicable to web services: the replies to http requests are generally not a simple file content, and even when it is the case, the integrity checks would have to be integrated in browsers, with all the complexity associated with PKI management.
- A hacker could still replace the current copies of the files with obsolete copies with their original signatures.
- It would not solve the remote server management problem: the administrator would still have to retrieve the contents of all the files to check their integrity.

5. RELATED WORK

Tripwire® [Kim & Spafford 1993] is the most famous file integrity checker. It has been designed to monitor a set of files and directories for any changes according to signatures previously stored. Tripwire proposes a set of signature functions (MD5, MD4, MD2, Snefru and SHA). By default, MD5 and Snefru are stored and checked for each file but the selection-mask can be customized and any of these functions can be selected. The user must first generate, offline, a configuration file containing the list of the files to be monitored and constitute a database of signatures corresponding to this configuration file. When running, Tripwire scans periodically the file system for added or deleted files in the directories specified in the configuration file, and computes the signatures of the monitored files to compare them with the signatures stored in the database. As previously stated, this approach cannot be directly applied to check the integrity of files stored on a remote server: a corrupted server can store locally the signatures of monitored files before modifying them and, on request by the verifier, return these signatures instead of freshly computed ones.

The SOFFIC project (Secure On-the-Fly File Integrity Checker) is carried out at UFRGS (Brasil) [Serafim & Weber 2002]. Their goal is to create a framework for intercepting file system calls and checking the correctness of any request to access a file (read/write/execute). It should be able to deny access to illegally modified files and to protect itself against tampering. The SOFFIC is implemented as a patch to the Linux kernel so the majority of its components resides in the kernel. The idea is to generate off-line hashes for all the files to be checked (Hash List) and generate a list of non-checked files (Trusted File List). Each time a user process attempts to access a file (which is not in the Trusted File List), SOFFIC is activated to grant or deny the access: the access is denied if the hash stored in the Hash List differs from the hash computed on-the-fly. For writable files, a new hash is computed after modification.

Rather than modifying the kernel, it is possible to insert a middleware layer between the application software and the system kernel. This solution is more portable and easier to maintain than kernel modifications. Jones [Jones 1993] has proposed to implement this approach by Interposing Agents that control all or parts of the system interface. These agents can be used to implement monitors to check the correctness of system calls, in particular for accessing files. Fraser et al. propose a similar approach, based on software wrappers, to augment the security functionality of COTS software [Fraser et al. 1999]. Such wrappers could be used to protect kernel and critical system files from non-authorized changes.

All these approaches suffer the same problems as Tripwire: if a server is corrupted, its kernel can be modified or the middleware can be bypassed to remove all integrity checks.

6. CONCLUSION

In this paper, we proposed two methods for remote file integrity checking. The first one is based on a table of multiple challenges and precomputed responses for each file to be checked, the response being computed by the hash of the challenge concatenated with the content of the file. The freshness of the response computation by the server is guaranteed by the fact that a challenge is never reused before reboot of the server. With the second method, a single value is precomputed and stored on the verifier for each file to be checked, and the challenge is generated randomly. This second method requires more computation (modular exponentiation instead of a hash on the content of the file), but does not require a large table to be stored by the verifier. Many optimizations are possible on the second method to reduce the computation cost, and they will be presented in a future article, with performance comparison with the first method.

ACKNOWLEDGEMENTS

This research was initiated during collaboration between LAAS-CNRS and SRI International, partially supported by DARPA under contract number N66001-00-C-8058. The views herein are those of the authors and do not necessarily reflect the views of SRI International or DARPA. We are deeply grateful to our SRI International colleagues for the fruitful cooperation on this project, and for their help in improving this paper.

REFERENCES

[CERT 2002] CERT Advisory CA-2002-24, *Trojan Horse OpenSSH Distribution,* August 1, 2002.

[Diffie & Hellman 1976] W. Diffie and M.E. Hellman, "New Directions in Cryptography", *IEEE Transactions in Information Theory,* 22(1976), pp. 644-654.

[Fraser et al. 1999] T. Fraser, L. Badger and M. Feldman, "Hardening COTS Software With Generic Software Wrappers", Proc. of IEEE Symposium on Security and Privacy, 1999, pp. 2-16.

[Green et al. 1999] John Green, David Marchette, Stephen Northcutt, Bill Ralph, "Analysis Techniques for Detecting Coordinated Attacks and Probes", in *Proc. 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, USA, April 9-12, 1999, available at: <http://www.usenix.org/publications/library/proceedings/detection99/full_papers/green/green_html/>

[Huang et al. 1995] Y. Huang, C. Kintala, N. Kolettis, N.D. Fulton, "Software Rejuvenation: Analysis, Module and Applications", in *Proc. 25th IEEE Symposium on Fault Tolerant Computing Conference (FTCS-25),* Pasadena, CA, USA, June 1995, pp. 381-390.

[Jones 1993] M. Jones, "Interposition Agents: Transparently Interposing User Code at the System Interface", *Proc. 14th ACM Symp. on Operating Systems Principles,* Operating Systems Review, 27[5], December 1993, pp. 80-93.

[Kim & Spafford 1993] G.H. Kim and E.H. Spafford, *The Design and Implementation of Tripwire: a File System Integrity Checker,* Technical Report CSD-TR-93-071, Computer Science Dept, Purdue University, 1993.

[Lamport 1981] Leslie Lamport, "Password Authentication with Insecure Communication", *Comunications of the ACM,* 24(11), pp. 770-772, November 1981.

[Serafim & Weber 2002] Vinícius da Silveira Serafim and Raul Fernando Weber, *The SOFFIC Project,* < http://www.inf.ufrgs.br/~gseg/projetos/the_soffic_project.pdf>.

[Valdes et al. 2002] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saïdi, V. Stavridou and T. Uribe, "An Adaptative Intrusion-Tolerant Server Architecture", in *Proc. 10th International Workshop on Security Protocols,* Cambridge (UK), April 2002, to appear in Springer LNCS Series.

AUTOMATED CHECKING OF SAP SECURITY PERMISISONS

Sebastian Höhn
Software & Systems Engineering, Informatics, TU Munich, Germany
hoehn@in.tum.de

Jan Jürjens
Software & Systems Engineering, Informatics, TU Munich, Germany
juerjens@in.tum.de
http://www.jurjens.de/jan

Abstract: Configuring user security permissions in standard business applications (such as SAP systems) is difficult and error-prone. There are many examples of wrongly configured systems that are open to misuse by unauthorized parties.

To check permission files of a realistic size in a medium to large organization manually can be a daunting task which is often neglected.

We present research on construction of a tool which automatically checks the SAP configuration for security policy rules (such as separation of duty). The tool uses advanced methods of automated software engineering: The permissions are given as input in an XML format through an interface from the SAP system, the business application is described ba a diagram modeled with standard UML CASE (Computer-Aided Software Engineering) - tools and output as XMI, and our tool checks the permissions against the rules using an analyzer written in Prolog. Because of its modular architecture and its standardized interfaces, the tool can be easily adapted to check security constraints in other kinds of application software (such as firewall or other access control configurations).

Key words: integrity and internal control in financial systems, automated configuration review, security restraints

1. INTRODUCTION

The management and configuration of security-related resources in standard business applications is one of the most important tasks in mission-critical departments. There is not only the potential of a negative impact of public disclosure of confidential information and a resulting loss of faith among customers, but the threat of direct financial losses. Computer breaches are a real threat as a study by the Computer Security Institute shows:

- Ninety percent of the respondents detected computer security breaches within the last twelve months.
- Forty-four percent of them were willing and/or able to quantify their losses. These 223 firms reported $455,848,000 in financial losses [Pow02].

It is important to realize that the existence of security mechanisms itself does not provide any level of security unless they are properly configured. That this is actually the case is often non-trivial to see. One example is the rule of "separation-of-duty", meaning that a certain transaction should only be performed jointly among two distinct employees (for example, granting a large loan). Difficulties arise firstly from the inherent dynamics of permission assignment in real-life applications, for example due to temporary delegation of permissions (for example to vacation substitutes). Secondly, they arise from the sheer size of data that has to be analyzed (in the case of the large German bank, which motivated the current work, some 60,000 data entries). A manual analysis of the security-critical configurations through system administrators on a daily basis is thus practically impossible, which might result in security weaknesses in practice. This observation motivated the current research which has been initialized in cooperation with a large German bank and their security consulting partner. The goal was to develop a tool which can be used to automatically check security permissions against given rules in a specific application context (such as the separation of duty rule in the banking sector). The tool should in particular be applied to analyze the SAP security permissions of the bank at hand. The current paper reports on the design and development of this tool.

The permissions are given as input in an XML format through an interface from the SAP system, the business application is described by a diagram modeled with standard UML CASE-tools and output as XMI, and our tool checks the permissions against the rules using an analyzer written in Prolog. Because of it's modular architecture and it's standardized interfaces, the tool can be easily adapted to check constraints in other kinds of application software (such as firewalls or other access control configurations).

In the next section, we explain the task the tool is supposed to solve in more detail (including the format of permissions and rules to be supported), as well as the architecture of and the underlying concepts and important design decisions regarding the tool. Section 3 explains the actual analysis performed in the tool at the hand of some examples. We close with a discussion of related work and a conclusion.

2. AUTOMATED ANALYSIS OF SECURITY RULES

2.1 The Goals

As explained above, the correct configuration of secure business applications is a challenging task. So there is a need for automated tool-support. The tool presented here takes a detailed description of the relevant data structure of the business application, the business data, and some rules written by the administrator. Using this information, the tool checks whether the rules hold for the given configuration. If the rules do not hold this is written to the generated security-report. The tool should be able to accomplish the following specific tasks:

- It should read the configuration from the business application.
- It should automatically generate a report of possible weaknesses.
- It should provide a flexible configuration of the report's data.
- It should be easily configurable for different business applications.
- It should be able to check large-scale databases.
- The checking should be based on freely configurable rules.

Two other goals are particularly important to enable use of the tool beyond the specific task of checking SAP permissions of the SAP installation at hand: it has to be easy to integrate the tool with different business applications, and the rules that have to be checked need to be very flexible.

2.2 Architecture

The tool mainly consists of three parts. They store the information describing the relevant data structure of the business application, define the rules and evaluate the rules. An additional part is needed to import the data from the business application (such as the SAP system). As in our example this is the user data and some structural information about transactions.

The complete separation of the tool and the business application provides additional security and privacy: Firstly, by separating the tool from the business application, there is no way the tool could add any weaknesses to this security-critical part of the company's IT-system. The tool does not interact with the system at all, the only interaction the tool requires is data export. When the tool has completed it's task, there is a list of proposals for the administrator to review. So it is the administrator's task to decide whether he will follow the proposal or not. So there is no way the tool itself could add any weaknesses to the system.

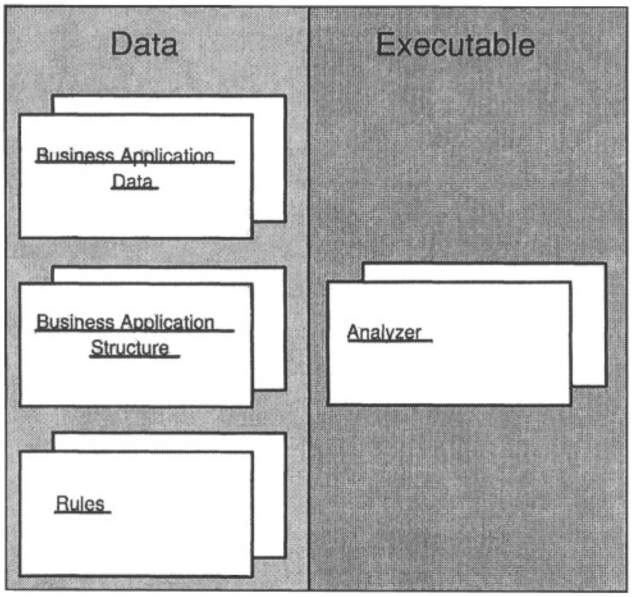


Figure 1. Overview of the Tool's Architecture

Secondly, this way it can be made sure that only the information needed for the analysis is exported to a foreign tool, which is important privacy matters. Both aspects should facilitate adoption of the tool.
The information itself is completely stored in XML. The business application's data has to be exported to XML files. In the specific application of the tool – the analysis of SAP security permissions – this task is outside the scope of the current paper.
The data structure of the business application is defined by UML class diagrams. Any case tool capable of saving XMI data can thus be used to do the modeling. The modeling in the current project will be done manually, because that will add some additional security (misconfiguration could result in a wrong model which will not be recognized as wrong then) and it is rather easily done. The complexity of the creation of this model depends one the size of the system. In SAP you can think about one diagram for each data table and the associations between these tables. Rules are stored in XML.

There is a graphical user interface in development which will help with the creation of rules.

2.3 The Business Application as a Model

Following conventions published by the Object Management Group (OMG) as "the classical four layer meta-model framework"[Obj02], software systems can be modeled particularly flexibly in an approach based on several layers of information (see Figure 2). Throughout the description of the analyzer there will be several types of information that fit into different layers on OMG's meta-model framework. In this framework there are UML models on layer 1 (M1) and application data on layer 0 (M0) (see Meta-object facility, pp. 2-2 to 2-3.
According to this separation of "model" and "information" the analyzer needs two distinct types of data. First it needs "metadata" which is the description of the data structure of the business application itself and is given as an UML model of the application. This is what sometimes is called the "structure of the business" application and it is on level M1. On the other hand the analyzer needs to know about the data itself, this is what is called "instance data" and it is information on level M1.

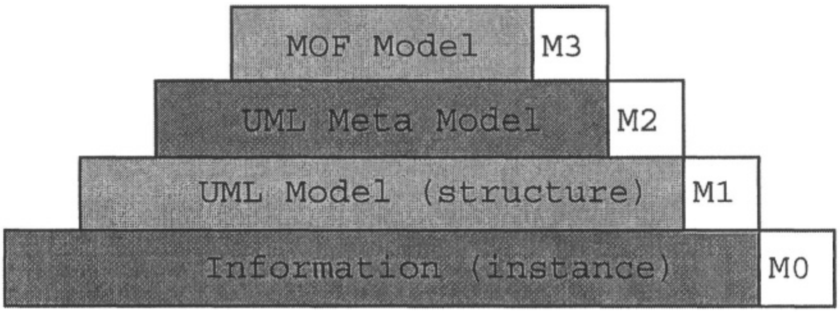


Figure 2. **Meta-model Framework according to OMG**

To illustrate the separation of data on layer M1 and data on layer M0 we consider an example. Assume there is "some" user-data in the business application. Every user has a name and a password. To formally describe the meaning of "some" in the expression "some user-data" there is a "model" that tells the tool about the class user and it's attributes name and password. This is done with an UML model and is data on level M1. When the tool checks the rules and needs to evaluate information of some special user e.g. "John", it needs what is called "information" in the "meta-model framework". This information is called "instance-data" and it is given as XML documents (this is, as the analyzer uses it, placed on layer M0)[Obj02].

2.4 Permissions

To associate permissions for transactions via roles to users in role based access control (RBAC), the tool uses UML class diagrams. These diagrams can be directly used to give this information, and we do not need to introduce any additional features. The tool reads the class diagram and evaluates classes and associations.

In general, the analyzer is not restricted to such an RBAC model or to any specific model at all. It is capable of evaluating rules on *any* class diagram that has the connection attributes assigned as names of the associations and the direction of associations defined by the navigable flag. The analyzer evaluates the model as a graph with classes as nodes and associations as edges, where edges are directed. As we will see later, for the evaluation of rules, we need to require that there must be a path between the two classes involved in that rule, and there must be instance data so that the connecting attributes of each class match.

To explain this in more detail, we consider the example in Figure 3: the class diagram assigning permissions to users consists of the classes *user, role, transaction,* and *permission,* with attributes as in Figure 3. There is an association *role_id* between *user* and *role,* an association *role_id* between *role* and *transaction,* and an association *transaction_id* between *transaction* and *permission.* The analyzer uses this model to automatically find a user's permissions.

Note that when assigning a permission *p* to a user *u* via a role *r,* and the user *u* also happens to have another role ***r'***, then (of course) it is not admissible to conclude that any user ***u'*** with the role ***r'*** should also be granted the permission *p.* In that sense, assigning permissions to users via roles is "unidirectional". In the class diagrams defining permissions, this is specified by using the "navigable" flag of UML class diagrams. This flag is an attribute of an association's endpoint. If this flag is set to "true" at the endpoint of a class *c* (signified by an arrow at that side of the association), our rule-analyzer may associate information from the other end of the association with *c.* If it is set to "false", this information may not be evaluated. This way our tool may gather the permissions with respect to transactions granted to a given user by traversing the class diagram along the associations in the navigable directions permitting a "flow of information". This way the tool "collects" all users that have a given role, but does not recursively collect all users that have any of the roles that a given user has (as explained above).

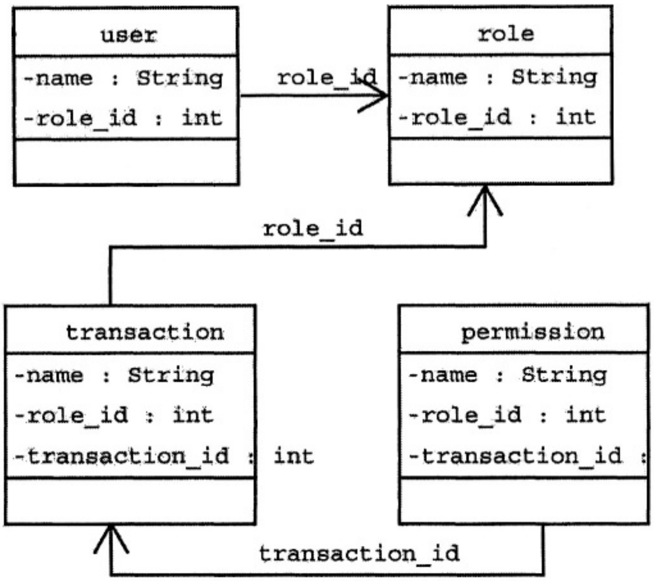


Figure 3. Simple Role Based Access Control

To know how the elements in the application are connected there must be some kind of ID that can be evaluated at both sides of the connection. As in the short example above the user would have some kind of "role-id" in his user data, and a role would have the same id. The application retrieves the user's "role-id" and finds the role with the same id. To express that mechanism in our static UML class diagram, there is an association between the classes that exchange information. The user-class would be associated with the role-class, so the tool knows there is some kind of interaction (i.e., the application is able to find a user's role). To enable the search of information the analyzer implicitly adds an additional attribute to either class at the association's endpoints. These new attributes are assigned the association's name.

```
<rubacon>
   <user>
      <name>john</name>
      <uid>500</uid>
      <group>users</group>
   </user>
   <group>
      <group>users</group>
   </group>
...
</rubacon>
```

Figure 4. Snippet from an instance file

2.5 Instance Data

Besides the structural data elements explained above, we need so-called "instance data". Here an instance may, for example, be a real user of the system. This information is very important for most of the rules one would like to evaluate. There are, of course, rules that do not need instance data (if one is checking the UML data structure model itself for some constraints, for example), but in general there will be instance data. It is read by the analyzer from additional XML files (for an example see Figure 4), containing a tag for every class, and within that tag another tag for each attribute. The analyzer is able to generate the XML-Schema file for an UML model specified by the user, because the contents of the instance file depends on the model of the business application.

2.6 Rules

As defined in the previous section, the business application data structure is represented by a class diagram, that is, a directed graph together with the data from the business application. These two pieces make up a rather complex graph whose structure can be seen in Figure 5 as an example. One can see that for every user in the business application data structure, a node is added. The model gives the tool the information that there is a connection between *user* and *role,* but in the graph in Figure 5 there are only edges between certain users and certain roles. It shows that there is an edge between user *john* and role *users,* because there is the attribute *role* that instantiates it. There is no edge between user *john* and role *admins,* because *john* does not have *admins* in his roles. This is the graph that the analyzer uses to analyze the rules.

Rules in this paper consist of the following elements:

- a name (used as a reference in the security report)
- the type of the rule, which can be either of PROHIBITION or PRECONDITION (meaning that the condition given in the sub-rule defined below should either not be fulfilled, or be fulfilled)
- a message (printed in the report if the rule fails)
- a priority level (to build a hierarchy of importance, so that less important rules can be turned off easily - typical values may include DEBUG, INFO, WARNING, ERROR, FATAL, or a numerical value)
- a sub-rule, which defines a path in the analyzer's graph and a set of constraints, as defined below

A sub-rule has the following elements:

- the head, which is the starting point of the path in the analyzer's graph defined by the sub-rule

- the target, which is the target of that path
- a list of constraints, which defines conditions that the path has to satisfy

Here a constraint consists of the following elements:

- element, the node that has to be checked
- condition, to be checked on that node

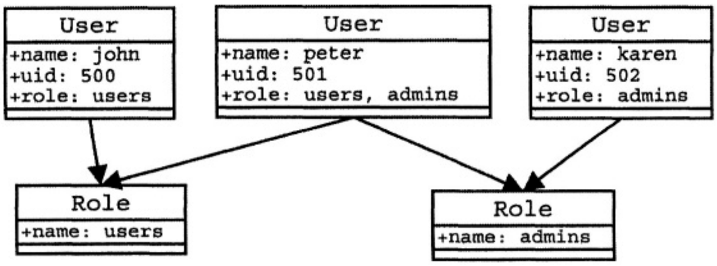


Figure 5. The graph after model and information is inserted

We consider the following example: If it has to be ensured that a certain user, say *john,* does not have the role *admins* assigned, the following parameters would be set for the rule:

name	check user roles
type	PROHIBITION
message	check user for given roles
priority	ERROR = 4

In this example, we have a single sub-rule.

head	user
target	role
constraint	head.user.name = param.user.name
constraint	target.role.name = param.role.name

This rule has two parameters that the user has to provide when generating the report, indicated by the keyword **param**: the user-name *john* and the role *admins.* A suitable XML document that provides these parameters for every rule is expected as input.

The evaluation of this example rule is as follows: The analyzer attempts to find the head of the rule (i.e. "user: john") in the analyzer's graph. Afterwards, it tries to find a path to the target (i.e. "role: admins"). If that succeeds it prints the given message in the security report.

The separation between the rule itself and the two parameters (*param.user.name* and *param.role.name*) is introduced to make editing more comfortable: One does not need to edit a rule for every user and every role that has to be checked.

With the help of these elements rather powerful rules can be defined. To the analyzer the model is a graph representing the business application data structure. The head and the target represent nodes within that graph. For example, head could be *user* and target could be *role.* With that definition there should exist a path between head and target. If it does not, the rule fails. If that path exists, the analyzer will try to fill that path with valid data from the given instance-data. That means that for a valid connection from head to target, every association along that path is instantiated with a discrete entry from the business application's data. If there is no valid instantiation, the rule fails. If there is one, the constraints are checked. Every instantiated element will be examined, and if one of the conditions fails, the rule fails. Otherwise, it succeeds.

To make the rules more expressive, a rule can consist of several sub-rules, where a sub-rule does not have the additional type, message and level attributes. This way the analyzer is powerful enough to check rules such as separation of duty, for example by using the sub-rules:

- check for distinct role A,
- check for distinct role B and
- ensure that no user has both of them.

For a rule to succeed, each of the sub-rules has to succeed.

The type is given to distinguish between preconditions and prohibitions, meaning that either the success or failure of that rule is reported. So it is conveniently possible to define states that must be fulfilled for every configuration and to define states that may not appear within a configuration. For example, it may be vital for a system to have the password set for the super-user account. Conversely, for separation of duty, it would be forbidden for the same user to have two exclusive roles. A template system prints out the messages with any of the instance's attributes in a freely configurable manner. So it is possible to insert values from the violating instance to the message, for example as: "there is no password for user Joe". Only with such a feature the messages become readable and thus the tool easily usable by a human user.

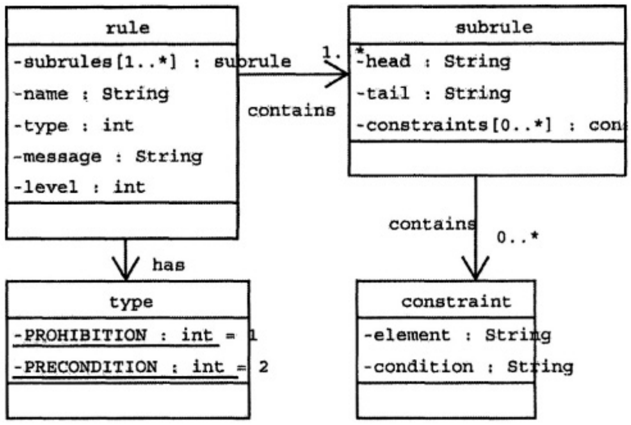


Figure 6. **Class diagram showing the structure of rules**

3. EVALUATING RULES

We use Prolog for the evaluation of the rules. Prolog seems particularly suitable, because it was specifically designed for such a task. In our experience it is also sufficiently efficient for a real-life application. To evaluate the efficiency of Prolog we implemented some simple tests with randomly generated clauses. These clauses were generated such that they had the same structure as the ones generated by the analyzer. We then could evaluate rules, with a database of up to several million clauses in seconds. So it seems, that Prolog will do for the project, were according to our business partners 60.000 entries are generated. If there really will be far more than several millions of clauses, one could think about batch processing.
The advantage of using Prolog is that we can concentrate on the essential problems specific to the rules without having to solve the hard problems of finding the instances along the paths.

3.1 Translating rules to Prolog clauses

First of all the data structure of the business application is defined in Prolog. For this each class from the model describing the business application is converted to a predicate with an argument for each attribute. A class user *(U)* with two Attributes (name (*n*), role-id (*r*)) gives the following expression:

$$U(n,r).$$

To evaluate an expression like “the user’s role”, we need an additional predicate for each association. The “connecting” predicates have the following form: Assume there is a predicate *U(n, r)* and a predicate *R(m, r).* Then the connection *C(n, m, r)* is given by the following term:

$$C(n,m,r) \rightarrow U(n,r) \wedge R(m,r).$$

That means that there is a user *n* in role *m* if there is a user *n* and a role *m* such that the role-id *r* is the same. These predicates can be extended to paths with any number of intermediate nodes, because Prolog evaluates all predicates to true, and the provided connecting attributes match, as in the following example:

$$A(u,v,w,x,y,z) \rightarrow B(u,v) \wedge C(v,w) \wedge D(w,x,y) \wedge E(y,z).$$

Note that the tool could be modified to eliminate the arguments not needed to determine the existence of a path. However, it is convenient to be able to include this additional information in the report.
After the structure is added, the instance will be added, too. For every class several predicates are created. In Prolog syntax that’s what it looks like:

user(john,500).

With the above Prolog-clauses in place, the analyzer can ask for instantiations of the rules.
A short remark regarding the efficiency of the analysis: The “connecting” predicates are added only when a rule needs them. If one would insert every possible connection from every imaginable head to every target, there were up to $n(n-1)$ of these connections. But to evaluate *m* sub-rules one would need at most *m* of these connections. Thus a connection is only added when a sub-rule implies it, reducing the number of connections in general significantly.

3.2 Evaluating Separation of Duty in SAP systems

We use an example configuration from [Sch03] to explain how separation of duty in SAP systems can be evaluated by the rules. First of all, the structure of the business application needs to be defined. For simplicity it will be assumed that the structure looks like the one presented in Figure 3. It certainly is just a very small part of the SAP security concept but as an example, it will be sufficient.

Table 1. Small separation of duty example

User	Role	Transaction	Permission
Karen	Employee (in charge of service)	Create purchase	Is allowed to create purchase in SAP
Susan	Employee (in charge of service, senior in rank to Karen)	Commit purchase	Is allowed to release purchase created by Karen
John	Employee (purchasing agent)	Place orders	Is allowed to place orders by some delivery agent

There are three employees: Karen, Susan and John. Karen and Susan are just employees in any department, and John is a purchasing agent at the company. To have separation of duty, Karen may create a purchase and Susan may release that purchase to John. John may order the desired goods at some supplier firm. With that in place the Prolog rules would be very straight forward:

```
user(Karen, 1)
role(create-purchase, 1)
```

To have separation of duty in place there are two exclusive roles, which may not be assigned to the same user: "create-purchase" and "release-purchase". John just places the orders, he does not do any supervision here. The first sub-rule must have the head "user" and the target "role". The second sub-rule must have the same head and target but it needs a condition:

```
rulel.user.name = rule2.user.name
```

The type of that rule is PROHIBITION, the other attributes do not matter for this example. What does the tool do now? It has created the predicates and inserted the users and the role from the instance files. Afterwards it searches the paths for the rule:

```
user_role(name, role_id, rname) :- user(name, role_id), role(name,
role_id).
```

With that predicate the rule can be evaluated to:

```
user_role_rule(name, role_id1, role_id2) :- user_role(name1, role_id1,
X), user_role(name2, role_id2, Y), name1 = name2.
```

Now Prolog can be asked for

```
user_role_rule(X, create-purchase, release-purchase).
```

and Prolog calculates the correct answer. In the example from Table 1 there is no solution to the predicate, because there is only Karen for role "create-purchase" and Susan for role "release-purchase", and user Karen is not equal to user Susan.
Although this example is very simple, it serves as a demonstration of how the rules can be used. In a real application, the path from user to role might contain several nodes or one might not know the roles that have to be exclusive, just the permissions, so one could exclude permissions contained in roles with several hundreds of entries each.
Note that we do not currently aim to treat object-based permissions, but remain at the class level. While it should be possible to extend our approach in that direction, it is beyond the scope of the current investigation. In particular, this applies to a special kind of separation-of-duty specific to SAP systems: The system can be configured to require more than one user with a certain role to start a transaction. Since the checks needed to enforce this requirement are performed within the SAP system, it would not make sense to repeat them as well at the analysis level. But one should note, that this "internal" separation-of-duty differs form what is presented in our examples. It is often useful to have separation-of-duty throughout different departments, so that the internal one is not sufficient (i.e. if there is some kind of revision after the transaction was performed).

3.3 SAP Transactions

Another example for a use of the rules to improve security is, when the transactions are also part of the data structure. Because of the design of the SAP system (which may seem surprising from a security point of view), there are no security checks performed when a transaction calls another one. By this transitivity, it is very difficult in large systems to see who can execute a transaction. The permission to execute a transaction includes the permission to execute every transaction called by the first one and there does not seem to be a possibility to disable this feature. Thus creating a transaction in SAP is a permission that gives access to everything. One should notice that an employee who is allowed to create a transaction and execute it, can execute any transaction by calling it from his self-created one.
If access needs to be restricted to some transactions, it is therefore not sufficient to ensure that the permission is given only in the roles associated with that transaction, and that only the users allowed to execute that transaction are assigned those roles. It has to be ensured furthermore that there is no transaction calling the restricted one, because SAP would not perform secu-

rity checks there and one would not prevent execution of the restricted transaction.
To do so, one may model the transactions with its sub-transactions as part of the analyzer's model. Then the tool creates rules to check whether permissions grant any user additional rights that are not part of his role. It is usually not advisable to report every transaction that can be executed without explicit permission. Because of the error-prone design, there will be a lot of transactions that are meant to be called implicitly. But the possibility to check for some very "dangerous" transactions (in particular the ones for changing permissions and roles) is a great enhancement of security. This would be improved even more if there was a way to automatically create the data that represents the dependencies of the transactions. We aim to inquire the possibilities to do so.

3.4 Use-Case to Check SAP Permissions

The SAP database is used to generate the information necessary for the rules. An employee creates an UML model describing the SAP system. We use the CASE tool Poseidon for UML to do so. These two documents describe the business application. With these documents in place one can create the rules. For creating the rules there is a graphical user interface but the XML files necessary can be edited manually, too.
When all the documents are prepared, the rules can check the rules automatically. After the rules has finished the checks, the user can read the security-report and start reconfiguring the business application in order to fulfill all the conditions contained in his rule-set.
The security report is formatted as defined by the templates that are part of the analyzer. The analyzer writes a freely configurable HTML file for review with a web browser.

3.5 Further applications

The analyzer can not only be used to check SAP systems, it can be used to check most configurations of large scale applications. The modular architecture makes it easy to adapt to a new application. One needs to define the application's structure in UML, then the instance data must be converted to proper XML files, corresponding to the XML Schema provided by the tool's schema generator. Afterwards, the rules have to be defined. There the graphical user interface can be used, or the XML files can be written manually or generated by any tool fitting the needs of the application. Then the report can be generated by the analyzer.

With that open architecture, we hope to establish a tool for a wide range of rule-checking tasks of configuration files. Our main focus of application is security, but there are other fields where one could use the analyzer,

4. RELATED WORK

One approach to analyzing security configurations is called "Configuration Review Test" [Pol92]. As far as we know there is no implementation of these tests that uses rules for this purpose. Existing tools for this approach check some conditions of specific applications, mostly operating systems. These tools are designed to check for certain security weaknesses, common to a number of systems. Compared to this specific tools, the open architecture presented in the current paper is new for configuration review tests. We consider it a useful new idea for tool-supported security checks. The analyzer presented here could also replace some of the more specific tools, by adding some applications that collect all the information necessary to check, for example, operating system's configurations.

Penetration tests are commonly used to assess the security of a system [Wei95]. In our view, they are complementary to our approach: On the one hand, penetration tests would profit from the information gathered by the analyzer's report. On the other hand, the analyzer presented here does not warn about weaknesses in the software itself (such as programming errors or buffer overflows), but it reports configuration errors. To have a penetration test reveal the errors, the analyzer is designed to check for, one would have to try out every possible transaction. This is usually impractical because there are too many of them. Also, when performed on a live system, the penetration test would be rather invasive.

There are several recent approaches using UML for security analysis, including [Jür02], [Jür03], and [LBD02], and several contributions in [JCF+02]. More generally, there has been a lot of work on formulating security requirements in object-oriented data models (see for example [JKS95] and the references there). Other approaches using logic programming for access control analysis include [BdVS02]. [RS01] uses SQL to administer permissions for distributed data. Compared to that approach, our tool can not only be applied to data bases, but more generally to security configurations. [GAR03] uses a model-checker to analyze Linux configurations.

5. CONCLUSION

The analyzer introduced in this paper is capable of reading the business applications configuration as an UML model and a XML file, therefore it can be easily configured for a wide variety of business applications. The rules used for checking are rather flexible and powerful. While the tool uses a template system for it's report the layout of that report can be freely adopted to any form required.
Misconfiguration of security mechanisms is a major source of attacks in practice. The current work aims to address this issue by providing automated tool-support for checking SAP security permissions. The tool allows one to formulate rules (such as separation-of-duty) that the permissions are supposed to satisfy. It enables one to check automatically that the permissions actually implement the rules even in situations where this is difficult, laborious, and error-prone to perform by hand, because of dynamic changes and the size of the data volumes involved.
Because of its modular architecture and its standardized XML interfaces, the tool can be easily adapted to check security constraints in other kinds of application software (such as firewall or other access control configurations). By making use of standardized mechanisms (such as UML) for specifying the rules, it should be easily learnt to use.
The analysis engine currently exists in a prototype version; development of a product based on it (including the interface to the SAP system) is currently under negotiation. We tested the prototype with example business applications (including some XML test data), including a performance evaluation for large datasets (1,000,000 entries). Hence we can be confident that the tool can be built into an industry-strength application if the market analysis will turn out positive.
One of the advantages of the current work in comparison to other possible approaches to the problem is the possibility to link the analysis of the SAP permissions with an analysis of a business process model given as a UML activity diagram, which is work in progress [Alt03].
More information on the analyzer and on how to obtain a license can be found at **http://www4.in.tum.de/~umlsec.**

REFERENCES

[AJP95] M. Abrams, S. Jajodia, and H. Podell, editors. Information security: an integrated collection of essays. IEEE Computer Society Press, 1995.

[Alt03] E. Alter. SAP permissions and business processes. Master's thesis, TU Munich, 2003. In preparation.

[BdVS02] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. ACM Transactions on Information and System Security, 5(1):1-35, February 2002.

[GAR03] J. D. Guttman, A. L. Herzog, and J. D. Ramsdell. Information flow in operating systems: Eager formal methods. In Workshop on Issues in the Theory of Security (WITS'03). IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS, 2003.

[JCF+02] J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, and R. Sandner, editors. Critical Systems Development with UML, number TUM-I0208 in TUM technical report, 2002. UML'02 satellite workshop proceedings.

[JHC02] J.-M. Jézéquel, H. Hussmann, and S. Cook, editors. UML 2002 - The Unified Modeling Language, volume 2460 of Lecture Notes in Computer Science, Automated Checking of SAP Security Permissions 21 Dresden, Sept. 30 - Oct. 4 2002. Springer-Verlag, Berlin. 5th International Conference.

[JKS95] S. Jajodia, B. Kogan, and R. Sandhu. A multilevel-secure object-oriented data model. In Abrams et al. [AJP95].

[Jür02] J. Jürjens. UMLsec: Extending UML for secure systems development. In Jezequel et al. [JHC02], pages 412-425.

[Jür03] J. Jürjens. Secure Systems Development with UML. Springer-Verlag, Berlin, 2003. In preparation.

[LBD02] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In Jézéquel et al. [JHC02].

[Obj02] Object Management Group. Meta-object facility, version 1.4. In OMG Specifications. OMG, April 2002.

[Pol92] W. Timothy Polk. Automated tools for testing computer systems vulnerability. In NIST Special Publications. National Institute of Standards and Technology, December 1992.

[Pow02] Richard Power. 2002 CSI/FBI computer crime and security survey. Technical report, Computer Security Institute, Spring 2002.

[RS01] A. Rosenthal and E. Sciore. Administering permissions for distributed data: Factoring and automated inference. In IFIP11.3 Conf. on Data and Application Security, 2001.

[Sch03] Marillyn Aidong Schwaiger. Tool-supported analysis of business processes and SAP permissions, 2003. Study project, TU Munich. In preparation.

[Wei95] C. Weissman. Penetration testing. In Abrams et al. [AJP95], chapter 11, pages 269-296.

A FORMAL ANALYSIS OF A DIGITAL SIGNATURE ARCHITECTURE *

David Basin
ETH Zurich
basin@inf.ethz.ch

Kunihiko Miyazaki
Hitachi Systems Development Laboratory
kunihiko@sdl.hitachi.co.jp

Kazuo Takaragi
Hitachi Systems Development Laboratory
takara@sdl.hitachi.co.jp

Abstract We report on a case study in applying formal methods to model and validate an architecture for administrating digital signatures. We use a process-oriented modeling language to model a signature system implemented on top of a secure operating system. Afterwards, we use the Spin model checker to validate access control and integrity properties. We describe here our modeling approach and the benefits gained from our analysis.

Keywords: Formal methods, model checking, data integrity, security architectures.

1. Introduction

We report on a case study in modeling and validating an architecture for administrating digital signatures. The signature architecture is based on the secure operating system DARMA (Hitachi's Dependable Autonomous Realtime Manager), which is used to control the interaction between different subsystems, running on different operating platforms. In particular, DARMA is used to ensure data integrity by separating

*This research project is partially sponsored by Telecommunications Advancement Organization of Japan (TAO).

user API functions, which run on a potentially open system (e.g., connected to the Internet), from those that actually manipulate signature-relevant data, which run on a separate, protected system. The overall architecture should ensure data-integrity, even when the open system is compromised or attacked. We investigate the use of formal methods to validate that this is indeed the case.

More abstractly, we investigate how formal methods can be applied to model and validate the integrity and internal control of an industrial-scale information system. In our study, the architecture modeled is quite complex, involving multiple operating systems and processes communicating between them, which carry out security-critical tasks. A full-scale verification of a particular implementation would not only be impractical, the results would be too specialized. The key here is to find the right level of abstraction to create a model suitable for establishing the security properties of interest.

In our case, which is typical for many data-integrity problems, the relevant security properties concern restricting access to data (e.g., passwords and signatures), where a user's ability to carry out operations depends on past actions, for example, whether the user has been authenticated. Abstractly, these properties correspond to predicates on traces (i.e. sequences) of system events, which suggests building an event-oriented system model that focuses on processes, relevant aspects of their internal computation, and their communication.

Concretely, we model the signature architecture as a system of communicating processes, abstracting away the operational details of the different operating systems as well as functional details like the exact computations performed by different cryptographic primitives. The resulting model describes how processes can interact and semantically defines a set of event traces. We formalize security properties as (temporal) properties of these traces and verify them using the SPIN model checker [4].

Our application of model checking to validate data integrity properties of a security architecture is, to our knowledge, new. Of course, model checking is the standard technique used to verify control-oriented systems [3, 9] and is widely used in hardware and protocol verification. Our work shares with security protocol verification approaches like [10–11] an explicit model of an attacker, where attacker actions can be interleaved with those of honest agents. Our work is also related to the use of model checkers to validate software and architectural specifications [2, 6, 16] and it shares the same problem: the main challenge is to create good abstractions during modeling that help overcome the large, or infinite, state spaces associated with the model.

Organization. In Section 1.2 we present the signature architecture and its requirements. Afterwards, in Sections 1.3 and 1.4, we show how both the system and its requirements can be formalized and rigorously analyzed. Finally, in Section 1.5, we draw conclusions and consider directions for future work.

2. The Signature Architecture

2.1 Overview

The signature architecture is based on two ideas. The first is that of a *hysteresis signature* [14], which is a cryptographic approach designed to overcome the problem that for certain applications digital signatures should be valid for very long time periods. Hysteresis signatures address this problem by chaining signatures together in a way that the signature for each document signed depends on (hash values computed from) all previously signed documents. These chained signatures constitute a signature log and to forge even one signature in the log an attacker must forge (breaking the cryptographic functions behind) a chain of signatures.

The signature system must read the private keys of users from key stores, and read and update signature logs. Hence, the system's security relies on the confidentiality and integrity of this data. The second idea is to protect these using a secure operating platform. For this purpose, Hitachi's DARMA system [1] is used to separate the user's operating system (in practice, Windows) from a second operating system used to manage system data, e.g., Linux. This technology plays a role analogous to network firewalls, but here the two systems are protected by controlling how functions in one system can call functions in the other. In this way, one can precisely limit how users access the functions and data for hysteresis signatures that reside in the Linux operating system space.

Our model is based on Hitachi documentation, which describes the signature architecture using diagrams (like Figures 1 and 2) and natural language text, as well as discussions with Hitachi engineers.

2.2 Functional Units and Dataflow

The signature architecture is organized into five modules, whose high-level structure is depicted in Figure 1. The first module contains the three signature functions that execute in the user operating system space. We call this the "Windows-side module" to reflect the (likely) scenario that they are part of an API available to programs running under the Windows operating system. These functions are essentially

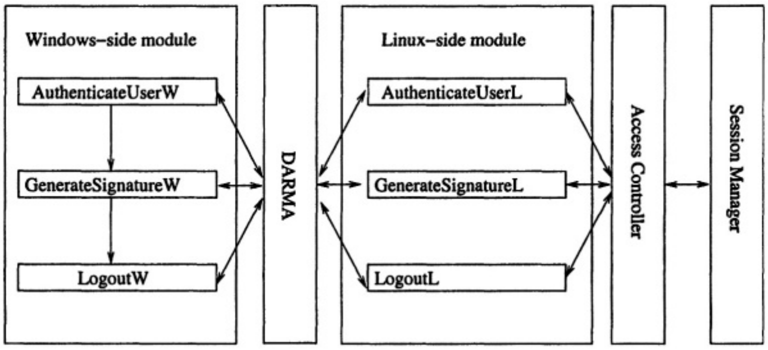


Figure 1. The Signature Architecture

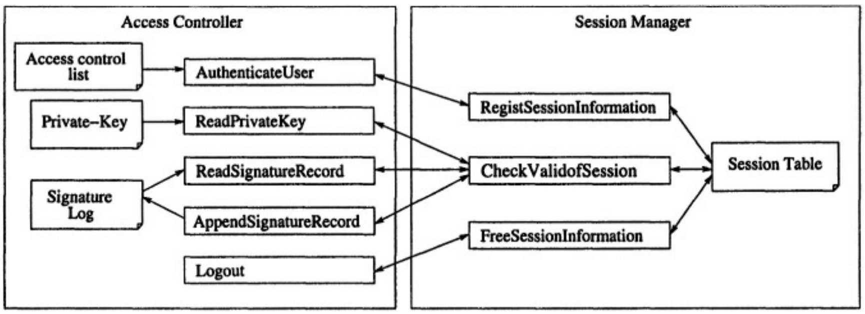


Figure 2. The Access Controller and Session Manager Modules

proxies. When called, they forward their parameters over the DARMA module to the corresponding function in the second, protected, operating system, which is here called the "Linux-side module", again reflecting a likely implementation. There are two additional modules, each also executing on the second (e.g., Linux) operating system, which package data and functions for managing access control and sessions.

To create a hysteresis signature, a user application takes the following steps on the Windows side:

1 The user application calls *AuthenticateUserW* to authenticate the user and is assigned a session identifier.

2 The application calls *GenerateSignatureW* to generate a hysteresis signature.

3 The application calls *LogoutW* to logout, ending the session.

Parameters

Input:

username: sent by *AuthenticateUserW* through *Darma.*

password: sent by *AuthenticateUserW* through *Darma.*

Output:

SessionID: If user authentication is successful, then *SessionID* > 0, otherwise *SessionID* ≤ 0.

Details

1 Calculate hash value of *password* using Keymate/Crypto API. If successful, go to step 2, otherwise set *SessionID* to *CrypotErr* ($\leq$0) and return.

2 Authenticate user using the function *AuthenticateUser* of *Access Controller.*

3 Output *SessionID* returned by *AuthenticateUser.*

Figure 3. Interface Description for *AuthenticateUserL*

As explained above, each of these functions uses DARMA to call the corresponding function on the Linux side. DARMA restricts access from the Windows side to only these three functions. The Linux functions themselves may call any other Linux functions, including those of the *Access Controller,* which controls access to data (private keys, signature logs, and access control lists). The *Access Controller* in turn uses functions provided by the *Session Manager,* which manages session information (*SessionID,* etc.), as depicted in Figure 2.

The Hitachi documentation provides an interface description for each of these 16 functions. As a representative example, Figure 3 presents the description of *AuthenticateUserL.*

2.3 Requirements

The Hitachi documentation states three properties that the signature architecture should fulfill.

1 The signature architecture must authenticate a user before the user generates a hysteresis signature.

2 The signature architecture shall generate a hysteresis signature using the private key of an authenticated user.

3 The signature architecture must generate only one hysteresis signature per authentication.

In Section 1.4, we will show how to model properties like these in temporal logic.

3. Modeling the Signature Architecture

3.1 Process Modeling

Abstraction is the key to creating a formal model of the signature architecture. One possibility is to build a *data model* by formalizing the system data and the functions computed. Alternatively, we can focus on the dynamics of the system and build a *process* or *event-oriented model.*

We take the latter approach. One reason is that data and functions play a limited role in the system description. For example, the architecture description is abstract to the particulars of which cryptographic functions are used to hash or sign messages (i.e., those of the Keymate/Crypto API referred to in *AuthenticateUser* in Figure 3). A second reason is that the properties to be verified are event oriented and have a temporal flavor. They formalize that whenever certain events take place then other events have (or have not) also taken place. This suggests the use of temporal logic for formalizing properties and model checking for property verification.

There are two additional design decisions involved in creating our model, which are representative of the decisions arising in modeling any security architecture. First, we cannot completely abstract away data since control depends on data. In particular, the actions processes take depend on the values of keys, session identifiers, hash values, etc. The solution is to abstract large (or infinite) data domains into finite sets, and abstract functions over data to functions over the corresponding finite sets. The difficult part here is finding an abstraction that respects the properties of the functions acting on data. We will describe our approach to this in Section 1.3.3.

Second, to show that the security properties hold when the system is executed in a hostile environment, we must explicitly model the powers of an attacker. Here we adapt a common approach used in modeling security protocols [8, 12]: In addition to formalizing the system itself, we also formalize how different kinds of users can use the system. That is, we formalize (in Section 1.3.3) both normal "honest" users, who use the system as it is intended, and attackers who use the system in perhaps unintended ways and attempt to exploit and break into the system. The overall system model is built from submodels that define processes for each of the different subsystems together with the processes that model the normal users and the attacker. We then prove that the desired security properties hold of the system, even given all the possible malicious actions that can be taken by the attacker.

We have used the Spin model checker to formalize and check our model. Spin is a generic model checker that supports the design and

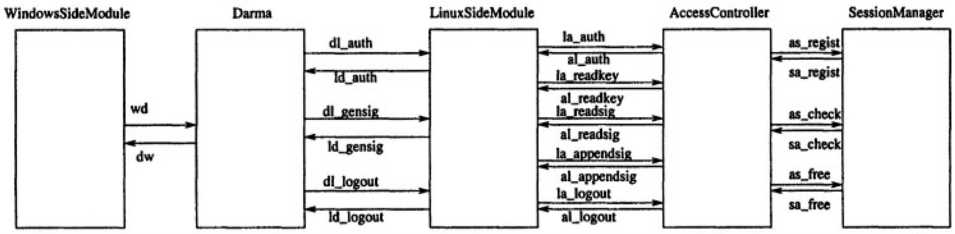


Figure 4. **Modules and Channels**

verification of distributed systems and algorithms. Spin's modeling language is called PROMELA (PROcess MEtaLAnguage), which provides a C-like notation for formalizing processes, enriched with process-algebra like primitives for expressing parallel composition, communication, and the like. Properties may be expressed in a linear-time temporal logic (LTL) and Spin implements algorithms for LTL model checking. Due to space restrictions, we will introduce Spin constructs as needed, on-the-fly. For a detailed description of Spin, the reader should consult [4–5].

3.2 Functions and Function Calls

As suggested by Figures 1 and 2, we can model the signature architecture in terms of five modules that communicate with each other in restricted ways. We will model each such module as a PROMELA process, where each process communicates with other processes over channels. A PROMELA channel is a buffer of some declared size that holds data of specified types. For each function in a module, we define two channels: one for modeling function calls and the other for modeling the return of computed values. This is depicted in Figure 4, which names the channels used for passing data between processes. All channels are declared to have size zero. According to the semantics of PROMELA, this means that communication on these channels is synchronous: the process sending data on a channel and the process receiving data from the channel must rendezvous, i.e., carry out their actions simultaneously.

As the figure shows, between Windows and DARMA we have just one calling channel *wd* and one returning channel *dw*.[1] This reflects that we have only one function in the *Darma* interface. This function is called

[1]Note that we ignore channels for calling the Windows functions since the functions that actually call *AuthenticateUserW, GenerateSignatureW,* and *LogoutW* fall outside the scope of our model, i.e., we do not consider who calls them, or how the caller uses the return values.

by marshaling (i.e., packaging) the function arguments together, including the name of the function to be called on the Linux side. We model this by putting all these arguments on the channel. For example, the expression *wd!AuthUser,username,password* (which occurs in our model of a normal user, given shortly), models that the function *AuthenticateUserW* calls *Darma,* instructing *Darma* to call *AuthenticateUserL* with the arguments *username* and *password.*

3.3 User Modeling

We now explain our formalization of the powers and actions of both ordinary users and system attackers.

The description of the signature architecture in Section 1.2 describes how the system is intended to be used by normal users. As we will see, it is a simple matter to translate this description into a user model.

The Hitachi documentation, describes, in part, the powers of an attacker, in particular that he cannot access functions on the Linux side. This is a starting point for our formalization of an attacker model, but it leaves many aspects open, for example, whether an attacker can operate "within" the system as a legitimate user with a valid password, or if he is an outsider, without these abilities. Moreover, it is unspecified what the attacker knows, or can guess or feasibly compute.

One achieves the strongest security guarantees by proving the safety of a system in the face of the most general and powerful attacker possible. Hence, we model an attacker who cannot only function as a legitimate user of the system, but can also call functions in unintended ways, with arbitrary parameters. Moreover, he knows, or can guess or compute, the names of other users, messages, and message hashes, and of course he knows his own password. However, we assume he can neither guess the passwords nor the session identifiers of other users. (If either of these were the case, then forging signatures would be trivial.)

We summarize these assumptions as follows:

1 The attacker can call *AuthenticateUserW, GenerateSignatureW,* and *LogoutW* in any order.

2 The attacker is also a legitimate user with a user name and a password.

3 The attacker knows the names of all users, and can guess messages and message hashes.

4 The attacker can only give his (good) password or a bad guessed password.

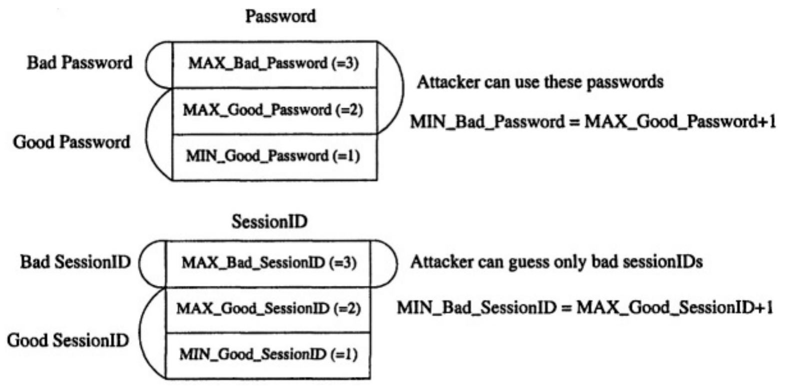


Figure 5. Modeling Passwords and Session Identifiers

5 The attacker cannot guess a good *SessionID,* i.e., one used by other users.

6 Generated *SessionID*s are always good.

In our model, we define sets of objects, namely finite intervals of natural numbers, for modeling the different kinds of objects in the problem domain: names, messages, hash values, and passwords. The key idea is to partition these sets into those things that are known by the attacker (or can be guessed or computed) and those that are not. For example, there is a set of user names, formalized by the set of natural numbers {*MIN_username, ..., MAX_username*}. We model that the spy knows, or can guess, any of these names by allowing him to guess any number in this set. However, we partition the ranges corresponding to passwords and session identifiers so that the attacker can only guess "bad" ones, which are ones that are never assigned to normal users. However, the attacker also has a "good" password, which allows him to use the system as a normal user and generate a good session identifier. Figure 5 depicts this partitioning, with the concrete values that we later use when model checking. For example, the good passwords are $\{1, 2\}$, where 2 represents the attacker's password. He can only guess passwords in the range {2,3}, where 3 models a bad password, i.e., one that does not belong to any normal user. As he cannot guess the password 1, he cannot use the system (e.g., to generate a signature) as any user other than himself.

Given this abstraction, it is now a simple matter to model the actions of normal users and the attacker.

```
1  proctype WindowsSideModule_Normal() {
2    byte username, password, sessionID, message, message_hash, signature, result;
3
4   setrandom(username, MIN_Good_Username, MAX_Good_Username);
5   setrandom(password, MIN_Good_Password, MAX_Good_Password);
6
7    do
8    :: wd!AuthUser,username,password;
9       dw?AuthUser,sessionID;
10
11      setrandom(message, MIN_Message, MAX_Message);
12      message_hash = Hash(message);
13
14      wd!GenSig,sessionID,message_hash;
15      dw?GenSig,signature;
16
17      wd!Logout,sessionID,0; /* second argument '0' is dummy */
18      dw?Logout,result
19  od}
```

Figure 6. User Model

Normal Users. Figure 6 shows our model[2] of a normal user, which directly models the steps that a normal user takes when using the signature architecture.

In lines 4 and 5 we model different possibilities for who uses the system and their messages. The macro *setrandom(x,lower,upper)* uses nondeterministic choice to set x to a value, *lower* $\leq x \leq$ *upper*. Hence these lines set the username and password to those of a normal user, chosen nondeterministically from the predefined ranges.

Afterwards, the user generates a hysteresis signature. On line 8, the user calls *Darma* on the *wd* channel, specifying the execution of the Linux-side user authentication function, along with his username and password. On line 9, the result, a session identifier (whose value is greater than zero when authentication is successful), is returned on the *dw* channel.

On lines 11–12, a message from the space of possible messages is nondeterministically selected and its message hash is computed. We model *Hash* simply as the identity function. Although this does not reflect the functional requirements of a real hash function, in particular, that it is a one-way function, it is adequate for establishing the stipulated properties of our process model, which only rely on passwords and session

[2]Model excerpts are taken verbatim from our PROMELA model, with the exception of pretty printing, line numbering, and minor simplifications for expository purposes.

```
1 proctype WindowsSideModule_Attacker() {
2   byte username, password, sessionID, signature, dummy, result;
3   bit message_hash;
4
5   setrandom(username, MIN_username, MAX_username);
6   setrandom(message_hash, MIN_Message_Hash, MAX_Message_Hash);
7   setrandom(password, MAX_Good_Password, MAX_Bad_Password);
8   setrandom(sessionID, MIN_Bad_SessionID, MAX_Bad_SessionID);
9
10  do  /* Attacker calls these three functions in any order */
11  :: wd!AuthUser,username,password;
12     dw?AuthUser,sessionID
13
14  :: wd!GenSig,sessionID,message_hash;
15     dw?GenSig,signature
16
17  :: wd!Logout,sessionID,dummy;
18     dw?Logout,result
19
20     /* Or, Attacker guesses the following values */
21  :: setrandom(username, MIN_username, MAX_username)
22  :: setrandom(message_hash, MIN_Message_Hash, MAX_Message_Hash)
23  :: setrandom(password, MAX_Good_Password, MAX_Bad_Password)
24  :: setrandom(sessionID, MIN_Bad_SessionID, MAX_Bad_SessionID)
25  od}
```

Figure 7. Attacker Model

identifiers being unguessable. On line 14, the user calls *Darma* on the *wd* channel, instructing *Darma* to generate a signature with the session identifier returned from the previous round of authentication and a message hash. The generated signature is returned on line 15. Note that the return value can also indicate an error, e.g., if the session identifier was invalid.

Lines 17–18 model the user logging out, which invalidates his session identifier.

The Attacker. Figure 7 shows the PROMELA process that formalizes our attacker model. Here we see that the attacker can guess an arbitrary user name and message hash (lines 5–6). However, in accordance with the guessing model depicted in Figure 5, he can only guess one good password (*Max_Good_Password*), which allows him to log in as a normal user, or bad passwords (line 7). Similarly, he can only guess bad session identifiers (line 8).

Afterwards, we use a *do/od* loop with nondeterministic choice to model the attacker repeatedly calling *Darma* (on the *wd* channel) with

```
1    :: dl_auth?username_LINUX,password
2       -> password_hash = Hash(password);
3          if
4          :: (password_hash <= 0) -> sessionID_LINUX = HashFunctionErr;
5                                     goto DONE_AuthL
6          :: else
7          fi;
8
9          la_auth!username_LINUX,password_hash;
10        al_auth?sessionID_LINUX;
11
12      DONE_AuthL:
13         ld_auth!sessionID_LINUX
```

Figure 8. AuthenticateUserL

these guessed values, in any order he likes. Alternatively, as modeled by the last four actions, he can guess new values at any point in time.

This example again illustrates the power of nondeterminism in a process-oriented modeling language. As with the user model, we use it to leave open which values are taken on by variables. This models a system where these variables can take on any value from the specified sets at system runtime. In addition, we use nondeterminism to describe the different possible actions that can be carried out by a user, while allowing the actions to be ordered in any way. The result is a succinct description of a general, powerful attacker. Of course, formalizations like this, which involve substantial nondeterminism, will typically lead to verification problems with large states spaces. But this can be seen as a feature, not a bug: model checkers can often search the resulting state spaces much quicker and more accurately than humans can.

3.4 Function Modeling

The majority of our PROMELA model describes the 16 functions contained in the system modules. As a representative function, we return to *AuthenticateUserL,* first described in Section 1.2.2.

Figure 8 shows the part of the PROMELA process that models *Authenticate UserL* (this module also contains definitions for the other Linux-side functions). This directly models the three steps explained in Section 1.2.2: calculate a hash value (lines 2–7), authenticate the user (lines 9–10), and return the session identifier (line 13).

Here we have a simple example of how creating a rigorous specification forces us to make all definitions explicit. Step 1 of the textual explanation states "If [hash value calculation is] successful, go to Step 2 ...". But the Boolean predicate "successful" isn't defined. Such omis-

```
init {
    run WindowsSideModule_Normal();
    run WindowsSideModule_Attacker();
    run Darma();
    lsm = run LinuxSideModule();
    run AccessController();
    run SessionManager()}
```

Figure 9. Initialization Process

sions arise frequently. In this example, it is easy to determine what is intended by reading other parts of the specification. Here we formalize success by stating that a *HashFunctionErr* is generated when the password hash is less than or equal to zero, and the operation is successful otherwise. In general, not all ambiguities are so easily resolved. One of the benefits of using a formal specification language is that we are forced to be unambiguous at all times; PROMELA contains a syntax checker and automatically detects undefined symbols.

3.5 Putting It All Together

We build the overall model by composing in parallel the processes defined above. Namely, we compose the two processes formalizing the Windows-side module (as used by normal users and by the attacker) and the processes for the remaining modules. This is depicted in Figure 9. Note that we associate an identifier *lsm* with the process executing the Linux-side module. This will be used during verification to refer to particular labels in an invocation of the *LinuxSideModule* process, as described in the next section.

4. Verification

We now describe how we use Spin to show that our model has the intended properties. To do this, we formalize "bad" behavior (by formalizing and negating "good" behavior) as temporal logic formulae. Spin converts, on-the-fly, our PROMELA model of the system and the temporal logic formula to automata (reducing model checking to an automata-theoretic problem as described in [15]), and then constructs and searches the resulting product automaton. If Spin finds a trace accepted by this automaton, the trace explains how the system allows the bad behavior. Alternatively, if Spin succeeds in showing, by exhaustive analysis of the state space, that no errors exists, then the model is verified with respect to the property.

As an example, we formalize the first of the three properties described in Section 1.2.3. Our first requirement states that the signature architecture must authenticate a user before the user generates a signature. The bad property is therefore the negation of this. Informally:

The signature architecture generates a signature for an unauthenticated user.

To formalize this as a temporal property, observe that to generate a signature, we first require a valid session identifier, which is the result of a successful user authentication. Suppose that *UAS(uname,sID)* denotes that the user *uname* is authenticated with the session identifier *sID* and that *GHSS(sID)* represents that the signature architecture has generated a hysteresis signature with the session identifier *sID* (greater than zero). This can be formalized in PROMELA as follows.

```
#define UAS(uname,sID) (LinuxSideModule[lsm]@DONE_AuthL
    && username_LINUX == uname && sessionID_LINUX == sID)

#define GHSS(sID) (LinuxSideModule[lsm]@DONE_GensigL
    && signature_LINUX > 0 && sessionID_LINUX == sID)
```

In these definitions, we reference labels (using @) in our PROMELA model to formalize that processes have reached certain points in their execution, and we use predicates on variables to express conditions on the system state.

We can now express the above informally stated property as

$$\exists s : \mathit{session}.\ \mathit{GHSS}(s)\ \mathit{before}\ \exists u : \mathit{user}.\ \mathit{UAS}(u, s)\,. \tag{1}$$

This is not yet a formula of linear-time temporal logic. First, “before” is not a standard LTL operator. However it can be expressed using the LTL operator “until”, written as infix **U**, by defining *A before B* as $(\neg B)\ \mathbf{U}\ A$, i.e., *A* occurs before *B* if and only if $\neg B$ holds until *A*. In our case

$$\exists s : \mathit{session}.\,(\neg\exists u : \mathit{user}.\ \mathit{UAS}(u, s))\ \mathbf{U}\ \mathit{GHSS}(s)\,. \tag{2}$$

Second, we must eliminate the two quantifiers over sets. Since these sets are finite, we can replace each quantifier by finitely many disjunctions, i.e., the formula $\exists s$: *session.* $P(s)$ can be expanded to $P(s_1) \vee P(s_2) \cdots \vee P(s_n)$, where $s_1, \ldots, s_n$ are the finitely many model representatives of session identifiers.

The resulting property is automatically verified by Spin in 2 hours of computation time on a 450 MHz UltraSparc II workstation. In doing so, it builds a product automaton with over 20 million states and searches over 70 million transitions. The formalization and verification of the other two properties is similar.

5. Discussion

It took approximately one man-month to build and analyze the signature architecture. This included considering alternative designs and studying different ways of specifying the requirements. The resulting model is 647 lines of PROMELA.

Although the formal analysis did not expose any design errors, the process itself was still quite valuable. During the formalization, we uncovered numerous ambiguities and omissions in the Hitachi documentation, such as missing cases and undefined values. For example, as described in Section 1.3.3, we needed to explicitly formalize implicit assumptions on the environment. Indeed, one reason why verification was successful is that, during the modeling process itself, we uncovered and fixed these oversights and omissions. Moreover, the discipline involved in creating the formal model improved our understanding of the design and helped us identify better solutions. This process of sharpening and improving a design is often one of the major benefits of using formal methods.

As a concrete result, we have produced a verified model. It serves as unambiguous documentation, with a well-defined mathematical semantics, for subsequent system development. It also provides a starting point for formally certifying the signature architecture with respect to standards like the Common Criteria.

As future work, we would like to explore the possibility of undertaking a full scale verification, perhaps as part of such a certification. An important step here would be to formally verify the correctness of the abstractions used, i.e., that the verification of our small finite model entails the verification of the corresponding infinite state system with unbounded numbers of interacting users and infinite data domains. Techniques based on data-independence, such as those of [7, 13], may help automate this task. It would also be interesting to supplement our process model with a data model that formalizes the properties of the signature architecture functions. Although this is not required for verifying the three requirements examined here, this would be necessary to go beyond treating cryptography as a "black box", e.g., to reason about the adequacy of different cryptographic mechanisms. This would also provide a more complete (formal) documentation of the design and provide a starting point for code verification.

References

[1] Toshiaki Arai, Tomoki Sekiguchi, Masahide Satoh, Taro Inoue, Tomoaki Nakamura, and Hideki Iwao. Darma: Using different OSs concurrently based on nano-kernel technology. In *Proc. 59th-Annual Convention of Information Processing Society of Japan,* volume 1, pages 139–140. Information Processing Society of Japan, 1999. In Japanese.

[2] W. Chan, R. J. Anderson, P. Beanie, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering,* 24(7):498–520, July 1998.

[3] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* The MIT Press, 1999.

[4] Gerard J. Holzmann. The model checker SPIN. *Software Engineering,* 23(5):279–295, 1997.

[5] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, 1991.

[6] Daniel Jackson and Kevin Sullivan. COM revisited: tool-assisted modelling of an architectural framework. In *ACM SIGSOFT Symposium on Foundations of Software Engineering,* pages 149–158. ACM Press, 2000.

[7] Gavin Lowe. Towards a completeness result for model checking of security protocols. In *PCSFW: Proceedings of The 11th Computer Security Foundations Workshop,* pages 96–105. IEEE Computer Society Press, 1998.

[8] Gawin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS'96,* LNCS 1055, pages 147–166. Springer, Berlin, 1996.

[9] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[10] C. Meadows. The NRL protocol analyzer: an overview. *Journal of Logic Programming,* 26(2):113–131, 1996.

[11] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *Proceedings of IEEE Symposium on Security and Privacy,* pages 141–153, 1997.

[12] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security,* 6:85–128, 1998.

[13] A. W. Roscoe and Philippa J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security,* 7(1):147–190, 1999.

[14] Seiichi Susaki and Tsutomu Matsumoto. Alibi establishment for electronic signatures. *Information Processing Society of Japan,* 43(8):2381–2393, 2002. In Japanese.

[15] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences,* 32:183–221, 1986.

[16] Jeannette Wing and Mandana Vaziri-Farahani. A case study in model checking software systems. *Science of Computer Programming,* 28:273–299, 1997.

USING PARAMETERIZED UML TO SPECIFY AND COMPOSE ACCESS CONTROL MODELS

Indrakshi Ray, Na Li, Dae-Kyoo Kim, Robert France
Department of Computer Science, Colorado State University

Abstract: Situations can arise in which organizations have to merge policies that are based on different access control frameworks, such as Role Based Access Control (RBAC) and Mandatory Access Control (MAC). Integrating policies requires addressing the following question: How will the integration impact access to protected resources? In particular, one needs to determine that the integration does not result in security breaches or unavailability of resources. A way of addressing these concerns is to model the access control frameworks, compose the models, and analyze the resulting model to identify problems. In this paper we outline a technique for modeling and composing access control policy frameworks. Specifically, we model RBAC and MAC using a variant of the Unified Modeling Language (UML) and show how to compose the models. The composed model can be used as a base for defining access control policies in a military domain.

Key words: Mandatory Access Control, Role-Based Access Control, Unified Modeling Language

1. INTRODUCTION

An access control framework provides a set of rules, concepts, and guidelines for defining access control policies. Two popular frameworks are Role Based Access Control (RBAC) and Mandatory Access Control (MAC). Situations can arise in which organizations have to merge policies that are based on different access control frameworks. In such cases one needs to ensure that unauthorized persons are not inadvertently given access to protected resources, and that authorized persons are not denied access to resources as a result of emergent properties of the merged policies.

In this paper we propose a modeling approach that can be used to describe the effect of composing policies developed under different frameworks. In the approach the frameworks involved are modeled and the resulting models are composed to produce a hybrid model. The hybrid model can then be analyzed to uncover undesirable emergent properties that can potentially prevent authorized access or allow unauthorized access.

The variant of the Unified Modeling Language (UML) [10] is used to specify access control frameworks. A number of reasons motivated the use of the UML. First, the UML is a de-facto industry standard for modeling software artifacts. Second, the graphical nature of the UML and its use of widely understood concepts make it relatively easy to use and understand. Third, the graphical notation allows one to more easily detect inconsistencies and other problems in model.

In our modeling approach, access control frameworks are specified by generic models that describe the family of access control behaviors and structures that conform to the rules defined in the framework. A framework is described by a set of UML template models. Instantiation of a template model results in a UML model that describes an application-specific policy that conforms to the framework.

Composition of access control models results in a *hybrid access control* (HAC) model that is also expressed in the template form. To compose two access control models we first determine (1) the elements in the two models that will be merged, (2) the elements that will appear "as is" in the composed model and (3) the elements that will be modified or omitted in the composed model. We also determine the application domain specific constraints that affect the hybrid model. This information is then used to compose the models to produce the HAC for a given domain. In this paper we illustrate our approach by integrating the hierarchical RBAC framework and the MAC framework for a military domain.

The rest of the paper is organized as follows. Section 2 gives an overview of RBAC and MAC and shows how one can model them using template forms of UML diagrams. Section 3 describes how the models can be composed to produce a hybrid access control model. Section 4 describes some of the related work in this area, and Section 5 concludes the paper.

2. MODELING RBAC AND MAC

In this section we give an overview of the UML, RBAC and MAC, and illustrate how RBAC and MAC can be modeled using template forms of UML diagrams. The UML is a standard modeling language maintained by the Object Management Group (OMG) (See http://www.omg.org/uml for details). The UML defines notations for building many diagrams that each presents a particular view of the artifact being modeled. In this paper we utilize the following diagram types:

Class Diagram: A class diagram depicts the static structural aspects of an artifact being modeled. It describes the concepts and the relationships between them. Relationships are expressed using associations and generalizations/specializations. Concepts are described in terms of their properties, where a property can be represented by an attribute, a behavioral unit (an operation), or a relationship with another concept.

Interaction Diagram: An interaction diagram describes a behavioral aspect of an artifact being modeled. Specifically, it describes how instances of the concepts described in a class diagram collaborate in order to accomplish behavioral goals. The UML has two interchangeable forms of interaction diagrams: collaboration diagrams show both the interactions and the supporting instance structure; sequence diagrams show only the interactions. In this paper we use collaboration diagrams because they capture more information.

Examples of these diagrams will be described when used in this paper.

2.1 Role-Based Access Control

RBAC [1] is used to protect information objects (henceforth referred to as objects) from unauthorized users. To achieve this goal, RBAC specifies and enforces different kinds of constraints. Depending on the nature of the constraints present in a specific RBAC, the RBAC will use one or more of the following components: *Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations,* and *Dynamic Separation of Duty Relations.*

Core RBAC embodies the essential aspects of RBAC. The constraints specified by Core RBAC are present in any RBAC application. The Core RBAC requires that users (human) be assigned to roles (job function), roles be associated with permissions (approval to perform an operation on an object), and users acquire permissions by being members of roles. The Core RBAC does not place any constraint on the cardinalities of the user-role assignment relation or the permission-role association. Core RBAC also includes the notion of user sessions. A user establishes a session during which he activates a subset of the roles assigned to him. Each user can activate multiple sessions; however, each session is associated with only one user. The operations that a user can perform in a session depend on the roles activated in that session and the permissions associated with those roles.

Hierarchical RBAC adds constraints to Core RBAC for supporting role hierarchies. Hierarchies help in structuring the roles of an organization. Role hierarchies define an inheritance relation among the roles in terms of permissions and user assignments. In other words, role *r*1 inherits role *r*2 only if all permissions of *r*2 are also permissions of *r*1 and all users of *r*1 are also users of *r*2. There are no cardinality constraints on the inheritance relationship. The inheritance relationship is reflexive, transitive and anti-symmetric.

Static Separation of Duty Relations (SSD) are necessary to prevent conflict of interests that arise when a user gains permissions associated with conflicting roles (roles that cannot be assigned to the same user). SSD relations are specified for any pair of roles that conflict. The SSD relation places a constraint on the assignment of users to roles, that is, membership in one role that takes part in an SSD relation prevents the user from being a member of the other conflicting role. The SSD relationship is symmetric, but it is neither reflexive nor transitive. SSD may exist in the absence of role hierarchies (referred to as SSD RBAC), or in the presence of role hierarchies (referred to as hierarchical SSD RBAC). The presence of role hierarchies complicates the enforcement of the SSD relations: before assigning users to roles not only should one check the direct user assignments but also the indirect user assignments that occur due to the presence of the role hierarchies.

Dynamic Separation of Duty Relations (DSD RBAC) aims to prevent conflict of interests as well. The DSD relations place constraints on the roles that can be activated within a user's session. If one role that takes part in a DSD relation is activated, the user cannot activate the other conflicting role in the same session.

In this paper we do not consider the constraints specified by DSD RBAC relations. We use hierarchical SSD to explain our ideas. In other words, our model comprises the following components: Core RBAC, hierarchical RBAC and SSD RBAC. A model of hierarchical SSD RBAC is shown in Fig. 1.

The hierarchical SSD RBAC (Fig. 1) consists of: 1) a set of users *(USERS)* where a user is an intelligent autonomous agent, 2) a set of roles *(ROLES)* where a role is a job function, 3) a set of objects *(OBS)* where an object is an entity that contains or receives information, 4) a set of operations *(OPS)* where an operation is an executable image of a program, and 5) a set of permissions *(PRMS)* where a permission is an approval to perform an operation on objects. The cardinalities of the relationships are indicated by the absence (denoting one) or presence of arrows (denoting many) on the corresponding associations. For example, the association of user to session is one-to-many. All other associations shown in the figure are many-to-many. The association labeled *Role Hierarchy* defines the inheritance relationship among roles. The association labeled *SSD* specifies the roles that conflict with each other.

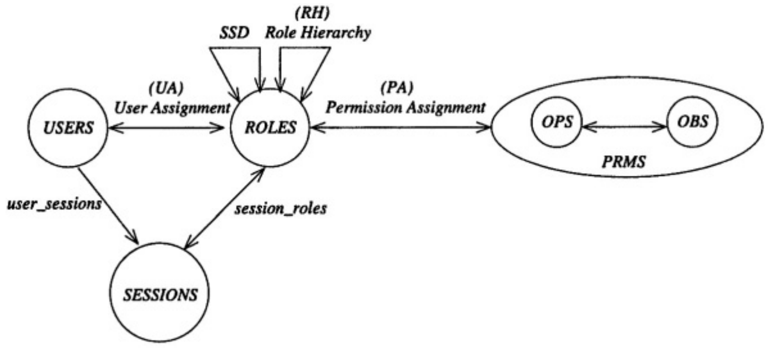


Figure 1. **Hierarchical RBAC with SSD**

The constraint in hierarchical SSD is given below (see [1] for formal definition): users cannot be assigned to roles that are involved in an SSD relation. In the following sections we give a graphical approach by which such constraint violations can be easily detected.

2.2 Modeling RBAC using Parameterized UML

In this section we specify the hierarchical SSD RBAC in terms of UML class diagram templates. A class diagram template consists of parameterized class elements, for example, parameterized class attributes, class operations, and relationships. A parameterized class is a class descriptor with parameters. It defines a family of classes where each class is obtained by binding the parameters to actual values.

Fig. 2 shows a class diagram template describing the hierarchical SSD RBAC. In the diagram, we use a symbol "|" to indicate parameters to be bound. This is not standard UML: we use this notation instead of the parameter list notation defined in the UML standard because it is more concise when there are many parameters involved. Each class template consists of two parts: one part consists of attribute templates that produce class attributes when instantiated, and the other part consists of operation templates that produce class operations when instantiated.

The RBAC template consists of the following parameterized classes: *User, Role, Session, Permission, Object, Operation.* The class template *User* has one attribute template named *UserID.* The behavior of the class template *User* is described by the operation templates *CreateSession* (creates a new session), *DeleteSession* (deletes an existing session), *AssignRole* (assigns a new role to the user) and *DeassignRole* (removes an existing role from the user). Typically there are many other *User* RBAC behaviors that can be described by operation templates. To keep the diagram simple, we do not show all of them. One operation template not shown in the figure is

G*etRoles* (returns the roles assigned to the user). The class templates *Role, Session,* and *Permission* are similarly specified.

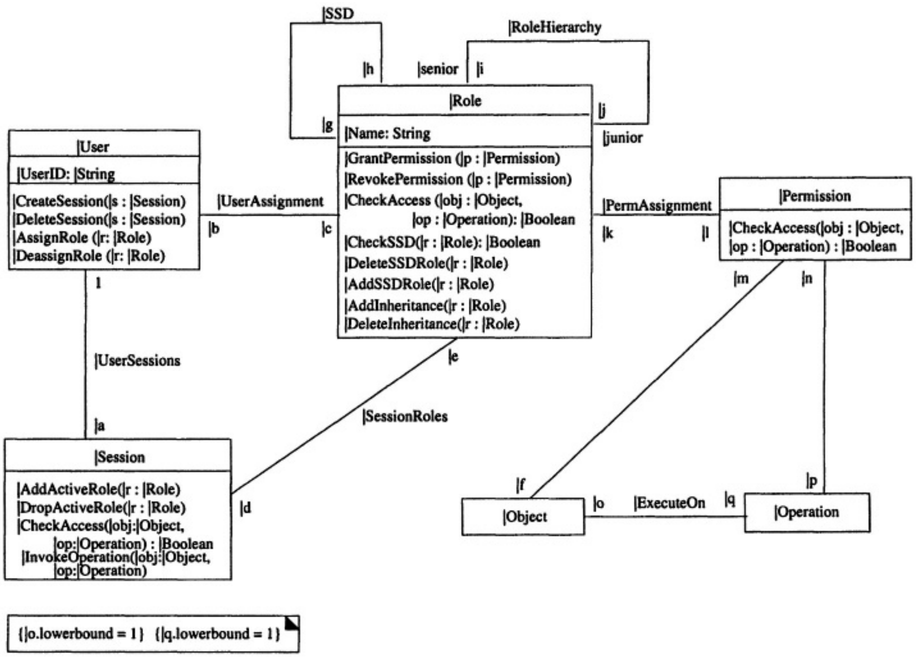

|SSD
|RoleHierarchy
|h
|senior
|i
|Role
|g
|Name: String
|GrantPermission (|p : |Permission)
|RevokePermission (|p : |Permission)
|CheckAccess (|obj : |Object, |op : |Operation): |Boolean
|CheckSSD(|r : |Role): |Boolean
|DeleteSSDRole(|r : |Role)
|AddSSDRole(|r : |Role)
|AddInheritance(|r : |Role)
|DeleteInheritance(|r : |Role)
|j
|junior
|User
|UserID: |String
|CreateSession(|s : |Session)
|DeleteSession(|s : |Session)
|AssignRole (|r: |Role)
|DeassignRole (|r: |Role)
|UserAssignment
|b
|c
|PermAssignment
|k
|l
|Permission
|CheckAccess(|obj : |Object, |op : |Operation) : |Boolean
1
|m
|n
|e
|UserSessions
|SessionRoles
|a
|Session
|AddActiveRole(|r : |Role)
|DropActiveRole(|r : |Role)
|CheckAccess(|obj:|Object, |op:|Operation) : |Boolean
|InvokeOperation(|obj:|Object, |op:|Operation)
|d
|f
|p
|Object
|o
|ExecuteOn
|q
|Operation
{|o.lowerbound = 1} {|q.lowerbound = 1}

Figure 2. **UML Class Diagram Templates Modeling Hierarchical SSD RBAC**

Association templates, such as *UserAssignment* and *SessionRoles* produce associations between instantiations of the class templates they link. An association template consists of multiplicity parameters (one at each end) that yield association multiplicities (integer ranges) when instantiated. A *UserSessions* link is created by the *CreateSession* operation in an instantiation of the RBAC model; this link gets deleted by the corresponding *DeleteSession* operation. The operation *AssignRole* creates a *UserAssignment* link; the *DeassignRole* removes this link. The operations *GrantPermission* and *RevokePermission* are responsible for creating and deleting, respectively, the link *PermAssignment.* An *SSD* link is created by the *AddSSDRole* operation; this link is deleted by the *DeleteSSDRole* relation. The operation *AddInheritance* adds a link *RoleHierarchy; DeleteInheritance* deletes this link. For lack of space, we do not show the full specification of each operation. Pre- and postcondition templates in the operation template *CreateSession* is given below:

context |User :: |CreateSession(|s: |Session)

post: result = |s and |s.oclIsNew() = true and self.|Session includes(|s)

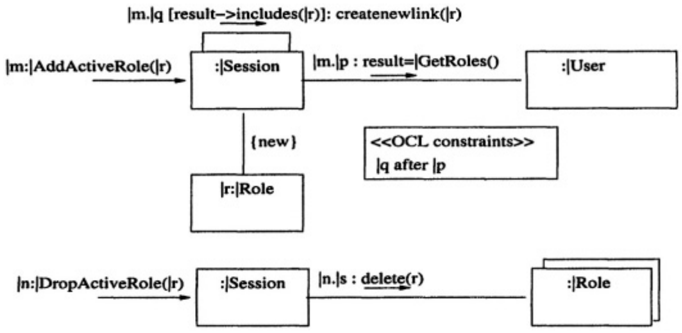


Figure 3. UML Collaboration Diagram Templates Modeling AddActiveRole and DropActiveRole

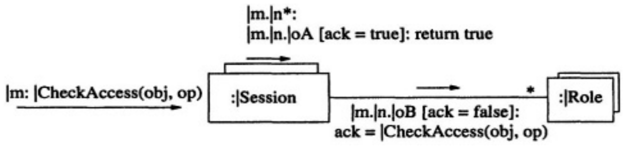

|m.|n*:
|m.|n.|oA [ack = true]: return true
|m: |CheckAccess(obj, op)
:|Session
*
:|Role
|m.|n.|oB [ack = false]:
ack = |CheckAccess(obj, op)

Figure 4. UML Collaboration Diagram Templates Modeling CheckAccess

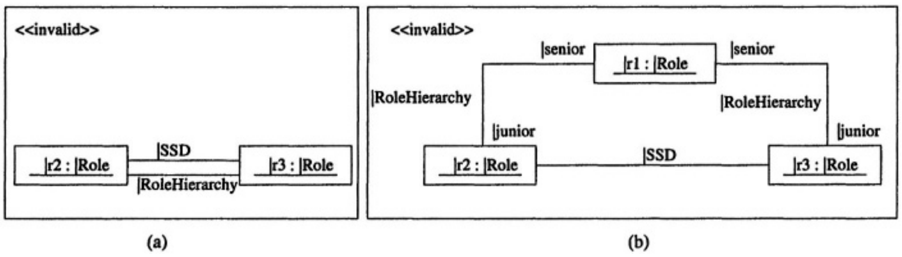


Figure 5. UML Object Diagram Templates Modeling Conflicts

The constraints in the bottom left corner of Fig. 2 impose restrictions on the cardinalities of the associations. For example, {|o.lowerbound = 1} restricts an association that is an instance of *ExecuteOn* to having a multiplicity of at least one at the association end *Object.* The multiplicity "1" of *UserSessions* association end on *User* is strict: a session can only be associated with one user.

We can specify constraints using the Object Constraint Language (OCL) in class diagram templates (as is done in Fig. 2). Alternatively, one can express valid and invalid structures using object diagram templates. For example, Fig. 5 shows an object diagram template specification that illustrates the violation of the constraint of the hierarchical SSD RBAC. Fig. 5(a) shows that there cannot be a policy in which two roles *r*2 and *r*3 connected by a hierarchy are also involved in an SSD relation. Fig. 5(b) specifies that there cannot be a policy in which two roles *r*2, *r*3 that are in an SSD relation have the same senior role *r*1. Presence of such patterns indicates a problem with the specification.

The class diagram templates do not specify the interactions between operations. Collaboration diagram templates are used for this purpose. Fig. 3 shows interactions involving the operations *AddActiveRole* and *DropActiveRole* in the class template *Session.* For example, the *AddActiveRole* causes the invocation of the operations *GetRoles* in *User.* If the role to be activated *r* is included in the result, then a new link is established between the session and the new role *r. DropActiveRole* causes this link to be deleted. The details of *CheckAccess* operation in *Session* appears in Fig. 4.

2.3 Mandatory Access Control

The Mandatory Access Control framework that we use is adapted from the Bell-Lapadula model [8]. The Bell-Lapadula model is defined in terms of a security structure $(L, \geq)$. L is the set of security levels, and $\geq$ is the dominance relation between these levels. The main components of this model are objects, users, and subjects. Objects contain or receive information. Each object in the Bell-Lapadula model is associated with a security level which is called the classification of the object. User, in this model, refers to human beings. Each user is also associated with a security level that is referred to as the clearance of the user. Each user is associated with one or more subjects. Subjects are processes that are executed on behalf of some user logged in at a specific security level. The security level associated with a subject is the same as the level at which the user has logged in.

The mandatory access control policies in the Bell-Lapadula model are specified in terms of subjects and objects. The policies for reading and writing objects are given by the Simple Security and Restricted-★ Properties.

- *Simple Security Property*: A subject *S* may have read access to an object *O* only if the security level of the subject *L*(*S*) dominates the security level of the object *L*(*O*), that is, $L(S) \geq L(O)$.
- *Restricted-★ Property:* A subject *S* may have write access to an object *O* only if the security level of the subject *L(S)* equals the security level of the object *L*(*O*), that is, $L(O) = L(S)$.

The static structural aspects of the MAC are described in the class diagram template shown in Fig. 6. The write and read operations prohibited by MAC are shown using object diagram templates in Fig. 7 and Fig. 8. The behavior of the *CheckAccess* operation is described by the collaboration diagram template shown in Fig. 9.

3. HYBRID ACCESS CONTROL: HAC

In this section we show how RBAC and MAC can be composed. The following steps identify how the composition of access control models takes place. Later in Section 3.1 we show how the application domain causes modification to the hybrid model.

Step 1 Identify the entities in each of the access control frameworks.

Step 2 Compare the definition of an entity in one model to that of another in the second model, and determine which represent similar concepts and which represent dissimilar concepts.

Step 3 Matched entities (those representing similar concepts as determined in the previous step) are merged in the composed model.

Step 4 Dissimilar entities that must be present in the composed model are added "as is" to the composed model, or are modified (as determined in step 2). Access model entities that have been identified for elimination are not added to the composed model.

We make the following observations about the elements in MAC and RBAC.

- *User, Object, Operation* are used in both the models and they each refer to the same concepts (see Section 2).
- *Subject* (used in MAC) and *Session* (used in RBAC) refer to the same concept [9].
- *SecurityLevel* appears in one model (MAC) but not in the other (RBAC).

Based on the above observations, we apply our algorithm to generate the hybrid access control model.

1. The *User* elements are merged in the two models. The structural feature templates are identical in both RBAC and MAC. The MAC specifies only one operation *CreateSubject* for *User.* The merged element includes the behavioral feature template from the element in RBAC and also those of MAC. The merged element appears in the hybrid model and we refer to it as *User.*
2. The elements *Object* and *Operation* in both the models are identical. Each of these elements is added to the hybrid model.

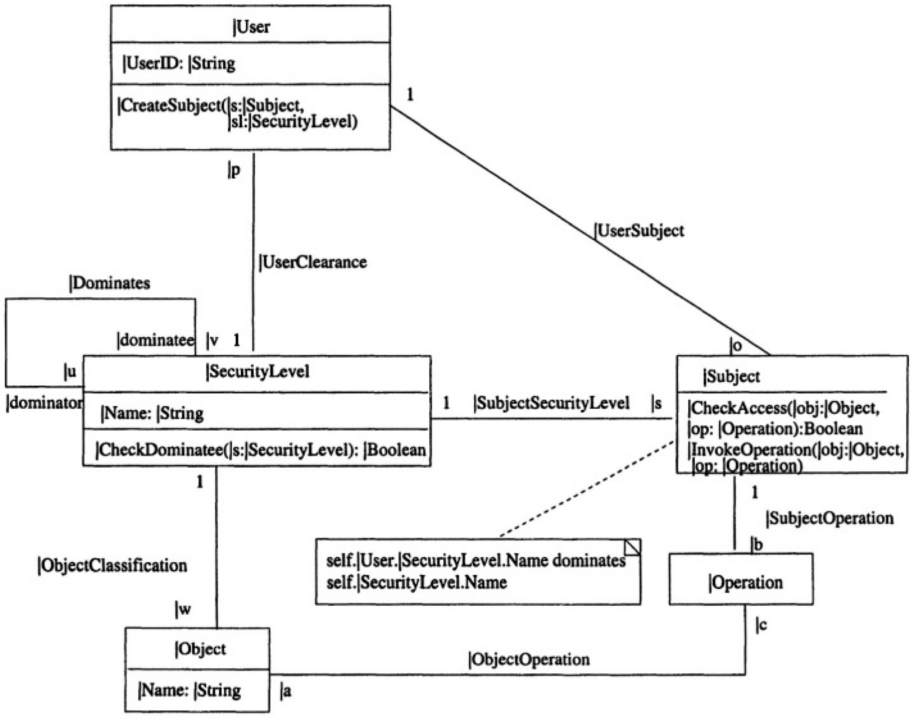

|User
|UserID: |String
|CreateSubject(|s:|Subject, |sl:|SecurityLevel)
1
|p
|UserClearance
|UserSubject
|Dominates
|dominatee
|v
1
|u
|dominator
|SecurityLevel
|Name: |String
|CheckDominatee(|s:|SecurityLevel): |Boolean
1
|SubjectSecurityLevel
|s
|o
|Subject
|CheckAccess(|obj:|Object, |op: |Operation):Boolean
|InvokeOperation(|obj:|Object, |op: |Operation)
1
|SubjectOperation
|b
|ObjectClassification
1
self.|User.|SecurityLevel.Name dominates self.|SecurityLevel.Name
|Operation
|w
|c
|Object
|Name: |String
|ObjectOperation
|a

Figure 6. Mandatory Access Control (MAC)

3. The element *SecurityLevel* (present in MAC but not RBAC) is added to the hybrid model.
4. The elements *Session* and *Subject* refer to the same concept. These elements are merged. Since *Subject* is associated with *SecurityLevel,* the merged element is now associated with a security level. The merged element is referred to as *Session* in the hybrid model. Comparing these two elements in the two models, we see that RBAC *Session* has behavioral feature templates, such as, *AddActiveRole*and *DropActiveRole* that are not present in *Subject.* These behavioral feature templates are added in the *Session*element of HAC. Both *Subject* and *Session* have *CheckAccess* and *InvokeOperation.* These operations if different must be merged. The *CheckAccess* operation in HAC is changed to reflect this. The collaboration diagram template of *CheckAccess* appears in Fig. 12. The merging of *Subject* and *Session* also affects other model elements. For instance, consider the class template *User* in HAC. From Step 1, the operations of *User* are *CreateSession, DeleteSession, AssignRole, DeassignRole,* and *CreateSubject.* Since *Subject* and *Session* are merged, we need to have only one operation called *CreateSession* (because the merged entity in HAC is called *Session*). Moreover, the *CreateSession* of

RBAC must be changed because now the security level also has to be passed as a parameter. The collaboration diagram templates of *CreateSession* in RBAC, *CreateSubject* in MAC, and *CreateSession* in HAC are given in Fig. 10.

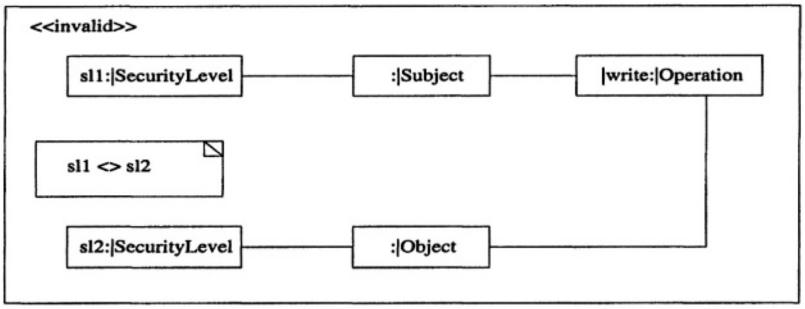


Figure 7. Write Operation Prohibited by MAC

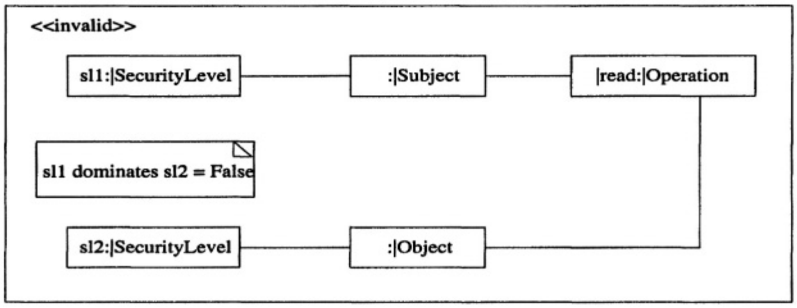


Figure 8. Read Operation Prohibited by MAC

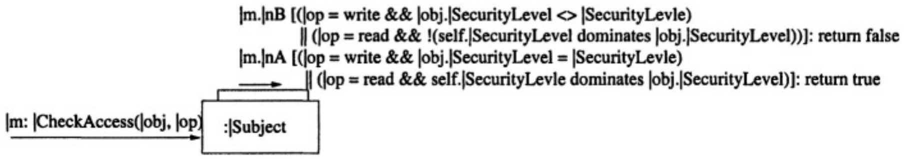

|m.|nB [(|op = write && |obj.|SecurityLevel <> |SecurityLevle)
|| (|op = read && !(self.|SecurityLevel dominates |obj.|SecurityLevel))]: return false
|m.|nA [(|op = write && |obj.|SecurityLevel = |SecurityLevle)
|| (|op = read && self.|SecurityLevle dominates |obj.|SecurityLevel)]: return true
|m: |CheckAccess(|obj, |op)
:|Subject

Figure 9. CheckAccess Operation in MAC

The class diagram template of this HAC is shown in Fig. 11.

3.1 Military Domain Requirement

The HAC shown in Fig. 11 may be suitable for applications, such as, an academic environment, that do not require any additional domain-specific constraints. For others, we may need to modify the hybrid model to incorporate the application domain specific constraints. Consider, for example, the military environment. The concept of *Role* in the military environment is associated with security levels [7]. For example, the role *CentralCommander* is associated with a security level of *TopSecret.* The roles *JointPlanner, ArmyLogisticsOfficer* are associated with a security level of *Secret* etc. To capture this association, we add an additional link to Fig. 11 in order to get the HAC for the military domain in Fig. 15 (the additional link is shown as a dark line in the figure).

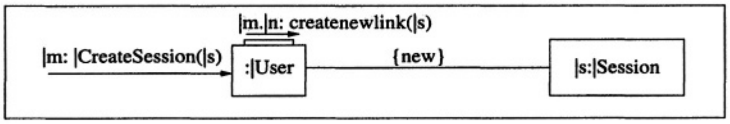
Roles being associated with security levels have other implications. For example, whenever a user is assigned to a role, the security level of the *User* must dominate the security level of the *Role.* In other words the *AssignRole* operation in the class *User* needs to be changed. Similarly, the roles that can be activated in a session must be the roles whose security levels are the same as the security level of the session. The *AddActiveRole* must be modified to check this additional precondition.

These additional constraints modify the HAC in the following way.

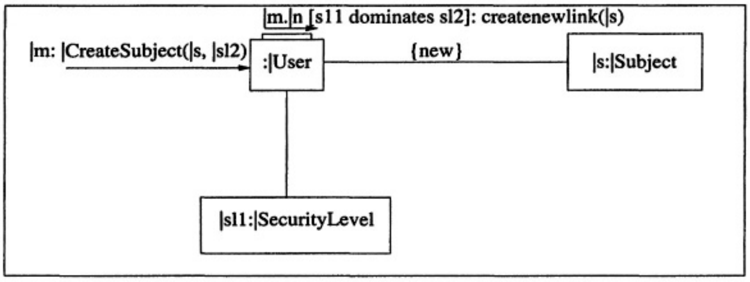
- An association is established between *Role* and *SessionLevel.*
- The operation *AssignRole* now must check that the security level of the *User* must dominate the security level of the *Role.* The operation *AddActiveRole* must be changed to check that the security level of the role activated must be equal to the level of the *Session.* To observe the change in the operation *AddActiveRole,* compare the *AddActiveRole* in RBAC (Fig. 3) with that of the *AddActiveRole* in HAC (Fig. 13).

The collaboration diagram templates in Fig. 14 show the *AssignRole* operations in RBAC and HAC.

Merging access control frameworks may result in conflicts. For example, RBAC may allow a specific role *r*1 at security level *s*2 to write an object at the same security level. Because of role hierarchy the senior role *r*2 at security Level *s*1 (where *s*1 dominates *s*2) is also allowed to write on that object. However, this behavior is prohibited by MAC. Fig. 16 describes some undesirable patterns that indicate a presence of conflict in the hybrid model.



(a) CreateSession in RBAC


|m.|n [sl1 dominates sl2]: createnewlink(|s)
|m: |CreateSubject(|s, |sl2)
:|User
{new}
|s:|Subject
|sl1:|SecurityLevel

(b) CreateSubject in MAC

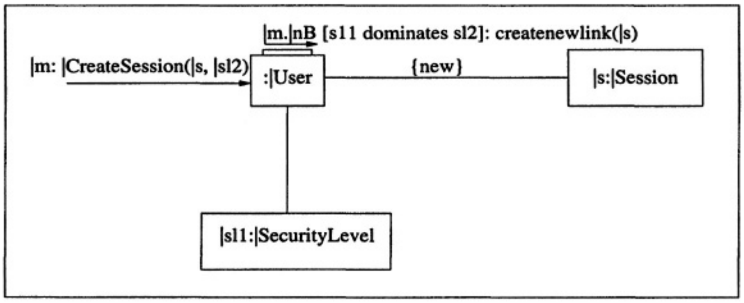

|m.|nB [sl1 dominates sl2]: createnewlink(|s)
|m: |CreateSession(|s, |sl2)
:|User
{new}
|s:|Session
|sl1:|SecurityLevel

Figure 10. Collaboration Diagram Templates for CreateSession

4. RELATED WORK

Several researchers have looked into integrating the mandatory access control and role-based access control models. Osborn [6] examines the interaction between RBAC and MAC. The author discusses the possible structures of role graphs that do not violate the constraints imposed by MAC.

In their approach when a subject is assigned to a role, the subject can perform all the privileges in the role, the possible structures that ensure that no violations of secrecy can occur are discussed. Their research shows that the combination of the structure imposed by the role graphs and the MAC rules means that the possible structure of a role graph in which roles are assignable to subjects without violating MAC rules is greatly restricted.

Nyanchama and Osborn [5] discuss the realization of MAC in role-based protection systems.

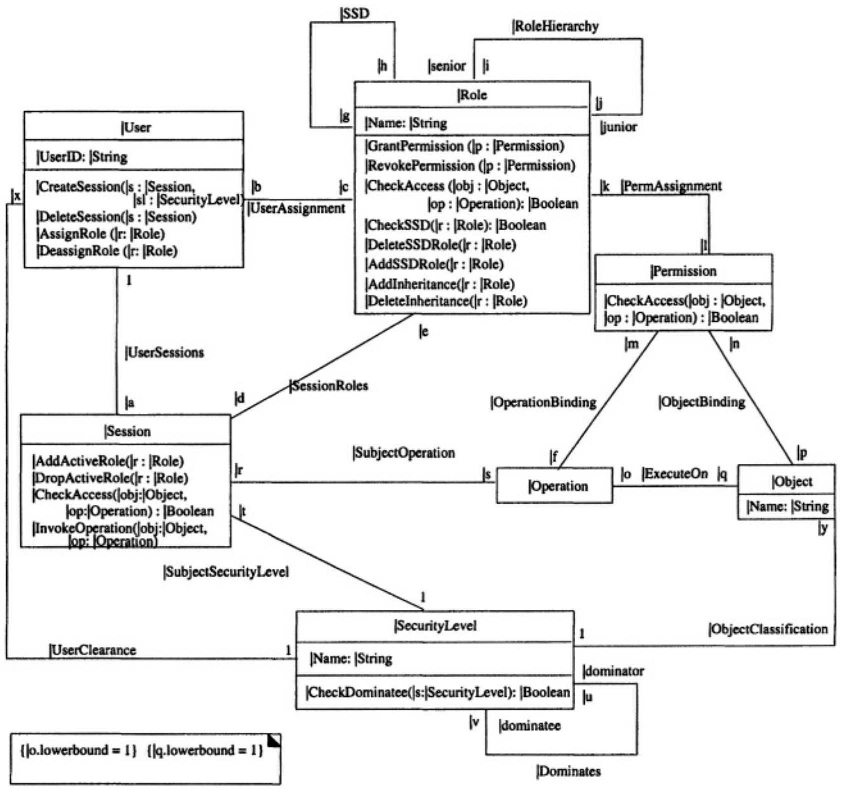

|SSD
|RoleHierarchy
|h
|senior
|i
|Role
|g
|Name: |String
|GrantPermission (|p : |Permission)
|RevokePermission (|p : |Permission)
|CheckAccess (|obj : |Object, |op : |Operation): |Boolean
|CheckSSD(|r : |Role): |Boolean
|DeleteSSDRole(|r : |Role)
|AddSSDRole(|r : |Role)
|AddInheritance(|r : |Role)
|DeleteInheritance(|r : |Role)
|j
|junior
|User
|UserID: |String
|CreateSession(|s : |Session, |sl : |SecurityLevel)
|DeleteSession(|s : |Session)
|AssignRole (|r: |Role)
|DeassignRole (|r: |Role)
|x
|b
|c
|UserAssignment
|k
|PermAssignment
|l
|Permission
|CheckAccess(|obj : |Object, |op : |Operation) : |Boolean
1
|UserSessions
|e
|m
|n
|a
|d
|SessionRoles
|OperationBinding
|ObjectBinding
|Session
|AddActiveRole(|r : |Role)
|DropActiveRole(|r : |Role)
|CheckAccess(|obj:|Object, |op:|Operation) : |Boolean
|InvokeOperation(|obj:|Object, |op: |Operation)
|r
|SubjectOperation
|s
|f
|Operation
|o
|ExecuteOn
|q
|p
|Object
|Name: |String
|y
|t
|SubjectSecurityLevel
1
|SecurityLevel
|Name: |String
|CheckDominatee(|s:|SecurityLevel): |Boolean
1
|ObjectClassification
|UserClearance
1
|dominator
|u
|v
|dominatee
|Dominates
{|o.lowerbound = 1} {|q.lowerbound = 1}

Figure 11. Hybrid Access Control (HAC)

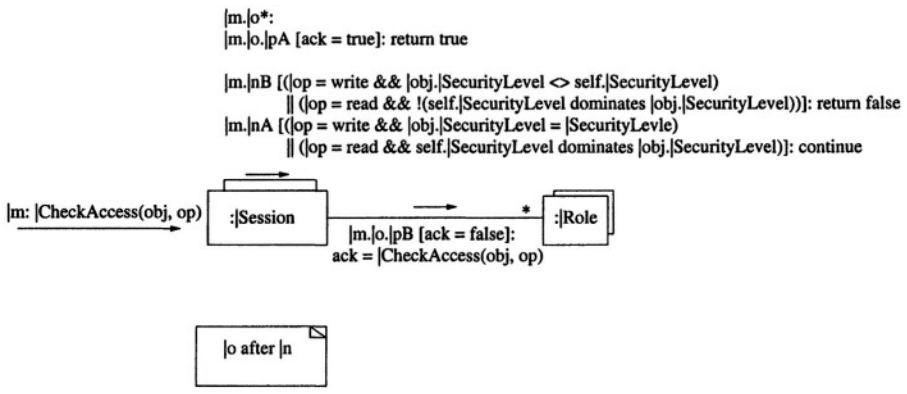

|m.|o*:
|m.|o.|pA [ack = true]: return true
|m.|nB [(|op = write && |obj.|SecurityLevel <> self.|SecurityLevel)
|| (|op = read && !(self.|SecurityLevel dominates |obj.|SecurityLevel))]: return false
|m.|nA [(|op = write && |obj.|SecurityLevel = |SecurityLevle)
|| (|op = read && self.|SecurityLevel dominates |obj.|SecurityLevel)]: continue
|m: |CheckAccess(obj, op)
:|Session
*
:|Role
|m.|o.|pB [ack = false]:
ack = |CheckAccess(obj, op)
|o after |n

Figure 12. CheckAccess Operation in HAC

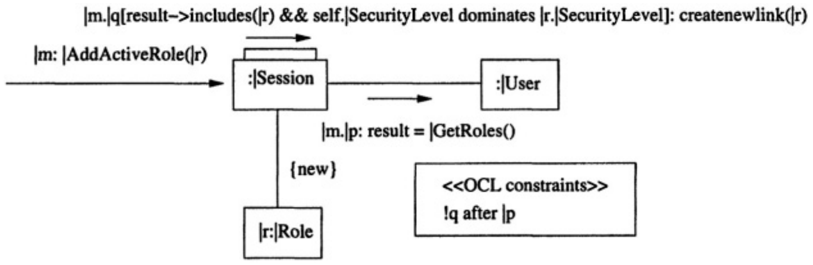

|m.|q[result->includes(|r) && self.|SecurityLevel dominates |r.|SecurityLevel]: createnewlink(|r)
|m: |AddActiveRole(|r)
:|Session
:|User
|m.|p: result = |GetRoles()
{new}
<<OCL constraints>>
!q after |p
|r:|Role

Figure 13. **Collaboration Diagram Template for AddActiveRole Operation in HAC**

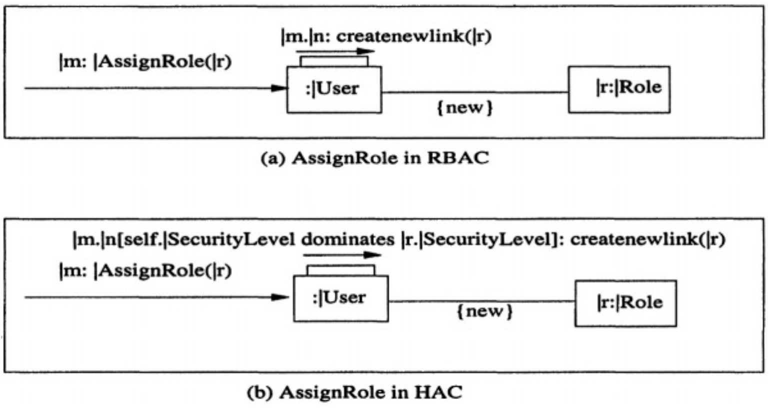


Figure 14. **Collaboration Diagram Templates for *AssignRole* Operation in HAC**

Phillips et al. [7] examine the unification of MAC and RBAC into a security model and enforcement framework for distributed applications. In their work, RBAC is extended to include MAC to ensure that the clearance of users playing roles meets or exceeds classification of resources, services, and methods being utilized. A role is assigned a classification, the authorized user must possess a classification greater than or equal to the role classification.

Researchers [4,2] have also investigated extending UML for representing access control. Lodderstedt et al. [4] propose SecureUML and define a vocabulary for annotating UML-based models with information relevant to access control. Jurgens [2] model security mechanisms based on the multi-level classification of data in a system using an extended form of the UML called UMLsec. The UML tag extension mechanism is used to denote sensitive data. Statechart diagrams model the dynamic behavior of objects, and sequence diagrams are used to model protocols.

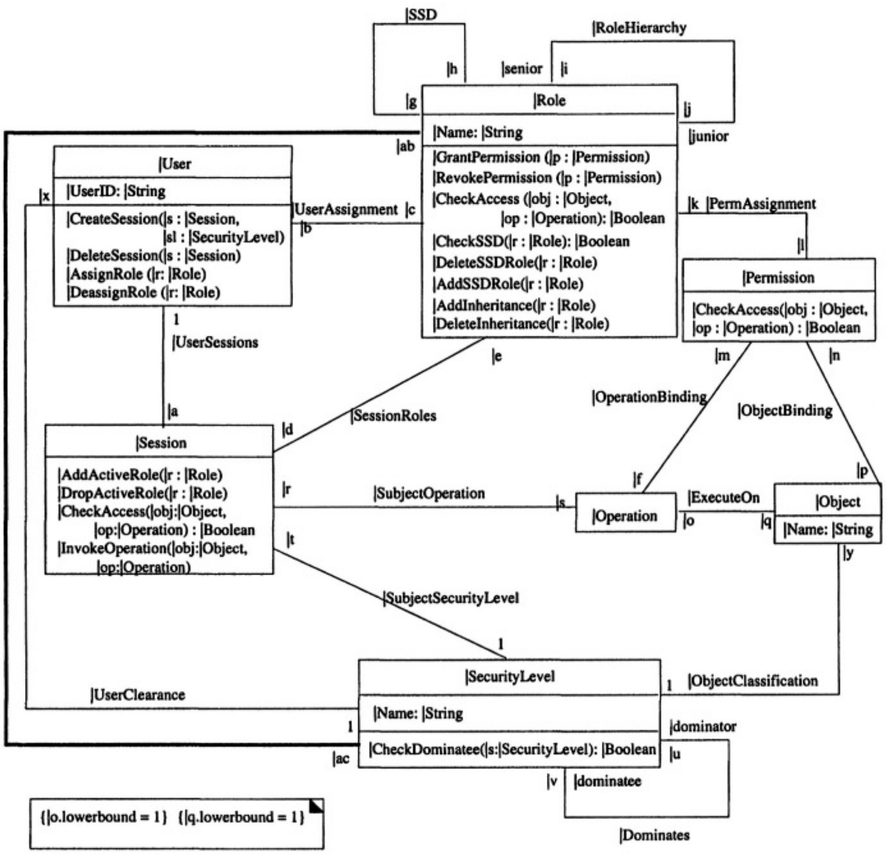

|SSD
|RoleHierarchy
|h
|senior
|i
|g
|Role
|j
|Name: |String
|junior
|ab
|GrantPermission (|p : |Permission)
|RevokePermission (|p : |Permission)
|CheckAccess (|obj : |Object, |op : |Operation): |Boolean
|CheckSSD(|r : |Role): |Boolean
|DeleteSSDRole(|r : |Role)
|AddSSDRole(|r : |Role)
|AddInheritance(|r : |Role)
|DeleteInheritance(|r : |Role)
|User
|UserID: |String
|CreateSession(|s : |Session, |sl : |SecurityLevel)
|DeleteSession(|s : |Session)
|AssignRole (|r: |Role)
|DeassignRole (|r: |Role)
|x
|UserAssignment
|c
|b
|k
|PermAssignment
|l
|Permission
|CheckAccess(|obj : |Object, |op : |Operation) : |Boolean
1
|UserSessions
|e
|m
|n
|OperationBinding
|ObjectBinding
|a
|d
|SessionRoles
|Session
|AddActiveRole(|r : |Role)
|DropActiveRole(|r : |Role)
|CheckAccess(|obj:|Object, |op:|Operation) : |Boolean
|InvokeOperation(|obj:|Object, |op:|Operation)
|r
|SubjectOperation
|s
|Operation
|f
|ExecuteOn
|o
|q
|p
|Object
|Name: |String
|y
|t
|SubjectSecurityLevel
1
|SecurityLevel
|Name: |String
|CheckDominatee(|s:|SecurityLevel): |Boolean
1
|ObjectClassification
|UserClearance
1
|ac
|dominator
|u
|v
|dominatee
|Dominates
{|o.lowerbound = 1} {|q.lowerbound = 1}

Figure 15. Hybrid Access Control (HAC) for Military Domain

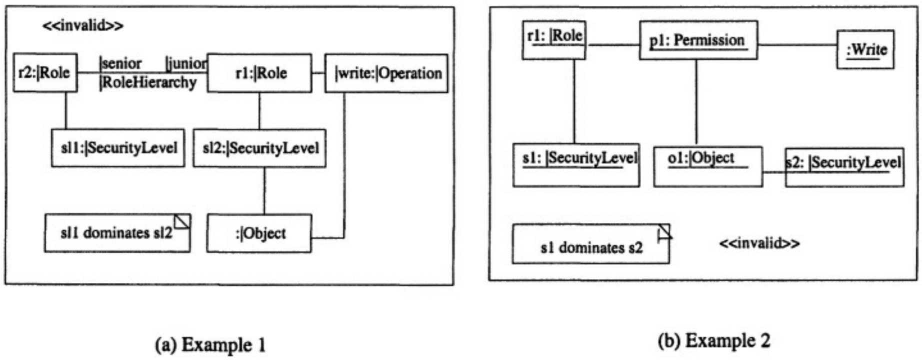


Figure 16. Examples of Invalid Patterns in HAC

5. CONCLUSION

The need to integrate different kinds of access control frameworks arise when organizations using these frameworks need to work together in a collaborative environment. In this paper we show how to model these frameworks in a form that allows for easy composition, how to compose these models, and identify conflicts arising because of the composition.

A lot of work still remains to be done. The integration process, presented in this paper, is done manually. We plan to develop tools that will automate, in part, this composition process. The tool can compare the elements in the different models based on their structural and behavioral specifications and aid the user in resolving conflicts. Other types of conflict, such as mismatch in multiplicity parameters, mismatch in the specification of operations, that occur during the integration process can also be automatically detected by tools.

References

[1] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security,* 4(3), August 2001.

[2] Jan J-urjens. UMLsec: Extending UML for Secure Systems Development. In UML2002, 2002.

[3] C. E. Landwehr, C. L. Heitmeyer, and J. McLean. A security model for military message systems. In ACM Transactions on Computer Systems 2(3): 198-222, August, 1984.

[4] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In 5th International Conference on the Unified Modeling Language, 2002.

[5] Matunda Nyanchama and Sylvia Osborn. Modeling Mandatory Access Control in Role-Based Security Systems. In IFIP Workshop on Database Security, 1995.

[6] Sylvia Osborn. Mandatory access control and role-based access control revisited. In Proceedings of the Second Workshop on Role-Based Access Control, 1997.

[7] C. E. Phillips, S. A. Demurjian, and T. C. Ting. Toward information assurance in dynamic coalitions. In Proceedings of the IEEE Workshop on Information Assurance, United States Military Academy, West Point, NY, June 2002.

[8] R. Sandhu and P. Samarati. Access Control: Principles and Practice. In IEEE Communications, Volumn 32, Number 9, September, 1994.

[9] Ravi Sandhu. Role-Based Access Control. In Advances in Computers, volume 46. Academic Press, 1998.

[10] The Object Management Group (OMG). Unified Modeling Language. Version 1.4, OMG, http://www.omg.org, September 2001.

ENFORCING INTEGRITY IN MULTIMEDIA SURVEILLANCE†

[2]**Naren B. Kodali,** [3]**Csilla Farkas** and [1]**Duminda Wijesekera**
[1]Center for Secure Information Systems, [2]George Mason University. [3]Department of Computer Science and Engineering, University of South Carolina..

Abstract: In this paper, we propose a secure distribution model for multimedia surveillance data, where different locations of the monitored facilities may have different security requirements. We propose a multilevel security framework, wherein the surveillance data streams are classified according to the sensitivity of the location of the surveillance devices, and users (guards) have corresponding security clearances. Guards monitor live multimedia feeds emanating from surveillance devices placed throughout the facility. Our position is that during normal mode of operation, guards are allowed to access only those multimedia streams for which they have the proper authorizations. However, in an emergency, guards may receive pre-computed emergency instructions and/or cover stories. We show how to compose multilevel secure SMIL documents, compute views for each security classification, enforce integrity, confidentiality and access control, and deliver the secure views to handheld devices while

Keywords: Physical surveillance, SMIL, Secure multimedia, Multi Level Security

1. INTRODUCTION

Modern physical surveillance and monitoring systems depend on electronic instruments [12,13,14] where monitored areas such as command and control centers and missile storage facilities of a military base or traffic controller units of airports

† This work was partially supported by the National Science Foundation under grants CCS-0113515 and 0112874.

are accessible only to predefined groups of people. It naturally follows that disclosure of live surveillance records of such facilities must follow the access restrictions of the physical facilities. For example, in an airport location, passengers and employees can enter common areas, like ticketing and waiting areas. However, secured areas, like luggage transport, are available for airport employees only. The highest security areas, such as the air traffic control room, are accessible to specialized personnel only. Our aim is to ensure that people who are not authorized to access a physical location should not be able to view the surveillance data of that location. We also ensure that the integrity of the data is preserved during transit. However, during emergency operations controlled dissemination of sensitive data may become necessary in order to obtain support services and/or to prevent panic. It has been shown, that during crisis people require clear instructions for cooperation. However, these instructions should not release unauthorized information or reveal the existence of such information. Therefore, it is necessary to develop methods and tools to allow selective access to surveillance feeds during normal and emergency operations. This dual-mode operation calls for issues of integrity to be consummately addressed. This paper proposes a framework to do so using appropriately secured *Synchronized Multimedia Integration Language (SMIL)* [3] formatted multimedia compositions.

Our proposal is to integrate monitoring and communication in a secure surveillance system that enforces access control restrictions on datwhile providing maximum availability. Our main contribution is the development of a framework to express multimedia compositions with their rich runtime semantics, techniques to enforce integrity and access control, and the use of cover stories to disseminate relevant material to users with lower clearance levels during emergencies. For simplicity, we assume a multilevel security classification of physical areas and their corresponding surveillance data. People accessing these facilities have varying levels of security clearances. Employees and visitors are allowed to enter or view the surveillance feeds of a particular location (e.g., via broadcasts) only if they have appropriately cleared. We enforce that requirement during normal operations for guarding personnel. The main difference between a traditional Multilevel Secure (MLS) system and MLS for live surveillance feeds is that surveillance systems require the appropriate dissemination of classified information. We propose that our multimedia surveillance system be equipped with a semantically rich, pre-orchestrated multimedia cover story repository, so that in emergencies cover stories can be released to subject with lower levels of clearances.

Our model provides an integrated solution to manage surveillance devices and to collect audiovisual evidence for forensic analysis. We use an XML-based multimedia composition language referred to as SMIL, adapted for multi-level physical surveillance. The reason for selecting XML is guided by recent industrial trends. First, many browsers and off-the-shelf display and communication devices are becoming XML compliant [12,14]. Second, mobile communication and usage of XML formatted data becomes faster and more prevalent [11,12] than in the past. Third, toolkit support is available to integrate XML compliant services [10,13,14], Therefore, with the right technological framework, our solution is portable to a wide range of general-purpose mobile multimedia devices such as those available in automobile navigation systems and hand-held PDA's.

The multimedia community has its own XML-like language known as SMIL [3] to compose presentations using stored and/or live sources. Unlike XML tags, SMIL associates explicit meaning to some of its tags, which is crucial to ensure that the semantics of the captured scene is appropriately presented at the display. Further, human perception associates a meaning to integrated audio-video sequences. Maintaining data integrity is crucial in conveying the intended meaning. We enrich existing proposals [4,8] for securing multimedia data by incorporating operational semantics of information. We show how non-emergency and emergency operations of a MLS facility can be composed as a SMIL document enriched with proposed extensions. We take such a composition and construct views authorized for different security classes. We refer to these constructs as MLS normal forms of a SMIL document with appropriate security decorations. Then, given the runtime characteristics of an operational platform, we show how to generate an view appropriate for that runtime, which we call a display normal form of a SMIL document. We then encrypt media streams and transmit them to intended recipients under normal and emergency operating conditions.

The rest of the paper is organized as follows. Section 2 provides our running example used to explain the application scenario and our solution. Section 3, describes related work. Section 4 describes SMIL. Section 5, describes the required static preprocessing, runtime activities including encryption and resource management are explained in Section 6. Section 7 concludes the paper.

2. RUNNING EXAMPLE

Figure 1 shows a hypothetical research facility with varying levels of sensitivity. Assume that the area enclosed by the innermost rectangle ABCD contains weapons with highest degree of sensitivity and is accessible (and therefore guarded by) personnel with the highest level of clearance, say top secret (TS). The area between the rectangles PQRS and ABCD is classified at medium level of sensitivity and therefore requires personnel with secret (S) security clearances. The area external to PQRS contains least sensitive material, and can be accessed by unclassified personnel, like visitors and reporters. We classify the areas into *Top-Secret (TS),* *Secret (S)* and *Unclassified (UC)* security levels with application domains, e.g., *Dom* as categories. Security labels form a lattice structure. For simplicity, we omit the application domain and use *TS, S,* and *UC* as security labels. The area inside ABCD is TS, the area inside of PQRS, but outside of ABCD is S, and the area outside

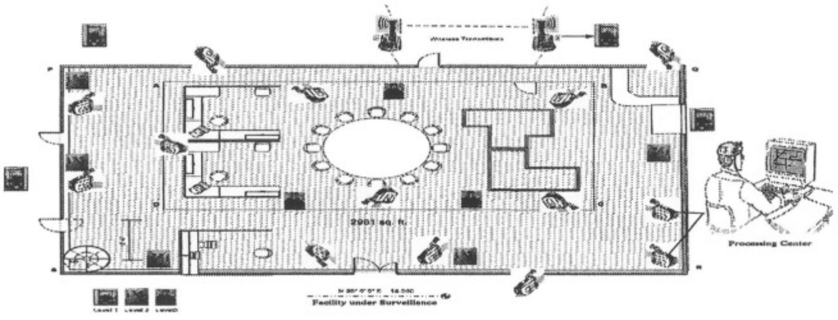


Figure 1: An Example MLS-Secure Facility

PQRS is UC. Employees, guards, support services personnel, and general public have *TS* • *S* • *UC* clearances, where • corresponds to the dominance relation defined in MLS systems. As depicted in Figure 1, an area with higher level of sensitivity is a sub-part of areas with all lower levels of sensitivities. Therefore, a guard with top-secret clearance may be used in the secret or unclassified area, but not vice versa. For electronic surveillance purposes, cameras and microphones are situated throughout the facility. Multimedia streams emanating from these devices are used to continuously monitor the facility. We propose a design where all multimedia data is transmitted to a centralized control facility and then directed to handheld devices of appropriate security personnel.

3. RELATED WORK

A distributed architecture for multi-participant and interactive multimedia that enables multiple users to share media streams within a networked environment is presented in [1], In this architecture, multimedia streams originating from multiple sources can be combined to provide media clips that accommodate look-around capabilities. MLS systems provide controlled information flow based on the security classification of objects (e.g., data items) and subjects (e.g., users) of the MLS system (e.g., applications running in behalf of a user) where data is allowed to flow only from low security levels to higher security levels [15]. Although our approach to provide controlled information flow in real-time multimedia systems is based in concepts similar to MLS, the developed methods and techniques are also applicable in other security models, like Role-Based or Discretionary Access Control models [15,16]. Regulating access to XML formatted text documents has been actively researched in the past few years offering many solutions. Bertino et al. [4,5], have developed Author-X, a Java based system to secure XML documents that enforces access control policies at various granularities and corresponding user credentials.

Damiani et al. [6,7] developed an access control model where the tree structure of XML documents is exploited using XPATH expressions at different levels of granularity. They also propose an access control model [7] with complex information filters using stereo video graphics (SVG) to render maps of physical facilities for controlled dissemination of sensitive data within an image. Bertino at al. [4] provides a security framework to model access control in video databases. They provide security granularity, where objects are sequences of frames or particular objects within frames. The access control model is based on he concepts of security objects, subjects, and the permitted access modes. The proposed model provides a general framework of the problem domain, although no explanation is offered as to how access control objects to be released are formalized and enforced.

4. SMIL

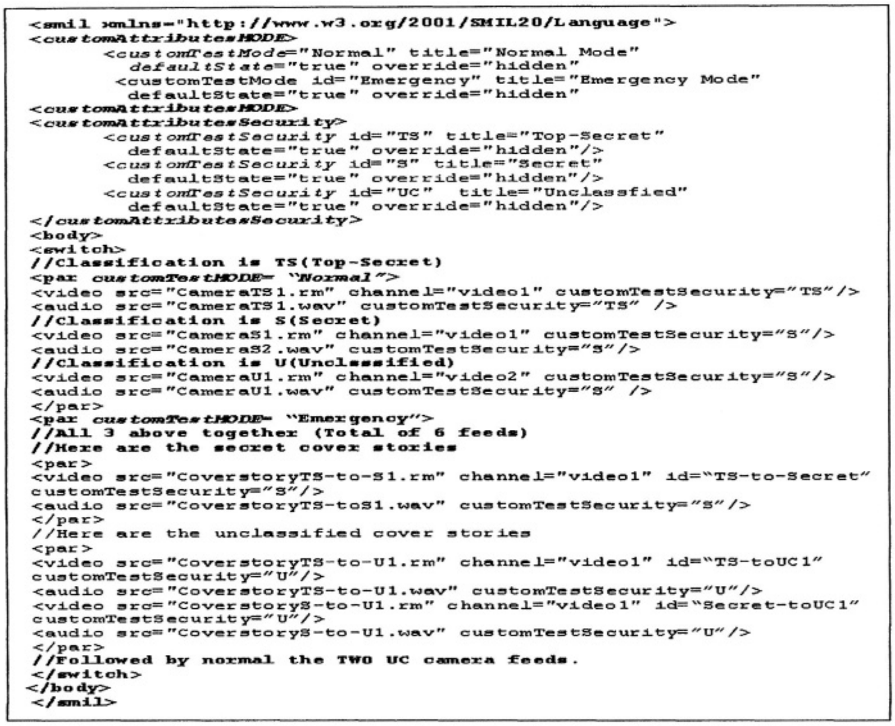
SMIL [3] is an extension to XML developed by W3C to author presentations, allowing multimedia components such as audio, video, text and images to be integrated and synchronized. SMIL has syntactic constructs for timing and synchronization. In this section, we explain those SMIL constructs, that are relevant and how they can be used to specify a multimedia document satisfying the application needs stated in Section 2.

SMIL constructs for synchronizing media are <seq>, <excl> and <par>. They are used to hierarchically specify synchronized multimedia compositions. The <seq> element plays the child elements one after another in the specified sequential order. <excl> specifies that its children are played one child at a time, but does not impose any order. The <par> element plays all children elements as a group, allowing *parallel* play out. For example, the SMIL specification *<par><video src=camera1><audio src = microphone1></par>* specify that media sources camera1 and microphone1 are played in parallel.In SMIL, the time period that a media clip is played out is referred to as its *active duration.* For parallel play to be meaningful, both sources must have equal active durations. The <*switch*> construct allows one to switch between alternatives compositions listed among its components. These alternatives are chosen based on the values taken by some specified attributes. For example, *<switch> <audio src="stereo.wav"* ***systemBitrate≥25><audio*** *src="mono.wav" systemBitrate* ***≤ 25></switch>*** plays stereo.wav when the SMIL defined attribute *systemBitrate* is at least 25 and *mono.wav* otherwise. We use this construct to specify our surveillance application. In order to do so, we define two custom attributes *customTestMode* that can take

values “normal” and “emergency” and *customTestSecurity* that take any value from (“TS”,“S”,“UC”). The first attribute is encodes the operating mode (normal or emergency) and the second encoding the security level of streams (top secret, secret or unclassified). SMIL also requires that every application-defined attribute (custom attribute in SMIL terminology) have a title and a default value.
Figure 2 shows a simplified example of a SMIL specification for surveillance. The custom attribute *customTestMode* has values “Normal” and “Emergency”. Since the value of *customTestMode* is *hidden,* this attribute in each corresponding stream cannot be changed. The second part of the file consists of a switch statement where media streams connected by <par> constructs. Notice that the <switch> statement consists of two sections where first begins with the line <par customTestMODE= “Normal”> and the second begins with the line <par customTestMODE= “Emeregency”>. That specifies that the streams inside be shown under normal and emergency operating conditions. In this example, each area has a camera and a microphone to record audio and video streams to be transmitted to appropriate guards. They are named CameraTS1.rm, CamerU1.wav etc. The security classification of each source is identified by the application defined SMIL attribute *customTestSecurity.* For example, <video src=“CameraTS1.rm” channel=“video1” customTestSecurity=“TS”/> specifies that the video source named *CameraTS1.rm* has the TS security level. The intent being that this source is to be shown only to top-secret guards. As the second half of Figure 2 shows, there are three audio-visual cover stories named *CoverstoryTS-to-S1.rm* to *CoverstoryS-to-UC1.wav* are shown with the appropriate security level specified with the attribute customTestSecurity. The main composition is encoded using a <switch> statement that is to be switched based on the operating mode (normal or emergency).

5. STATIC PREPROCESSING TO ENFORCE MLS

We assume that the source document specifies the security label of each source and that MLS policies are used to ensure that guards are permitted to view only those multimedia sources that are dominated by the guards’ security clearances. For this, we preprocess a given MLS multimedia document and produce views that are permitted to be viewed by guards for each security classification. Then, we separately encrypt and broadcast multimedia documents for each category, to the appropriate locations by efficient use of bandwidth. In order to achieve this objective, we first transform every SMIL document with proposed security and mode attributes to three SMIL documents, where all security labels in each document consists of solely one *customTestSecurity* attribute, namely the one that is appropriate to be seen by guards with the label value. We now formally state and prove that this can be done for an arbitrary SMIL document with our security labels.



```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<customAttributesMODE>
      <customTestMode="Normal" title="Normal Mode"
        defaultState="true" override="hidden"
       <customTestMode id="Emergency" title="Emergency Mode"
        defaultState="true" override="hidden"
<customAttributesMODE>
<customAttributesSecurity>
      <customTestSecurity id="TS" title="Top-Secret"
        defaultState="true" override="hidden"/>
      <customTestSecurity id="S" title="Secret"
        defaultState="true" override="hidden"/>
      <customTestSecurity id="UC"  title="Unclassfied"
        defaultState="true" override="hidden"/>
</customAttributesSecurity>
<body>
<switch>
//Classification is TS(Top-Secret)
<par customTestMODE= "Normal">
<video src="CameraTS1.rm" channel="video1" customTestSecurity="TS"/>
<audio src="CameraTS1.wav" customTestSecurity="TS" />
//Classification is S(Secret)
<video src="CameraS1.rm" channel="video1" customTestSecurity="S"/>
<audio src="CameraS2.wav" customTestSecurity="S"/>
//Classification is U(Unclassified)
<video src="CameraU1.rm" channel="video2" customTestSecurity="S"/>
<audio src="CameraU1.wav" customTestSecurity="S" />
</par>
<par customTestMODE= "Emergency">
//All 3 above together (Total of 6 feeds)
//Here are the secret cover stories
<par>
<video src="CoverstoryTS-to-S1.rm" channel="video1" id="TS-to-Secret"
customTestSecurity="S"/>
<audio src="CoverstoryTS-toS1.wav" customTestSecurity="S"/>
</par>
//Here are the unclassified cover stories
<par>
<video src="CoverstoryTS-to-U1.rm" channel="video1" id="TS-toUC1"
customTestSecurity="U"/>
<audio src="CoverstoryTS-to-U1.wav" customTestSecurity="U"/>
<video src="CoverstoryS-to-U1.rm" channel="video1" id="Secret-toUC1"
customTestSecurity="U"/>
<audio src="CoverstoryS-to-U1.wav" customTestSecurity="U"/>
</par>
//Followed by normal the TWO UC camera feeds.
</switch>
</body>
</smil>
```

Figure2: SMIL Specification of Figure 1

Definition 1 (MLS Normal Form)

We say that a SMIL specification S is in Multi Level Secure Normal Form (MLSNF) if it is of one of the following forms:

1. It is of the form <par> Cts(S) Cs(S) Cu (S) Cud(S) Cod</par> where all attributeTestSecurity attributes in Cts(S), Cs(S), Cu(S) are respectively TS, S and U. In addition, Cud(S) has no attributeTestSecurity and Cod(S) has two different values set for attributeTestSecurity.
2. It is of the for <par> Cts(S) Cs(S) Cu (S) Cud(S) Cod(S)</par> with one or two components of <par> may be missing. Here Cts(S), Cs(S) and Cu(S), Cud(S) Cod(S) satisfy requirements stated above.
3. It is of the form Cts(S), Cs(S), Cu(S), Cud(S), Cod(S) where Cts(S), Cs(S), Cu(S), Cud(S) and Cod(S) satisfy requirements stated above.

We say that Cts(S), and Cs(S) and Cu(S) are respectively the top secret, secret and unclassified views of the specification S. Cud(S) is the view with missing security classifications and Cod(S) is the view with contradictory security classifications.

As stated in Definition 1, a SMIL specification in MLSNF is one that is parallel composition of at most three specifications, where each specification belongs to one security class, that are said to be the views corresponding to the respective security classes. The latter two cases are degenerate cases of case 1 where one or more views of the specification become null.

In attempting to create views from an arbitrary SMIL document, one encounters two undesirable situations. The first is the missing security classifications resulting in a non-null Cud(S) or a contradictory security classification due to over specification. An example under specified SMIL specification is <audio src= "myAudio.wav">, and an example contradictory specification is <video src= "myMovie.rm" attributeTestSecurity=TS attributeTestSecurity=S>. Thus, it is tempting to avoid such situations by applying completeness and conflict resolution policies [17] designed to be used in XML formatted texts and databases. Because SMIL hierarchies are not due to inheritances and instead they are syntactic constructs for media synchronization, blindly applying such policies to resolve under and over specification of SMIL documents destroys the synchronized play out semantics of media streams. Here, we use the neutral policy of discarding under and over specified fragments Cud(S) and Cod(S) of a SMIL specification S. We now give an algorithm that transforms a given SMIL document into its MLSNF.

Algorithm 1:toMLSNF (Conversion to a Normal Form)

1. If S is a media stream (such as <video ...> or <audio ...>) with possibly an attributeTestSecurity attribute. Then:
    a. If attributeTestSecurity=TS, then Cts(S)=S, Cs(S)=ϕ and Cu(S)=ϕ and Cu(S)=ϕ.
    b. If attributeTestSecurity=S, then Cts(S)=ϕ , Cs(S)=S, and Cu(S)=ϕ.
    c. If attributeTestSecurity=U, then Cts(S)=ϕ , Cs(S)= ϕ, and Cu(S)=S.
    d. If attributeTestSecurity does not exists in S, then Cts(S)=ϕ , Cs(S)=ϕ, Cu(S)=ϕ and Cud(S)=S Cod(S)=ϕ.
    e. If there is more than one instance of attributeTestSecurity in S then Cts(S)=ϕ , Cs(S)=ϕ, Cu(S)=ϕ and Cud(S)= ϕ Cod(S)=S.
2. If S is <seq>S1 S2</seq> then,
    a. Cts(S)=<seq>Cts(S1) Cts(S2) </seq>
    b. Cs(S)=<seq>Cs(S1) Cs(S2) </seq>
    c. Cu(S)=<seq>Cu(S1) Cu(S2) </seq>
    d. Cud(S)=<seq>Cud(S1) Cud(S2) </seq>
    e. Cod(S)=<seq>Cod(S1) Cod(S2) </seq>
3. Similarly when, S is <par>S1 S2</par> , and S is <switch>S1 S2</switch>

However, if either of Cx(Si) are empty for some x ε{TS,S,U,UD,OD} and i ε{1,2}, then Cx(Si) in the right hand sides above must be substituted by NULL(Si) where NULL(Si) is defined as <type src=empty.type, attributeTestSecurity=Y, dur=Z type> where Z and Y are respectively durations and the *attributeTestSecurity* attribute values appearing in Si.

Then let MLSNF(S) = <par>Cts(S) Cs(S) Cu(S) Cud(S) Cod(S)</par>.

We now have to ensure that Algorithm 1 preserves semantics. That is, top secret, secret and unclassified viewers of a specification S will view Cts(S), Cs(S) and Cu(S) respectively. This proof is standard, if we have a formal operational semantics for SMIL. While providing such semantics is not difficult, it does not exist yet. Therefore, while we are in the process of developing formal semantics for SMIL, we provide a preliminary operational semantics for the purposes of showing that our algorithms work as expected.

5.1 A Preliminary Operational Semantics for SMIL

In this section, we provide a simple operational semantics for media streams and SMIL documents constructed using <par>, <seq> and <switch> commands. The sole objective of this exercise is to show that Algorithm 1 transforms a SMIL document to a collection of ones that remain invariant with respect to this semantics. The latter is referred to as semantic equivalence [18]. Following customary practices in programming language semantics, our operational semantics and the proof of semantic equivalence is inductive in nature. Our semantics is only applicable to our application scenario and syntactic constructs, and its extension to other purposes and constructs form our ongoing work.

Definition 2 (Timed Display Instance)

We say that a quadruple (S, T-begin, T-end, Security Set) is a timed display instance provided that:

1. S is a basic media element with a finite active duration **D≥0.**
2. T-begin ≦ T-end are arithmetic expressions of a single real variable **t** satisfying T-end=T-begin + D.
3. Security set a subset of {TS, S, U} consisting of *attributeTestSecurity* attribute values of S.
4. We say that a set of timed display instances is a timed display set provided that there is at least one timed display element with t as its T-begin value.
5. Taken as expressions containing the variable t, the smallest T-begin value of a timed display set is said to be the *origin* of the timed display set. We use the notation O(TDI) for the origin of the timed display set TDI.
6. Taken as expressions containing the variable t, the largest T-begin value of a timed display set is said to be the *end* of the timed display set. We use the notation E(TDI) for the end of the timed display set TDI.

The following two elements tdi_1 and tdi_2 are examples of timed display instances.

1. tdi-1 = (<video, src= "myVideo.rm", dur=5, attributeTestSecurity=TS>, t, t+7, {TS})
2. tdi-2 = (<audio, src= "myAudio.rm", dur=10, attributeTestSecurity=U>, t+7, t+17, {U})

Therefore, {tdi-1,tdi-2} is timed display set with its origin t and end t+17. The intent here is to consider TDI={tdi-1,tdi-2} as a possible playout of the SMIL specification <seq><video, src= "myVideo.rm", dur=5, attributeTestSecurity=TS>, <audio, src= "myAudio.rm", dur=10, attributeTestSecurity=U> </seq> that begin at an arbitrary but thereafter fixed time t and ends at t+17. Now we describe some algebraic operations on timed display sets that are necessary to complete the definition of our operational semantics of SMIL. The first is that of *origin substitution* defined as follows.

Definition 3 (Algebra of Timed Display Sets 1: Substitution)

Suppose TDS is a timed display set with the formal time variable t and s is any arithmetic expression possibly containing other real valued variables. Then TDS(s/t) is the notation for the timed display set obtained by syntactically substituting all timing values (that is T-begin and T-end values) of elements of TDI.

For the example TDI given prior to Definition 3, TDI(2t+7/t) consists of (tdi-1(2t+7/t),tdi-2(2t+7/t)} where tdi-1(2t+7/t) and tdi-2(2t+7/t) are defined as:

1. tdi-1(2t+7/t) = (<video, src= "myVideo.rm", dur=5, attributeTestSecurity=TS>, 2t+7, 2t+21, {TS})
2. tdi-2(2t+7/t) = (<audio, src= "myAudio.rm", dur=10, attributeTestSecurity=U>, 2t+21, 2t+31, {U})

The reason for having Definition 3 is that in order to provide formal semantics for the <seq> operator, it is necessary to shift the second child of the <seq> by the time duration of its first child and repeat this procedure for all of <seq>'s children. To exemplify the point, the first example the TDI={tdi-1,tdi-2} is infact {tdi-1} U TDI'(t+7/t) where TDI' is given by tdi' = (<audio, src = "myAudio.rm", dur=10, attributeTestSecurity = U>, t, t+10, {U}). We are now ready to obtain operational semantics for SMIL specifications.

Definition 4 (Basis Mapping)

Suppose M is the set of basic media elements of S. Then any mapping [[ ]] from M to a set of Timed Display Instances TDI is said to be a basis mapping for a denotation iff all T-begin elements of M have the same value t, where t is a real variable. Then we say that [[ ]] is a basis mapping parameterized by t.

Lemma 1 (Existence of basis mappings) A set M of basic media streams with time durations has a basis mapping.

Proof:

For each media stream m= <type, src= "...", dur=value, attributeTestSecurity= "..." type>, in M, let [[M]] map to (m, t, t+value, {Att Values}). Then [[ ]] is a basis mapping.

We now use a basis mapping to define operational semantics of any SMIL specification S as follows.

Definition 5 (Operational Semantics for SMIL) Suppose S is a SMIL specification and [[ ]] is a basis mapping for the basic media elements B of S with the formal parameter t. Then we inductively extend [[ ]] to S as follows.

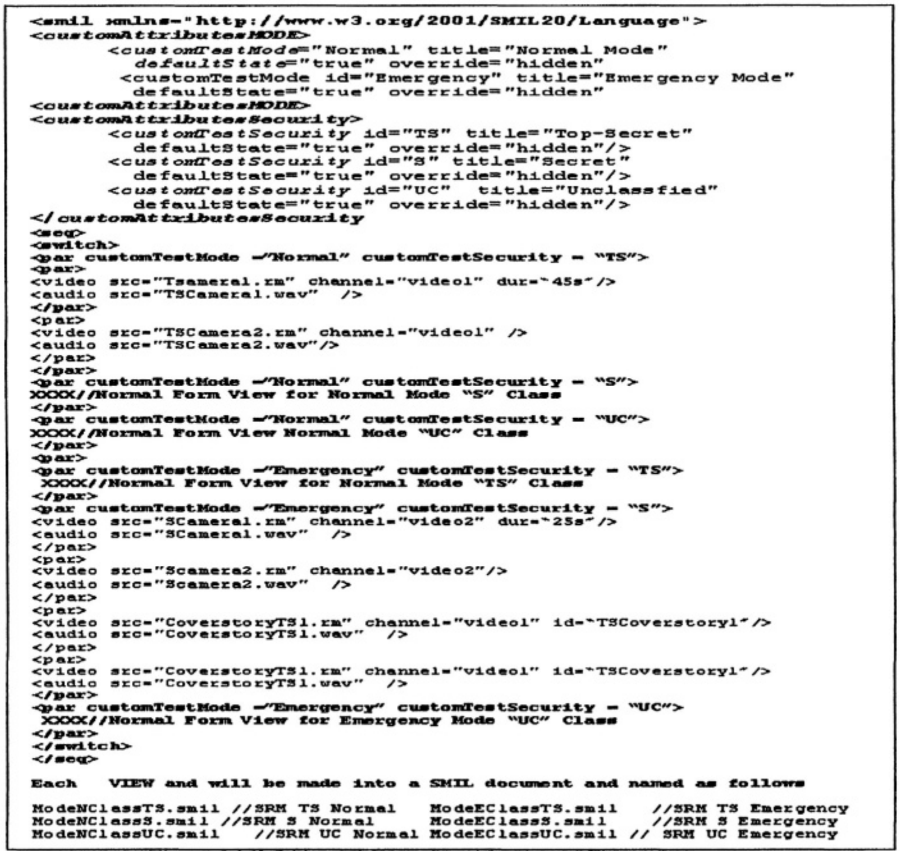
1. **[[Null]] = Φ.**
2. [[<seq> S1 S2</seq>]] = [[S1]] U [[S2]](end([[S1]])/t)
3. [[<par> S1 S2</par>]] = [[S1]] U [[S2]].

4. [[<switch> S1 S2 </switch>]] = [[S1]] if S1 satisfies the attribute of the switch.
   = [[S2]] otherwise if S2 satisfies the attribute of the switch.
   = **Φ** otherwise.

We now say that the extended mapping [[ ]] is a semantic mapping parameterized by t. To the best of our knowledge, the informal semantics given the SMIL specification is abstractly captured by our operational semantics provided we can evaluate the *attribute of the switch.* This can be easily formalized using customary practices of program language semantics, and is therefore omitted here for brevity.

On rewriting the example in Figure 2 in the MLS Normal form we create at the different views for each of the following cases each represented as a separate SMIL document. In the Figure 3 below, we have the format of such a specification denoting the entire structure of a "Top-Secret" view in the normal mode and a "Secret" view in the emergency mode.

6. RUNTIME BEHAVIOR OF THE MLS SYSTEM.



```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<customAttributesMODE>
      <customTestMode="Normal" title="Normal Mode"
        defaultState="true" override="hidden"
       <customTestMode id="Emergency" title="Emergency Mode"
        defaultState="true" override="hidden"
<customAttributesMODE>
<customAttributesSecurity>
      <customTestSecurity id="TS" title="Top-Secret"
        defaultState="true" override="hidden"/>
      <customTestSecurity id="S" title="Secret"
        defaultState="true" override="hidden"/>
      <customTestSecurity id="UC"  title="Unclassfied"
        defaultState="true" override="hidden"/>
</customAttributesSecurity
<seq>
<switch>
<par customTestMode ="Normal" customTestSecurity = "TS">
<par>
<video src="Tsameral.rm" channel="videol" dur="45s"/>
<audio src="TSCameral.wav"  />
</par>
<par>
<video src="TSCamera2.rm" channel="videol" />
<audio src="TSCamera2.wav"/>
</par>
</par>
<par customTestMode ="Normal" customTestSecurity = "S">
XXXX//Normal Form View for Normal Mode "S" Class
</par>
<par customTestMode ="Normal" customTestSecurity = "UC">
XXXX//Normal Form View Normal Mode "UC" Class
</par>
<par>
<par customTestMode ="Emergency" customTestSecurity = "TS">
 XXXX//Normal Form View for Normal Mode "TS" Class
</par>
<par customTestMode ="Emergency" customTestSecurity = "S">
<video src="SCameral.rm" channel="video2" dur="25s"/>
<audio src="SCameral.wav"  />
</par>
<par>
<video src="Scamera2.rm" channel="video2"/>
<audio src="Scamera2.wav"  />
</par>
<par>
<video src="CoverstoryTS1.rm" channel="videol" id="TSCoverstoryl"/>
<audio src="CoverstoryTS1.wav"  />
</par>
<par>
<video src="CoverstoryTS1.rm" channel="videol" id="TSCoverstoryl"/>
<audio src="CoverstoryTS1.wav"  />
</par>
<par customTestMode ="Emergency" customTestSecurity = "UC">
 XXXX//Normal Form View for Emergency Mode "UC" Class
</par>
</switch>
</seq>

Each   VIEW and will be made into a SMIL document and named as follows

ModeNClassTS.smil //SRM TS Normal   ModeEClassTS.smil   //SRM TS Emergency
ModeNClassS.smil //SRM S Normal     ModeEClassS.smil    //SRM S Emergency
ModeNClassUC.smil   //SRM UC Normal ModeEClassUC.smil // SRM UC Emergency
```

Figure 3:MLS Normal Form Specification of Figure 2.

In the most general case, a SMIL specification in MLSNF is of the form <par> Cts Cs Cu Cod Cud </par> where Cts Cs Cu Cod and Cud respectively have top secret, secret, unclassified, over specified and under specified security levels. How one resolves under specification and over specification is a matter of policy, and is not addressed in this paper. Independently, Cts, Cs, Cu are to be shown to guard with top secret, secret, and unclassified clearances. In addition, in order to respond to emergencies, these specifications have a mode switch encoded using the custom attribute *attributeTestMode.* As seen in Figure 3, this attribute is to be evaluated at the beginning of a <switch> statement. That is unsatisfactory for intended purposes, because after this switch statement is executed, the operating mode could vary many times. Because the <switch> is evaluated only once, the SMIL specification is now oblivious to such changes in application situations. In the next section, we show how to rewrite a SMIL document with one <switch> statement for changing a mode to another SMIL document that evaluates the *attributeTestMode* at regular intervals. Although in theory any system could switch its operating mode in an arbitrarily small time intervals, practical considerations limits this interval. This minimum switching granularity may depend upon many parameters such as hardware, software. Therefore, given a switching delay D, we rewrite the given SMIL document so that the mode attribute *attributeTestMode* re-evaluated every D time units. How that is done is discussed in the next section.

6.1 Informal Display Normal Form

The following SMIL specification in Figure 4 has the same structure as the specification in Figure 2 and Figure3.If we want to break up this specification so that the *attributeTestMode* is tested each D units of time and the switch reevaluated, then the code can be translated as follows (right hand side of Figure 4).

```
S1 =<switch>
          <par  attributeTestMode=
"normal">
          XX
          </par>
          <par attributeTestMode=
"emergency">
          </par>
     </switch>
```

```
S2 = <par dur=D,
repeatCount="indefinite">
          <switch>
            <par
attributeTestMode="normal">
            XX
            </par>
            <par
attributeTestMode="emergency">
            YY
            </par>
          </switch>
      </par>
```

Figure 4: Mode Evaluation Semantics.

Notice that the outer <par> construct specifies that enclosing specification be executed for duration of D time units and repeated indefinitely. However, the outer <par> construct has only one element, namely the switch. Therefore, the <switch> construct is executed for infinitely many times, and each time the attributeTestMode is tested. Given a SMIL specification with the *attributeTestMode* specified in the form where the switch is reevaluated every D time units is said to be in display normal for the attribute *attribbuteTestMode* and time duration D. We have now informally shown that every SMIL document where the *attribbuteTestMode* is used in the stated form can be translated into its display normal form.

We stress the informal nature of our argument because of our commitment to the specified operational semantics. Our translation into display normal form is not semantically equivalent under semantics provide in definition 6. However, we intend to enhance the semantics so that this construction will preserve semantic equivalence in our future work.

6.2 Dynamic Runtime Activity.

As explained, any given SMIL specification S for surveillance is statically translated into its MLS normal form MLSNF(S). Then, when the runtime provides D, MLSNF(S) is translated into its display normal form, say DNF(MLSNF(S),D). Then the runtime takes each the set of streams within the switch that has duration of D, evaluates the switch, and depending upon the mode, encrypts and transmits either the streams corresponding to normal operating mode or those that correspond to the emergency operating mode. The figure 5 shows the display normal form for the SECRET VIEW and briefly discusses mode evaluation procedures.

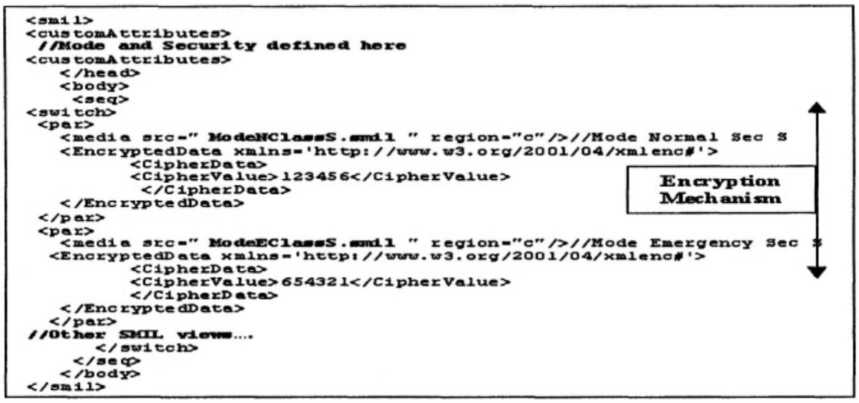
```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<customAttributesMODE>
      <customTestMode="Normal" title="Normal Mode"
        defaultState="true" override="hidden"
        uid="ControllerChoice" />
      <customTestMode id="Emergency" title="Emergency Mode"
        defaultState="false" override="hidden"
        uid="ControllerChoice" />
</customAttributesMODE>
<customAttributesSecurity>
      <customTestSecurity id="TS" title="Top-Secret"
        defaultState="true" override="hidden"/>
      <customTestSecurity id="S" title="Secret"
        defaultState="true" override="hidden"/>
      <customTestSecurity id="UC"  title="Unclassfied"
        defaultState="true" override="hidden"/>
</customAttributesSecurity>
  <body>
    <switch>
     <ref src="ModeNClassS.smil"   customTestMode ="Normal"  customTestSecurity ="S" />
    <ref src="ModeNClassS.smil"   customTestMode ="Emergency" customTestSecurity ="S" />
   </switch>
  </body>
</smil>
```

Figure 5: The Secret View of the SMIL Document of Figure3 in Display Normal Form.

The initial setting of the mode is taken from the value of the defaultState attribute. If no default state is explicitly defined, a value of **false** is used. The URI (Controller Choice) is checked to see if a persistent (Normal, Emergency) is defined instead of the default. As with predefined system test attributes, this evaluation will occur in an implementation-defined manner. The value will be (re) evaluated dynamically, as described above

6.4. The Confidentiality and Integrity Enforcing Encryption Model

Mobile handheld viewing devices [14] that have embedded SMIL players are the recipients. A smartcard, which enforces access control, is embedded into the display device [9]. Each display device has a unique smartcard depending on the classification of the guard that utilizes it and his classification and any other rules set by the controller. A decryption key associated with the privileges of the guard is also embedded in the smartcard. When a display device receives an encrypted SMIL document, the smartcard decrypts the appropriate segment depending on the available decryption key. We encrypt each view in the document as shown in Figure 6 with a unique Symmetric Key using the standard XML Encryption Specification. An inbuilt Cryptix Parser that is programmed in firmware (or in software) to handle the decryption process would enable selective decryption of the appropriate view based on the access privileges as defined in the smartcard.



```
<smil>
<customAttributes>
 //Mode and Security defined here
<customAttributes>
   </head>
   <body>
    <seq>
<switch>
 <par>
   <media src=" ModeNClassS.smil " region="c"/>//Mode Normal Sec S
   <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'>
         <CipherData>
         <CipherValue>123456</CipherValue>
          </CipherData>
   </EncryptedData>
 </par>
 <par>
   <media src=" ModeEClassS.smil " region="c"/>//Mode Emergency Sec S
  <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'>
         <CipherData>
         <CipherValue>654321</CipherValue>
         </CipherData>
   </EncryptedData>
  </par>
//Other SMIL views....
      </switch>
    </seq>
   </body>
</smil>
```

Encryption Mechanism

Figure 6: Display view with QoS and Encryption Parameters.

The Figure 6 depicted above shows the encryption tags applied to the display normal form of the secret view [Figure5] to achieve confidentiality. With encryption, we guarantee that nobody tampers the stream in transit even if there is mediate stream acquisition.

7. CONCLUSIONS

We have provided a model for audio-video surveillance of multi-level secured facilities during normal and pre-envisioned emergencies. We enhanced the SMIL specification with security decorations to satisfy MLS constraints during normal operations and provide controlled declassification during emergencies while maintaining the integrity and confidentiality of the data in transit. Then we showed how to transform such a SMIL composition to its MLS normal form that preserves runtime semantics intended by SMIL constructs while creating views compliant with

MLS requirements. Given the delay characteristics of a runtime, we show how to transform a SMIL document in MLS normal form so that the operating mode can be switched with the minimum delay while respecting runtime semantics of SMIL. Our ongoing work extends this basic framework to incorporate richer multimedia semantics as well as diverse security requirements such as non-repudiation of media evidence, and incorporate them in SMIL metamodels. Finally, this paper focuses on confidentiality issues. However, it is also important to address source authentication issues, which along with the development of a prototype system are part of our future work.

REFERENCES

[1] B. K. Schmidt "An Architecture for Distributed, Interactive, Multi-Stream, Multi-Participant Audio and Video". Technical Report No CSL-TR-99-781, Stanford Computer Science Department.

[2] D. Wijesekera and J.Srivastava, "Quality of Service Metrics for Multimedia" in Multimedia Tools and Applications, Vol 2, No3 1996, pp. 127-166.

[3] J. Ayers et al. "Synchronized Multimedia Integration Language (SMIL 2.0)". World Wide Web Consortium (W3C). http://www.w3.org/TR/smil20/ (August 2001).

[4] E.Bertino, M.A. Hammad ,W.G. Aref and A.K. Elmagarmid "An access control model for video database systems" in Conference on Info and Knowledge Management, 2000

[5] E. Bertino, E. Ferrari S. Castano "Securing XML Documents with Author-X" in IEEE Internet Computing, vol 5,no3 May/June 2001

[6] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, "Securing XML Documents," in Proc. of EDBT2000), Konstanz, Germany, March 27-31, 2000..

[7] E. Damiani, S. De Capitani di Vimercati, E. Fernandez-Medina, P. Samarati "An Access Control System for SVG Documents" in Proc. IFIP WG11.3 , DBSEC '02.

[8] E.Damiani, S. De Capitani di Vimercati "Securing XML-Based Multimedia Content" Security and privacy in the Age of Uncertainty, Pages 61-72.

[9] N.Kodali, D.Wijesekera"Regulating Access to SMIL formatted Pay-per-view Movies" in Workshop on XML Security, 2002.

[10] The Triclops Camera at http://www.ptgrey.com/products/triclopsSDK/triclops.pdf

[11] Alpha works Suite :XML http://www.alphaworks.ibm.com/xml

[12] Spymake Integrated Surveillance Tools
at http://www.spymakeronline.com/catalogue/surveillance.html

[13] Mobile VCMS™ - Field Data Collection System at http://www.acrcorp.com:8080/acr_vcms/mobile

[14] E. Ekudden, U.Horn, M.Melander and J.Olin"On-demand mobile media—A rich service experience for mobile users" Erricson.com White papers.

[15] D. Elliot Bell and Leonard J.LaPadula Secure computer systems: Mathematical foundations. Technical Report 2547 (Volume I), MITRE, March 1973.

[16] Sandhu, R. S., Coyne, E. J., Feinstein, H. L. and Youman, C. E. "Role-based access control models" In IEEE Computer, 1998

[17] S. Jajodia, P. Samarati, V. S. Subrahmanian, "A logical language for expressing authorizations," Proc. IEEE Symp. on Security and Privacy, Oakland, 1997, pages 31-42

[18] K. Mulmuley "Full Abstraction and Semantic Equivalence," ACM Doctoral Dissertation Award 1986, The MIT Press, Cambridge. MA, London, England, 1987

A LEARNING-BASED APPROACH TO INFORMATION RELEASE CONTROL

Claudio Bettini
DICO, University of Milan, Italy, and
Center for Secure Information Systems, George Mason University, Virginia
bettini@dico.unimi.it

X. Sean Wang
Department of Computer Science, University of Vermont, Vermont, and
Center for Secure Information Systems, George Mason University, Virginia
xywang@cs.uvm.edu

Sushil Jajodia
Center for Secure Information Systems, George Mason University, Viginia
jajodia@gmu.edu

Abstract: Controlled release of information from an organization is becoming important from various considerations: privacy, competitive information protection, strategic data control, and more. In most organizations, data protection is afforded only by using access control. However, it can be argued that access control suffers from at least two problems. First, effective access control assumes a perfect categorization of information ("who can access what"), which is increasingly difficult in a complex information system. Second, access control is not effective against insider attacks, where users with legitimate access rights send out sensitive information, either with malicious intent or by accident. Information release control is viewed as complementary to access control, and aims at restricting the outgoing information flow at the boundary of information systems. This paper presents an architectural view of a release control system. The system emphasizes the role of automated learning for release control constraints. This has resulted from the realization that the most difficult task of effective release control is how the release control constraints are specified. In a learning-based system, data mining and machine learning techniques are employed to generate release control constraints from samples provided by the security officer. The system applies continuous learning to adjust the release control constraints to reduce both mistakenly released and mistakenly restricted documents. This paper also provides a specific example on how to learn keyword-based release control constraints.

Keywords: Data protection, Release control

1. INTRODUCTION

Release control refers to the process of checking the output data generated upon a user request to determine if the information is appropriate for release across a security boundary. Nowadays, organizations have vast amounts of information that is shared with other organizations or even the general public. Such sharing takes the form of public web pages, financial reports, technical white papers, biographies of personnel, etc. Furthermore, the knowledge workers of the organizations send out messages for collaboration purposes. Such information sharing needs to be done in a controlled fashion, taking into account security and other considerations.

For information security, a number of aspects have been considered in the literature. These aspects include (1) secure communication, (2) perimeter control, (3) reliable authentication, (4) authorization to information cells (such as files, relational tables, columns in tables, and XML nodes), and (5) partitioning of the information into cells. According to Wiederhold [Wie00], aspect (5) is often under-emphasized and requires a highly reliable categorization of all information to those cells. It is in the management of that categorization where many failures occur. In other words, if some information is miss-categorized, which is highly likely in a complex organization, it is possible for the sensitive information to be released to unauthorized users.

A more serious threat to sensitive information comes from careless or malicious *insiders,* individuals or organizational entities with authorized access to the system. This type of threats are potentially very damaging since the access control rules (on which organizations rely heavily) are not effective. Insider threats need to be countered using several different methods, from intrusion analysis to information forensics. An important tool to counter such threats is release control, which blocks the information at the *gate* from “inside” to “out side.”

In addition to the aforementioned security considerations, some legal concerns of information release need to be addressed. Depending on the category of information, an organization may wish to append disclaimers, copyright and other legal notices, and even watermarks. Release control can be used to address this problem.

Release control thus needs to become an important component for securing and managing information of an organization. From a technical perspective, the release control system is best done by separating "release control constraints", which state what need be controlled, from "checking algorithms", which monitor the outgoing data against these constraints. This separation allows more convenient management of the release control system to fit the ever-changing organizational security and legal needs, and allows more opportunities for the introduction of new checking algorithms.

In view of the aforementioned technical considerations, for an effective and efficient release control system, at least two issues need to be considered carefully. The first is how the release constraints are established and refined, and the second is how the checking of the outgoing information can be done efficiently and in a meaningful way.

In this paper, we present an architectural view of the release control system and then focus on the first issue. One relevant source for coming up with release constraints is clearly the data store. Indeed, in addition to the ability of simply adding release constraints derived from experience or high-level requirements, security officers may query the data store to determine what needs to be controlled at the release point. We call this the "manual" method. For example, access control rules are usually adopted to restrict the access to the data store [JSSS01]. When some data have restricted access, it is likely that the release of such data should be checked. In addition, information that might be inferred through integrity constraints from restricted data should also be automatically added to the release constraints store. Furthermore, data that are similar to such restricted data may also need to be checked at the release point. Due to the involved complexity of these tasks, "automated tools" are necessary.

We introduce an automated learning approach to help acquire release control constraints. In a learning-based approach, the security office can give an initial sample set of documents, including both "cleared for release" documents and "restricted" documents. The system will try to learn an initial set of release control constraints from the given sample set. As the time goes by, when more and more documents are released and restricted (some of the releasing and restricting are certified by the security officer), the learning process will periodically adjust the release control constraints to do a better job: reducing the mistakenly released documents as well as the mistakenly restricted documents. In this paper, we outline such a learning based system architecture, and provide a specific example for learning release control constraints.

The remainder of the paper is organized as follows. In Section 2, we give a formal foundation for specifying release control constraints. We also give a language that can be used to represent release control constraints on XML documents. In Section 3, we outline an architecture for release control and emphasize the role of learning in the whole architecture. In Section 4, we give a specific example of learning keyword-based release control constraints. We discuss related work in Section 5 and conclude the paper in Section 6.

2. FORMAL FRAMEWORK

For a uniform treatment, we assume the data to be checked before release is in XML format (the underlying data may be a table resulting from a database query, semistructured data or a full text document). The data are checked against a number of release constraints.

2.1 Release Constraints

A *release constraint* (*RC*) is a pair $\langle R, CI \rangle$, where *R* is a *matching rule* and *CI* a set of *controlled items.*

Each RC is evaluated against the data being released and it prevents its release if satisfied. Formally, each RC gives a mapping that assigns each document with a label in {**Restrict, Release**}. A document *Doc* can be released if for each release constraint *RC*, *RC*(*Doc*)=**Release**, i.e., in the data, the controlled items do not appear in the way given by the corresponding matching rules.

The set *CI* of controlled items is a set of tuples of the form $\langle A_1{:}a_1, A_2{:}a_2, \ldots, A_n{:}a_n \rangle$, where A_j are variable symbols and a_j their values. Values can range over several simple domains (including integer, string, Boolean, etc.) or even complex domains (admitting single-valued, multi-valued, or possibly structured attribute values). Variable symbols may actually denote attribute names as well as paths in the document structure. In the simplest case, variable symbols are omitted and each tuple is a list of keywords.

Syntactically, the attribute part *A* of each pair can be specified by an XPath expression, while the value part *a* can be specified by a regular expression if it is a string (e.g., to denote any word starting with a certain prefix) or, in the case it is a numerical value, it can be specified by a simple condition (*op k*) with $op \in \{<, =, >, \leq, \geq\}$ and *k* being a number (e.g., to denote all values greater than or equal to a certain constant).

The matching rule R specifies how a document should be checked against the controlled items. As an example, when the data to be checked are answers from a database query, they can be represented as a sequence of tuples. Then, a simple matching rule may check if one of these tuples contains the attribute values specified in one of the tuples in the set of controlled items. In the case of data containing unstructured text, the rule may specify, for example, that the words in each tuple of the set of controlled items should not appear together in any k-words portion of the text, where k is a parameter defined by the specific rule. In other words, the set of controlled items essentially lists the pairs attribute-value involved in a release constraint, while the matching rule specifies their relationships.

Example 1. Consider a corporate database that includes data about Employee's salaries and assume an internal policy prohibits the release of the salary of Mr. Woo. The administrator will set up a *RC* with *CI* = *{⟨Employee:Woo, Salary:75,000⟩}* and R equal to the simple rule on relational query results checking all the query results being released for tuples containing the attribute values specified in *CI*. In this case, the system will check each tuple returned, as part of a database query result, making sure the values *Woo* and *75,000* do not appear in it.

Note that this is a conceptual model of *RC*s. In practice, controlled items will have compact representations, in the form of SQL queries, general XQuery expressions including predicates, strings with metacharacters or other representations.

The above formal model can be easily extended to represent overriding constraints by allowing negative terms in control items (in the form $\neg A{:}a$). For example, in protection software, among all documents rejected because containing the word 'chip', those in which the occurrence of 'chip' is always part of the sequence 'chocolate chip' should be released. This can be modeled by the above extension.

2.2 A language for matching rules on XML documents

In this subsection, we focus on the problem of representing matching rule when the data to be checked in contained in XML documents. We design a language that has the following expressiveness properties, which we consider necessary for our task.

- The language should be able to express the maximum distance between all pairs of values as a pre-determined upper bound. In terms of XML this may be the number of edges in the XML tree that separate the nodes storing the values.

- The language should be able to express the presence of certain nodes (with particular labels, semantic tagging, degree, type, e.g., leaf, root, etc.) in the path between the pair of nodes for which we consider the distance in the XML structure.
- The language should be able to express relationships between nodes. Example of relationships include the difference in depth in the XML tree, the difference in the semantic tagging possibly attached to the nodes, the presence of a common ancestor with a specific tag, relationships between node attributes, etc.
- The language should be able to discriminate the order of the values in the XML structure. The order may be particularly relevant in overriding. In our chocolate-chip example above, the word 'chocolate' must appear before the word 'chip'.

Following the above requirements, we represent each matching rule as a conjunction of a *cardinality rule* and one or more *node-relation* rules.[1]

A *cardinality rule* has the form $NumVar \geq k$ where k is a positive integer and *NumVal* is a language keyword denoting the number of different values specified in the controlled items that should be observed in the document. Hence, a cardinality rule specifies that the number of these values must be at least k, not considering negative terms in the controlled items representing overriding conditions. If no cardinality rule is given, a default cardinality rule is assumed with $k=1$; this is the most conservative choice.

Example 2. Given a set of controlled items $(A_1:a_1, \neg A_2:a_2, A_3:a_4, A_4:a_4)$, a cardinality rule $NumVar \geq 2$ would restrict any document that would contain at least 2 of the positive controlled items; i.e., either $A_1:a_1$ and $A_3:a_3$, or $A_1:a_1$ and $A_4:a_4$ or $A_3:a_3$ and $A_4:a_4$, but not containing $A_2:a_2$. In the case of XML documents by containment of $A:a$ we mean the presence of a node/attribute identified by A and having text/value a.

A *node-relation* rule $NR(A_1:a_1, \ldots, A_k:a_k)$ is represented by a Boolean *node formula.* A *node formula* is recursively defined from a set of *atomic distance formulas* and *atomic node formulas* by applying the standard conjunction and negation operators as well as quantification on nodes. Existential quantification has the form: $\exists N \in P: F(N)$, where N is a node appearing in the path P of the XML tree and $F(N)$ is a node formula. Paths can be defined as explained below for *atomic distance formulas.* Since

[1] Alternative matching rules (disjunctive behavior) can be established by adding more release control rules with the same controlled items.

negation is allowed, universal quantification and disjunction are implicitly part of the language.

An *atomic distance formula* has the form:

- $|P_2| |P_1|$ *op* $|P_2|$, where P_1, P_2 are paths in the XML document tree and *op* is in $\{<,=,>,\geq,\leq\}$,
- $|P|$ *op* k, where *P* is a path in the XML document tree, and *k* is a non-negative integer.

The notation $|P|$ stands for the length of path *P* in the XML tree of the document. For example, the root node of an XML document is easily characterized by having a path of length *'0'*.

Paths *P* are either among $P(a_1),\ldots,P(a_k)$, which denote the paths from the root of the XML tree to the nodes associated with a subset of values $a_1,\ldots,a_k$ among the controlled items, or they can be paths obtained from them by applying the following operators:

- $P_1 \cap P_2$ (Path Intersection)
- $P_1 - P_2$ (Path Difference)
- ***Prefix**$_k$**(P)*** (Path Prefix of length *k*)
- ***Suffix**$_k$**(P)*** (Path Suffix of length *k*)

Each *atomic node formula* compares a node property with a constant value or with the property of a different node. The set of atomic node formulas includes the following:

- Comparison of attribute values
  - ***AttrValue**(N_1**,attrName)*** *op* ***AttrValue**(N_2**,attrName)***, where *op* is in $\{<,=,>,\leq,\geq,\neq\}$ (applies only if the attribute has a numeric value).
  - *AttrValue(N,attrName) op k,* where *k* is an integer and *op* is in $\{<,=,>,\leq,\geq,\neq\}$ (applies only if the attribute has a numeric value).
  - ***AttrValue**(N_1**,attrName)*** *rel* ***AttrValue**(N_2**,attrName)***, where *rel* is in $\{=,\neq,$ *substr,...*$\}$ (applies only if the attribute has a string value).
  - *AttrValue(N,attrName) rel string,* where *string* is any string and *rel* is in $\{=,\neq,$ *substr,...*$\}$ (applies only if the attribute has a string value). For example, using conjunction, this can be used to check if a set of nodes has the same attribute value.
- Comparison of node meta properties
  - ***Order**(N_1**)*** *op* ***Order**(N_2**)***, where *op* is in $\{<,=,>,\leq,\geq\}$ and *Order*() is not an attribute, but a function assigning a unique value to each node in the XML tree, according to a specific order (e.g., preorder). This may be used to check if a certain word appears before or after another one in the document.

- *Tag(N) rel string,* where *string* is any string and *rel* is in $\{=, \neq, substr, \ldots\}$. For example, using conjunction, this can also check if a set of nodes has the same tag <*Table*>.
- *Degree*(*N*) *op k,* where *Degree*(*N*) denotes the number of children of *N* in the tree, *k* is a non-negative integer and *op* is in $\{<,=,>, \leq, \geq, \neq\}$. As an example, a leaf node can be characterized by having degree '*0*'.

Example 3. The following matching rule
$\exists N \in Suffix_2(P(N_1))$:*AttrValue(N, "attrName") superstring "relString"* is satisfied if either the parent node or the parent of the parent node of N_1 has a value for the attribute *"attrName"* which contains the string *"relString"*. Note that N_1 is identified by testing whether it contains one of the values in the controlled items.

Note that paths in atomic distance formulas can be considered constants when a specific XML document is considered. This is different from nodes in atomic node formulas that can be both constants and variables. Indeed, they are constants if the nodes are among the ones corresponding to the values $a_1, \ldots, a_k$ in the controlled items, and variables otherwise. If they are variables they will appear in the node formula under quantification.

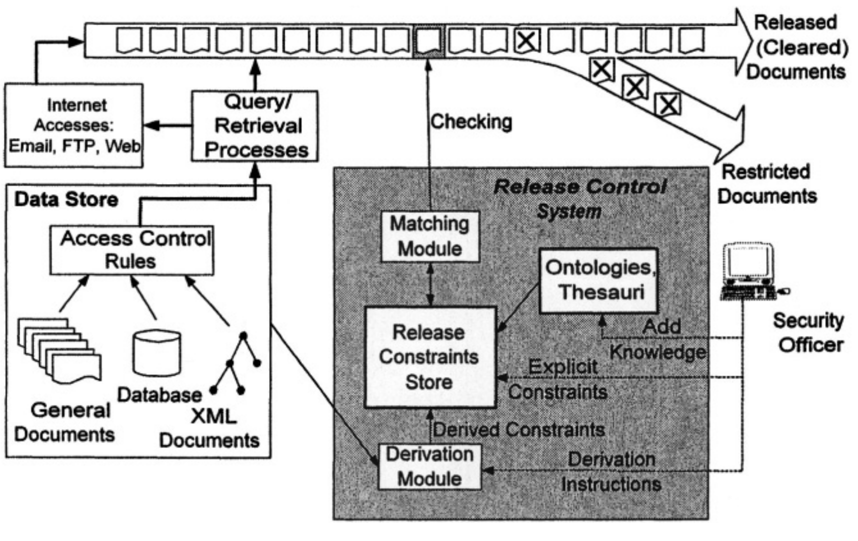


Figure 1. General architecture for release control.

3. RELEASE CONTROL ARCHITECTURE

The architecture we are proposing is based on three basic component: (i) the flow of documents that are going to be released, (ii) a Data Store from which the documents are extracted/derived, and (iii) a Release Control System monitored by a security officer. Figure 1 illustrates these components and their internal structure, ignoring at this level the machinery needed for the learning process.

3.1 The Basic Components

The Data Store can include different types of data sources like standard relational databases, XML or other document type repositories. Documents to be released may be obtained through queries on different data sources in the Data Store as well as through access to internet/intranet services. Databases in the Data Store as well as other sources can be assumed to be protected by usual access control systems.

The main modules in the Release Control System are the *Release Constraints Store* and the *Matching Module.* The first module is the repository of release constraints and includes constraints explicitly inserted by the security officer, constraints derived from the data store with processes guided by the security officer, and constraints derived from examples of restricted and released documents by a learning process, which will be explained later in this section. An example of derivation of constraints from the data store is using the information about access control rules in order to derive sensible associations of terms directly or indirectly bounded to items whose access is forbidden by those rules. Ontologies and thesauri can also be used to derive new release constraints by identifying "semantic" similarities to the given ones. Optimization modules (not depicted in the figure) will also operate on the Release Constraints Store. For example, the set of Release Constraints can be reduced considering the subsumption relationships along the hierarchies of matching rules and controlled items. As an example if RC_1 says that each tuple in the outgoing data should not contain the strings *Woo* and *75k,* and RC_2 says that the string *Woo* should not appear anywhere in the outgoing data, RC_1 can be disregarded.

Given a set of release constraints in the Release Constraints Store, the Matching Module is responsible for checking each one of them against the outgoing documents, and for blocking the release of those for which any of the constraints is satisfied. Since we assume documents in XML form, the module must contain a matching algorithm with a XML parsing component. A basic version of the algorithm may consider an explicit listing of controlled items in each release constraint, and hence, it will perform keyword-based matching. Clearly, appropriate indexing on the keywords appearing in the release constraints will be necessary, so that all applicable release constraints are efficiently selected upon reading one of the sensitive keyword in the outgoing data. Efficient techniques should also be devised in order to keep track of the current position within the list of controlled items in each release constraint. More sophisticated versions of the algorithm will consider working with compact representations of the controlled items (possibly in the form of queries).

3.2 Integration of a Learning Module

While the security officer in some cases may be able to explicitly provide both controlled items and associated matching rules, we believe there are a large number of documents to be restricted for which only more vague criteria are available. For this reason, our framework proposes the integration in the above architecture of a *Learning Module* that has the main goal of learning release constraints. In particular, in this paper we will show how the learning component can generate specific matching rules starting from controlled items, some domain knowledge, and a training set containing documents already marked as restricted or released.

In principle, given a sufficiently large training set of positive and negative examples we may ask a learning algorithm to derive controlled items and matching rules accordingly to the syntax described above. In practice, this is not a realistic requirement: the learning process, and, in particular, the extraction of features from the examples must be guided by some knowledge about the specific domain. One possibility is to start with a possible set of "critical" correlated keywords from the security officer and with a set of parameterized matching rules. For example, the security officer may consider the distance of two keywords in a document to be a relevant criterion, while the number of occurrences of keywords not to be a relevant one. In this case, the upper bound on the "distance" becomes a parameter to be tuned by the learning process.

The main issue is how to choose appropriate parameterized rules so that the algorithm may minimize the rate of mistakenly released and mistakenly restricted documents by tuning the parameters. In order to illustrate the general idea in the specific context of our matching rules, we give an example, by considering only cardinality rules. As we have observed in Section 2.2, the default and most conservative cardinality rule $NumVal \geq k$ is obtained by using $k=1$. The value of k may actually be used as a parameter in the learning process. For example, from the training set it may be observed that all correctly restricted documents contain at least 2 terms of the controlled items, while many mistakenly restricted ones contain only one. The value of *k* may then be raised to 2. Of course, there are several hypotheses on the document corpus and on the learning algorithm (including the size of the training set) that should hold to preserve a correct behavior of the system while reducing its mistakes.

In the context of security, it may be desirable to implement a learning process that preserves the following "conservativeness" property:

All documents that have been certified as restricted by the security officer will be restricted by the system.

Preserving this property implies that any derivation of new rules or refinement of existing rules must lead to a global set of release constraints that is still able to correctly classify documents that are known to be restricted.

Figure 2 depicts the Learning Module and its integration in the overall architecture, including a Monitoring Tool that will be discussed below..

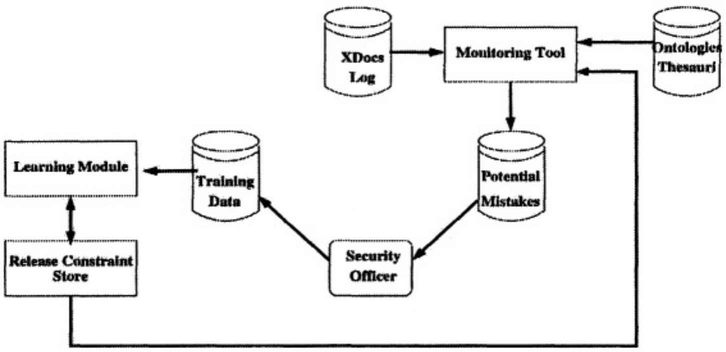


Figure 2. The Learning Module and the Monitoring Tool

3.3 The Learning Module

The learning module has two main functionalities:

- *It derives release constraints for the initial set-up of the system.* As mentioned above, performing this task requires a training set of documents marked to be restricted or released, approved by the security officer; it also requires some domain knowledge, possibly in the form of controlled items and/or parametric rules. We impose that the rules obtained by the learning algorithm will preserve the conservativeness property, i.e., the system using these rules would correctly restrict at least all documents marked to be restricted in the training set. The algorithms and strategies involved in this task under specific assumptions are described in Section 4.

- *It helps refining the system behavior upon the identification during system operation of mistakenly released and mistakenly restricted documents.* This task is based on the assumption that the security officer monitors the system behavior and provides feedback to the learning module by dividing samples of the processed documents in four categories: correctly restricted (CRes), correctly released (CRel), mistakenly restricted (MRes), and mistakenly released (MRel). It can be considered a form of online learning since the process may automatically start once a large enough set of these samples becomes available.[2] There are essentially two approaches to perform this task: (i) Re-applying the learning process used in the initial set-up considering as the training set the new samples as well as all documents whose classification has been verified in the past by the security officer. When the set becomes too large, many strategies are possible, including, for example, the use of a sliding time window for the past, or of a "least recently used" strategy. The rules obtained through this process replace the previous ones. (ii) Refining the current rules using only the new set of CRes, CRel, MRes, and MRel documents. This is potentially a more efficient approach, but details on how it can be done are very dependent on the learning algorithm.

3.4 The Monitoring Tool

A critical system will have a huge number of documents flowing through it, and specific strategies must be devised to monitor its behavior in order to provide feedback to the learning module. The most trivial strategy consists of periodically extracting samples and forwarding them to the security officer, but it is likely to be unsatisfactory, since any significant frequency of sampling involves an unsustainable amount of work for the security officer.

In our architecture we propose to introduce a monitoring tool that filters the documents based on a "similarity" metric so to drastically reduce the number of documents to be examined by the security officer. Note that in principle the tool should be applied both to restricted documents to identify potentially mistakenly restricted ones, and to released documents to identify potentially mistakenly released ones. However, while mistakenly restricted documents may have other ways to be recognized (e.g. feedback from users whose requests have been refused) and are less critical, the problem is

[2] Actually, considering the conservativeness property, even a single example of mistakenly released document can be useful to refine the rules.

serious for released ones. Also, each restricted document is associated with the release constraint (controlled items and matching rules) that has prevented its release. When the security officer examines the document this association can help recognizing the reason for the sensitivity and, in case of a mistaken restriction may lead to drop or explicitly modify a rule. Released documents on the other side have no attached information. Our focus is on tools to detect potentially mistakenly released documents.

The monitoring tool considers "similarity" of released documents to restricted ones based on the closeness of the document to the classification threshold. This technique is still based on learning and details really depend on the specific learning algorithm, but, intuitively, it works as follows: essential features of documents to be restricted and to be released can be represented by numerical values, and an n-dimensional boundary that separates the two types of documents can be found. Then, a document being examined has the same features computed and the closeness to this boundary can be evaluated. Intuitively, documents that are close to the boundary are similar both to restricted and released ones. The monitoring tool should rank the documents according to their closeness to the boundaries, so that the security officer can dynamically and asynchronously examine them.

The monitoring tool also uses other techniques to identify potential MRel documents:

- The use of ontologies and thesauri to substitute words or structures with those appear in controlled items.
- The explicit relaxation of some of the rules. For example, increase distance (for distance based condition), decrease cardinality, allow regular expression in the string literals, dropping rules by making them always satisfied (e.g., indefinite order, infinite distance).

Intuitively, the application of these techniques, as well as of the learning based one, should be guided by the goal of identifying a very small fraction of the released documents. This is both required due to the limited resources and, more importantly, by the implicit assumption that security officer policies are quite conservative: it is more likely that few restricted documents have been incorrectly released.

These components of the architecture operate asynchronously with respect to the main Release Constraint System.

4. LEARNING FOR KEYWORD-BASED RELEASE CONTROL CONSTRAINTS

As we mentioned in our discussion of the release control system architecture (Section 3), learning for the purpose of obtaining specific release control constraints plays an essential role. We also mentioned that domain experts or security officers should guide the learning by giving relevant features that the learning process should focus on. In this section, we study, in more detail, such a *feature-based learning* when keywords-based release control constraints are considered.

In general, a feature-based learning requires that domain experts provide certain domain knowledge to the task of learning. The domain knowledge specifies what types of features are important for the task at hand. A learning mechanism is to identify the specific features, within the given types, that can be used to distinguish between different sets of documents.

We believe this approach is rather useful in practice. Indeed, in information retrieval techniques, features of texts are routinely used in deciding the relevance of documents. For example, text appearing in subject line, or title, or abstract, may be more important than that appearing in the body of a document. However, the features used in information retrieval are usually "hard coded" into the information retrieval systems. This may be reasonable for documents that do not have much of structure. When a document is represented in XML, features of various types need to be considered.

Often, specific domains of applications determine the features that are important in XML documents. For example, in applications where documents are structured as a hierarchical tree (e.g., parts contain chapters, chapters contain sections, and section contain paragraphs), it is important to talk about contents belonging to a particular level of the hierarchy (e.g., two sections of the same chapter). Hence, it is important for the domain experts to specify certain 'generic' types of features that are important for the domain.

For release control, the above discussion about domain-specific feature implies the following strategy for generating release control rules. Step 1, the domain experts specify certain type of features that are important. Step 2, the learning system discovers the conditions on the features for the release control purpose. In the following, we illustrate this approach on keyword-based release control.

4.1 Keyword-based features

In this approach, we assume the controlled items are simply keywords, and a feature specifies the particular relationship based on particular ways of appearance of these keywords in a document. The "particular ways" of appearance is a type of feature given by a domain expert. We codify such features by using the notion of a *feature function.*

Formally, we define a *feature function* as a function *f* such that, given an XML document *Doc* and *m* nodes of *Doc,* it returns a tuple of *n* values, i.e., $f(Doc, N_1, \ldots, N_m) = (a_1, \ldots, a_n)$.

Note that the functions we define are on the nodes of XML documents. We still call them keyword-based since we will use the appearance of keywords to determine which nodes in the XML document to consider, as we will show later.

Example 4. The following are three example features:

- Distance feature: $dist(Doc, N_1, N_2) = D$, where *D* is an integer parameter, and $dist(Doc, N_1, N_2)$ is defined as $|P(N_1)| + |P(N_2)| - 2 * |P(N_1) \cap P(N_2)|$ and $P(N_i)$ means the path from the root of the XML document *Doc* to node N_i. This feature extracts the distance between the two nodes when the document tree is viewed as a graph.
- Least common ancestor feature: $lca(Doc, N_1, N_2)=T$, where *T* is the tag of the node defined by $Suffix_1(P(N_1) \cap P(N_2))$. Here *T* is a string and $P(N_i)$ is the path from the root of *Doc* to node N_i. This feature extracts the tag value of the lowest (in terms of the node level) common ancestor of the two given nodes.
- Ordering: $ocd(Doc, N_1, N_2)=R$, where $Order(N_1)\ R\ Order(N_2)$. Here, *R* is one of the relational comparison symbols. This feature gives the order relationship between two nodes.

In the above examples, each feature function only returns one value (more specifically, a tuple with one value in it).

The above feature functions can naturally be used for specifying conditions on the appearance of keywords in XML documents. For example, given two keywords K_1 and K_2, we want to talk about the appearance of them within certain distance in an XML document. To do this, we only need to specify a condition involving the distance of the nodes that "contain" K_1 and K_2, respectively.

More formally, we define the notion of *occurrences of keywords* in an XML document as follows: Given a set of keywords $\mathcal{K} = \{K_1, \ldots, K_n\}$ and a document *Doc,* an *m-occurrence of* $\mathcal{K}$ *in* Doc, where *m* is an integer no greater than the number of elements in $\mathcal{K}$, is a partial mapping *occ* from $\mathcal{K}$ to the nodes of *Doc* such that $occ(K_i)$ contains K_i for each $i=1,\ldots, n$ if $occ(K_i)$ is defined, and the number of K_i with $occ(K_i)$ defined is exactly *m.* Here, "contains" means either an attribute of $occ(K_i)$ has the value K_i or the value of $occ(K_i)$ is exactly K_i. To simplify the presentation, in the following, we use "contains" to mean that the value of the node is exactly the keyword.

Example 5. In the XML document in Figure 3, there are seven 2-occurrences of *{"A", "B", "K_3"}*. Note that in the tree representation of the XML document in Figure 3, we only show the node values in the XML tree. Other information of the XML documents is omitted. The labels within the nodes (within the circle) are meant to identify the nodes for presentation purpose, and are not part of the XML documents.

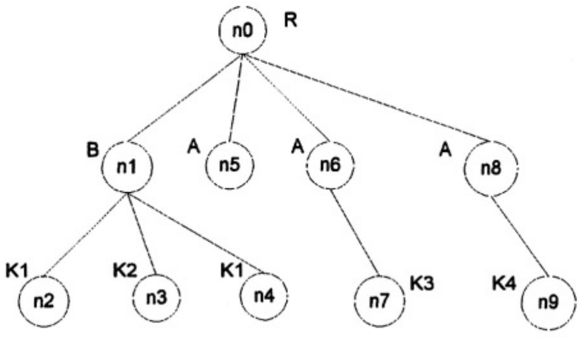


Figure 3. An XML document.

A partial occurrence *occ* of $\mathcal{K}$ in *Doc* is said to be a *maximum* one if *occ* is defined on the maximum number of keywords in $\mathcal{K}$ among all such partial occurrences.

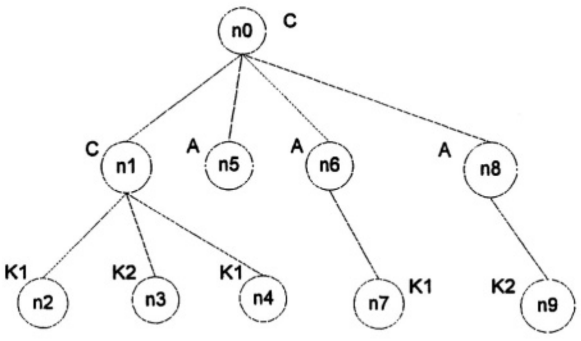


Figure 4. Another XML document.

Now we are ready to extract keyword-based features from XML documents. Assume the feature function takes the form $f(Doc, N_1, \ldots, N_m)=(a_1, \ldots, a_n)$. Then the features extracted from the document via function f is the set of n-tuples given as follows: $(a_1, \ldots, a_n)$ is one such n-tuple if and only if there exists an m-occurrence *occ* of $\mathcal{K}$ in *Doc* such that $f(Doc, occ(K_1), \ldots, occ(K_m))=(a_1, \ldots, a_n)$.

Example 6. Given the distance feature function and the set of keywords *{"K_1", "K_2"}*, the set of features extracted from the XML document in Figure 3 is *{(2)}*. The distance feature of the same keywords extracted from the XML document in Figure 4 is *{(2), (4)}*.

Together with a set of keywords (used as controlled items), a Boolean condition based on the features extracted via feature functions can be used as a release control constraint. More specifically, given a document, we extract the features using the feature functions, and then test the Boolean condition on the features. If anyone of the specific features satisfies the condition, we should block the release of the document.

Example 7. Suppose a release control constraint specifies the distance between keywords "K_1" and "K_2" to be less than *3*. Then, none of the documents in Figures 3&4 can be released since each one contains at least one occurrence with distance 2. Suppose another release control constraint specifies the lowest common ancestor of keywords "K_1" and "K_2" to have value *"B"*. Then the document in Figure 3 cannot be released while that in Figure 4 can be.

4.2 Learning for keyword-based release control constraints

A security officer can certainly set up a set of release control constraints by giving a set of keywords and a condition on the keyword-based features (extracted from a set of feature functions). In a practical system, however, it is more likely that system need to learn from a set of examples to establish the specific release control constraints. In this subsection, we will show how this may be done for keyword-based release control constraints.

As mentioned earlier, we assume that (1) a set of feature extraction functions are set up by domain experts as the likely types of features to be concerned by the release control constraints; (2) the security officer gives a collection of keyword sets to be controlled, and (3) a set of documents is provided as learning samples (i.e., the documents either cleared for release or restricted by the security officer). Given (1), (2) and (3) above, the task of learning is to give a set of release control constraints by specifying conditions on the feature values extracted by the feature functions. We now outline a method to do this. The key issue is to convert this learning problem to one where traditional learning algorithms can apply.

Assume $\mathcal{K}$ is the set of keywords we are concerned with. For each feature function $f(Doc, N_1, ..., N_m)=(a_1, ..., a_n)$, we create the following attributes (as in a relational database schema): For each *i*, *i=1, ..., n,* and each subset *{K_1,*

..., $K_m\}$ (of size m) of $\mathcal{K}$, we get an attribute name $f[a_i, K_1, ..., K_m]$. We use a relational schema of all such attributes. Furthermore, we add a document ID as an extra attribute to the relational schema.

Example 8. Consider the keyword set $\{K_1, K_2, K_3\}$ and the three features given in Example 4. Then we have the following 18 attributes (in addition to the document ID attribute):

$dist[D, K_1, K_2]$	$lca[T, K_1, K_2]$	$ocd[R, K_1, K_2]$
$dist[D, K_2, K_1]$	$lca[T, K_2, K_1]$	$ocd[R, K_2, K_1]$
$dist[D, K_1, K_3]$	$lca[T, K_1, K_3]$	$ocd[R, K_1, K_3]$
$dist[D, K_3, K_1]$	$lca[T, K_3, K_1]$	$ocd[R, K_3, K_1]$
$dist[D, K_2, K_3]$	$lca[T, K_2, K_3]$	$ocd[R, K_2, K_3]$
$dist[D, K_3, K_2]$	$lca[T, K_3, K_2]$	$ocd[R, K_3, K_2]$

Of course, certain relationships between the parameters may lead to a reduction of the number of attributes. For example, since the value for $dist[D, K_1, K_2]$ is the same as $dist[D, K_2, K_1]$, we can omit one if we already have the other. Another example is that if we know $ocd[R, K_1, K_2]$, then we implicitly know $ocd[R, K_2, K_1]$, and we can omit one of these two attributes. From this reasoning, we can reduce the number of attributes in the example to nine (in addition to the document ID attribute).

By using the training samples, we set up feature tuples in the above relational schema as follows. Given an XML document *Doc* and a maximum occurrence *occ* of $\mathcal{K}$ in *Doc,* we generate a tuple as follows: For attribute $a_j[K_1, ..., K_m]$, it gets a **null** value if the number of keywords *that occ* is defined on is less than m; otherwise, the attribute gets the corresponding value from $f(occ(K_1), ..., occ(K_m))$.

Example 9. Consider keyword set $\mathcal{K}=\{K_1, K_2, K_3\}$ and the XML document in Figure 3. Consider the occurrence $\mathcal{K}$ in the document such that K_1 is mapped to n_2, K_2 to n_3 and K_3 to n_7. Then the value for $dist[D, K_1, K_2]$ is 2, while the value for $ocd[R, K_2, K_3]$ is "<".

In the above method, each occurrence of a set of keywords in an XML document sets up a feature tuple in the given relational schema.

Given a set of documents and a set of keywords, we can set up a set of feature tuples for each occurrence of keywords in each document. When we have a set of documents that need to be restricted (determined by the security officer), then the tuples obtained by that document form a set of "restricted" feature tuples. When we have a set of documents that can be released, the tuples obtained by that document form a set of "releasing" feature tuples. Note, however, that there is an apparent difference in the semantics of the feature tuples in the above two sets. In the “restricted” set of feature tuples, a document needs to be restricted even if only one tuple (among all those that belong to the same document) is “dangerous”.

Example 10. Suppose we want to restrict a document from being released when the keywords **“K_1”** and ***“K_2”*** appear within distance *3.* In this case, both XML documents in Figures 3&4 should be restricted. However, not all the distance features of *“K_1”* and *“K_2”* in Figure 4, namely *2* and *4,* satisfy the restricting constraint.

By reducing the learning task to two sets of feature tuples as given above, we can now apply any traditional learning algorithm [Mit97, Qui96]. In general, the learning algorithm will produce a *classification* condition on the parameter values. That is, given a feature tuple in the given relational schema, the classification condition gives either **restrict** or **release**. For each document, if any of its feature tuple results in the value **restrict**, then the document should be restricted from release. This classification condition will be used as the matching rule in a release control constraint and the corresponding keywords will be the controlled items.

5. RELATED WORK

The concept of information release control has been explicitly introduced recently in [Wie00, Mon01, RW01], but a general formal framework does not currently exist.

Some form of control over outgoing data has been performed since a long time in different contexts, but it has been mostly based on basic filtering tools, and heuristics have been directly coded into programs. An attempt to specify rules in a high level language is represented by *Felt* [Swa94] which, among its features, provides language statements to identify words, or parts of words in documents and to drop or substitute these words. Restricting rules are then compiled into a program for automatic matching. Despite we are not aware of any structured formal framework for release control as the one we are proposing, we should mention the very active research field of information filtering which also includes publication/subscription systems.

An information filtering system is an information system designed for unstructured or semistructured data [BC92], as opposed to typical database applications that work with highly structured data. With respect to the general information retrieval paradigm, in which a large body of data has to be searched against a specific user search criteria, in information filtering it is usually the case that there are a large number of specifications about information needs of a large number of people and/or tasks, and they all have to be matched against the same text data, in most cases dynamically produced and distributed by some data sources. Publication/subscription systems (see, e.g., [FJL+01] and [ASS+99]) are an instance of information filtering applications. For example, consider the task of sending to each user subscribing to a news service the subset of the daily news specified in his/her profile. The analogy with our work is quite clear: the set of release constraints can be considered a set of subscriptions, and any matching against the outgoing data leads to a specific action, which usually is preventing the release of the data. Despite this analogy, our goal is to deal with a more heterogeneous set of data that includes also structured data resulting from database queries.

Some work has been done specifically on XML documents filtering for publication/subscription systems [AF00, DFFT02, PFJ+01]. We are considering the algorithms and techniques proposed in this area for their adaptation to implement the matching module of our architecture. However, it is still not clear if the algorithms can be adapted to our language for matching rules and if they are compatible with the specific requirements that a security application imposes.

Alternative approaches for the matching module are represented by continuous query techniques [CCC+02, CDTW00, MSHR02].

Our work is also related to what is commonly known as Internet filtering software. Filtering or blocking software restricts access to Internet content through a variety of means. It may scan a Web site's content based on keywords, phrases or strings of text. It may also restrict access based on the source of the information or through a subjective ratings system assigned by an anonymous third party. Mostly, this software has been focused on blocking pornographic content, and it has not been considered very successful until now, either for under-blocking or over-blocking Internet content. This is partly due to the way blocking criteria have been devised and partly from the inherent complexity of the task. Despite some aspects are very related, we are considering several issues about the treatment of structured and semi-structured data while the data considered by these systems is usually unstructured, or the structure it has it is totally unknown. Regarding our learning-based approach, the general techniques to learn the release constraints from a training set of positive and negative examples are well known [Qui96, Mit97]. Learning has been extensively applied in text categorization and text filtering [Seb02], but efforts to study and apply learning techniques for the categorization and filtering of XML documents have just recently started and pose many open questions. We plan to investigate further the problem of learning release constraints and of refining the constraints considering, in particular, recent statistical learning approaches [CST00].

Regarding our intention of integrating keyword-based techniques with text categorization algorithms, we will take into account the results of the Genoa Technology Integration Experiment [Mon01] performed as part of a DARPA project on boundary control. In the experiment keyword-based and NLP (Natural Language Processing) techniques were compared in their accuracy on a concrete experimental scenario; the experiment involved a corpus of heterogeneous documents that had to be controlled for the release of potentially sensitive information for terrorist attacks. (The scenario referred in particular to the Aum Shinrikyo Japanese cult that bombed the Japanese metro system with an anthrax pathogen.)

Finally, a large amount of work has been done in the past decade on word sense disambiguation (see, e.g., [Les86]), and ontology-based reasoning (see, e.g., [GL02]) which are important issues for any content-based document management application and, in particular, for a more accurate release control. An example of use of word sense disambiguation and natural language processing in boundary control, limited to text documents, is the Genoa Technology Integration Experiment [Mon01] performed as part of a DARPA project on boundary control. In our case ontology-based reasoning is used to add relevant rules and/or feature functions over XML documents.

6. CONCLUSION

In this paper, we presented an architecture for controlled information release. We emphasized on the automated learning (for release control constraints) in the whole infrastructure for release control. We formalized the release control constraints and presented a learning strategy for keyword-based release control constraints.

The above release control system can be useful in a traditional data system, such as database system, FTP directories, and web sites. More recent applications, such as web services, can also benefit from release control. Web services [New02] are an emerging paradigm for internet computing heavily based on XML and on the SOAP protocol. Web services present to the network a standard way of interfacing with back-end software systems, such as DBMS, .NET, J2EE, CORBA objects, adapters to ERP packages, and others. While standards are currently under definition for authentication and authorization, as well as for encryption, controlling the information that is released through a web service to the general internet or to a restricted subset of cooperating processes will be one of the major issues that will also probably affect the success of the new paradigm. While the major objectives of the proposed project are not to develop a specific technology for web services, we envision a very interesting integration of the technologies that may emerge from our results into a web service architecture.

Content-based firewall is another interesting application of our release control system. Current firewall systems are mostly based on selecting incoming and outgoing packets based on source/destination IP and port numbers. Filtering software based on dirty-word checking or virus identification in some cases have been integrated. The content analysis is however quite primitive both in the definition of the filtering criteria and in the matching algorithms. We advocate an approach that incorporates the release control into firewall systems to allow more advanced monitoring on the contents that are released through the firewall.

ACKNOWLEDGEMENTS

This work was supported by the NSF under grant IIS-0242237. The work of Bettini was also partly supported by Italian MIUR (FIRB "Web-Minds" project). The work of Wang was also partly supported by NSF Career Award 9875114. The authors would like to thank Nicolò Cesa-Bianchi of the University of Milan for insightful discussions on computational learning techniques.

REFERENCES

[AF00] Mehmet Altinel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of 26th International Conference on Very Large Data Bases,* pages 53—64, USA, 2000.

[ASS+99] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC),* pages 53—62, May 1999.

[BC92] N. J. Belkin and W. B. Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM,* 35(12):29—38, December 1992.

[CCC+02] Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams – A new class of data management applications. In *Proceedings of the 28th International Conference on Very Large DataBases (VLDB),* pages 215—226, 2002.

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management ofData: May 16—18, 2000, Dallas, Texas,* pages 379—390, 2000.

[CST00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Machines (and other kernel-based learning methods),* Cambridge University Press, UK, 2000.

[DFFT02] Yanlei Diao, Peter Fischer, Michael Franklin, and Raymond To. Yfilter: Efficient and scalable filtering of xml documents. In *Proceedings of the International Conference on Data Engineering (ICDE),* pages 341—342, 2002.

[FJL+01] Francoise Fabret, H. Arno Jacobsen, Francois Llirbat, Joao Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast Publish/Subscribe systems. In *Proceedings of ACM International Conference on Management ofData (SIGMOD),* pages 115—126, 2001.

[GL02] Michael Gruninger and Jintae Lee. Ontology: applications and design. *Communications of the ACM,* 45(2):39—41, February 2002.

[JSSS01] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems,* 26(2):214—260, June 2001.

[Les86] Michael E. Lesk. Automated sense disambiguation using machine-readable dictionaries: How to tell a pinecone from an ice cream cone. In *Proceedings of the SIGDOC Conference,* 1986.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[Mon01] Eric Monteith. Genoa TIE, advanced boundary controller experiment. In *17th Annual Computer Security Applications Conference.* ACM, 2001.

[MSHR02] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD),* pages 49—60, 2002.

[New02] Eric Newcomer. *Understanding Web Services.* Addison Wesley, 2002.

[PFJ+01] Joao Pereira, Francoise Fabret, H. Arno Jacobsen, Francois Llirbat, and Dennis Shasha. Webfilter: A high-throughput XML-based publish and subscribe system. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB),* pages 723—725, September 2001.

[Qui96] J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys,* 28(1):71—72, March 1996.

[RW01] Arnon Rosenthal and Gio Wiederhold. Document release versus data access controls: Two sides of a coin? In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM),* pages 544—546, November 5—10, 2001.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys,* 34(1):1—47, 2002.

[Swa94] V. Swarup. Automatic generation of high assurance security guard filters. In *Proc. 17th NIST-NCSC National Computer Security Conference,* pages 123—141, 1994.

[Wie00] Gio Wiederhold. Protecting information when access is granted for collaboration. In *Proc. of Data and Application Security, Development and Directions, IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security,* pages 1—14, 2000,

Information Security Governance using ISO 17799 and COBIT

Elmari Pretorius

KPMG, Rand Afrikaans University – Johannesburg, South Africa

Prof. Basie von Solms

Rand Afrikaans University – Johannesburg, South Africa

Abstract: This paper discusses a project in which a mapping between ISO 17799 and COBIT's section DS 5 is being created. The purpose of this mapping is to synchronize these two documents to a certain extent, to make it easier to use both in an integrated way for information security governance and management.

Key words: ISO 17799, COBIT, Information Security Management, Information Security Governance

1. INTRODUCTION

In the last few years, Information Security had been moving very strongly towards the use of documents and guidelines based on so called Best Practices, Open Standards and Codes of Practice for governing and managing information security.

This can be seen as a very positive move, as it shows that a certain amount of maturity is being established in these areas.

Two documents that are playing an increasing role in this aspect are the ISO/IEC 17799 Code of Practice for Information Security Management, and Control Objectives for Information and related Technology, better known as COBIT. Section DS 5 of COBIT relates specifically to information security.

In many cases ISO 17799 is being used within IT departments as a guide for information security management, while COBIT DS 5 is used more by auditors for information security auditing. Furthermore, COBIT is raising its profile as an IT governance tool, and not only as an auditing tool.

In its role as an IT governance tool, COBIT can therefore be seen as being used on a strategic level, indicating the 'what' as far as governance is concerned. On the other hand, ISO 17799 can be seen more as being used on a lower level, specifying the 'how' as far as information security management is concerned.

Because of the wider use of these 2 documents (ISO 17799 and COBIT DS5), in many cases a need for synchronization between these two documents arise, in the sense that a mapping between the detailed control objectives of DS5 and the counter measures introduced by 17799, are beneficial.

This makes it easier in using COBIT DS5 for Information Security Governance and auditing, and implementing lower level counter measures based on ISO 17799.

Such a mapping will indicate which DS 5 detailed control objectives are mapped to which ISO 17799 controls, and vice versa.

Another angle that can prove to be beneficial for Information Security management and governance is a mapping of a company's Information Security policy to either ISO 17799 or COBIT DS5, or both. Such a mapping will indicate "compliance" areas within the policy document, and also highlight the "non-compliance" areas, and assist with the alignment of a company's IS policy to the ISO and COBIT frameworks.

This project will result in a database containing the mapping of ISO 17799 to COBIT, COBIT to ISO 17799, and a test case of a company Information Security mapping to the ISO and COBIT documents. A user interface will allow access from either the ISO 17799 side, or from the COBIT DS 5 side.

The purpose of this paper is to discuss these mappings, and to give an example of some of the results.

2. WHAT IS ISO 17799

ISO 17799 can be regarded as a comprehensive catalogue of "the best things to do in Information Security". Because it is a code, it is made up of best practice recommendations, which can be applied to fit each organisation's specific requirements.

Since ISO 17799 sets out the requirements for an Information Security Management System (ISMS), it helps to identify, reduce and manage the wide spectrum of threats to which information is commonly subjected.

ISO 17799 is organised into ten major domains, each covering a different topic or area. These domains are:

- Security policy;
- Organising information security;
- Asset classification;
- Personnel security;
- Physical and environmental security;
- Communications and operations management;
- Access control;
- Systems development and maintenance;
- Business Continuity Management; and
- Compliance.

Each of these domains is divided into security control areas that can be applied to improve an organisation's information security. Each control area then has its own objective, and one or more controls to assist in achieving this objective.

ISO 17799's biggest exposure is to the IS manager. IT departments often use it as a foundation to build its Information security environment on.

3. WHAT IS COBIT?

The keystone of the COBIT framework is that control in IT is approached by looking at information as being the result of the combined application of IT related resources that need to be managed by IT processes, and by looking at information that is needed to support the business objectives or requirements.

COBIT presents IT activities in a manageable and logical structure and documents good practice across this structure. COBIT's good practices will help optimise information investments and provide a benchmark to be measured against when things go wrong.

COBIT's structure groups IT process into four broad groups:

1 Planning and organisation;

2 Acquisition and Implementation;

3 Delivery and Support; and

4 Monitoring.

It then defines high-level Business control objectives for these processes, and supports these with Detailed control objectives.

COBIT's biggest exposure is to the IS auditor (IA). Management and IA's often use it as a foundation to build IT audits on.

4. INFORMATION SECURITY POLICIES

The information security policy document is considered to be a control that is common best practice in any information security environment. This policy as a control, applies to most organisations and in most environments, and can provide a good starting point for implementing effective and efficient information security management.

An information security policy provides management with direction and support for information security management in accordance with the organisation's business objectives and requirements, and the relevant regulatory and legislative information.

To be effective an information security policy should be effectively communicated to all relevant personnel, and guidance on compliance to the policy should be provided.

It is crucial for an organisation to adopt a structured framework for policy definition, using a process that enables policies to be derived from the organisation's business requirements.

Failure to develop a meaningful information security policy, or developing a policy that does not cover all risks posed to the organisation in the information security environment, could expose the organisation to potentially catastrophic breaches. From this it is important to ensure that all risk areas within the information security environment, applicable to the specific organisation, is mitigated by an effective and comprehensive information security policy.

5. USING THE MAPPING

It is envisaged that the resultant database (mapping) can be used for different purposes.

5.1 Suppose the IT Department of Company X has based their information security policy and counter measures on ISO 17799. The Information Security Manager now wants to see which DS 5 detailed control objectives are covered by the ISO 17799 counter measures he/she had installed or implemented. The mapping will immediately provide an answer.

5.2 An internal auditor of Company X wants to audit the company's IT Department's information security using DS 5. The auditor can now, using the mapping, see which of the detailed control objectives specified in DS 5 are implemented through ISO 17799, used by the Department.

Both these uses basically allow IT people, using ISO 17799, and auditors, using DS 5, to 'speak the same language' and compare apples with apples.

Maybe it will even allow IT people and internal auditors to become friends!

6. MAPPING EXAMPLE: ACCESS CONTROL MAP

The following is an example of the proposed mapping. This example covers Access control as an objective, and was mapped from ISO to COBIT.

6.1 Objectives covered by both ISO and COBIT

6.1.1 Business requirements for access control (Section 10.1 of ISO 17799)

ISO objective: Control access to information. Access to information and business process should be controlled on the basis of business **and** security req.

ISO control: 10.1.1 ACCESS CONTROL POLICY: Business requirements for access control should be defined and documented.

Link to COBIT control objective: 5.2 Identification, Authentication and Access

Motivation for link between ISO and COBIT: Both ISO's and COBIT's controls and objectives for access control state that specific procedures need to be in place and that these procedures must be documented, in order to keep access mechanisms effective.

6.2 Objectives only covered by ISO

6.2.1 Network access control (Section 10.4 of ISO 17799)

ISO objective: Protection of networked services. Access to both internal and external networked services should be controlled

ISO control: 10.4.8 NETWORK ROUTING CONTROL: For shared networks, routing controls should be implemented to ensure that computer connections and information flows do not breach the access control policy of the business applications.

Link to COBIT control objective: No apparent link to COBIT

Motivation for link between ISO and COBIT: COBIT deals with access control in a broad manner, and although it seems that it's access control objectives cover all areas of access, it does not have a control objective specific to network access control.

6.3 Objectives only covered by COBIT

6.3.1 Firewall architecture and connections with public networks

COBIT objective: Adequate firewalls should be operative to protect against denial of service attacks and any unauthorised access to internal resources

COBIT control: Firewalls

Link to ISO control objective: No apparent link to ISO

Motivation for above link between ISO and COBIT: ISO doesn't have a specific objective referring to firewalls and the use of it, while COBIT has a specific control objective dedicated only to firewalls.

7. CONCLUSION

Besides synchronising the two frameworks, the use of the mapping will also reduce confusion and deviations that exist between IT and Audit. The fact that the mapping can provide IT and Audit with the links at the touch of a button, means that a lot of time can be saved when auditing and setting up the Information Security environment. In addition the "IS policy alignment tool" can prove to be beneficial in the drafting and reviewing of a company's policy with regards to the ISO 17799 and COBIT DS5 frameworks.

REFERENCES

1. ISO / IEC 17799:"Code of Practice for Information Security Management"
2. COBIT - Audit Guidelines, 3rd edition

TRACING ATTACKS AND RESTORING INTEGRITY WITH LASCAR

Alexandre Aellig, Philippe Oechslin
LASEC EPFL

Abstract: We present a novel method to trace the propagation of intrusions or malicious code in networked systems. Our solution is aimed at large numbers of loosely managed workstations typical of a research environment as found in CERN. The system tags events which have a potential to become harmful. On a given machine all processes that results from the tagged event are marked with the same tag and the tag is carried on to others machines if a tagged process establishes a connection. Tag creation logs are stored in a central database. When an intrusion is detected at a later time, all machines and processes that may have lost their integrity due to this incident can easily be found. This leads to a quick and effective restoration of the system. Our implementation of the system is designed to incur very little overhead on the machines and integrates easily with many flavors of the Linux operating system on any type of hardware.

Key words: Recovery, Intrusion Propagation, Intrusion Detection

1. INTRODUCTION

Undesired intrusions are always a problem but intrusions in large groups of workstations are particularly serious. This is especially true if the workstations are managed by their users or many groups rather than by a single central entity. An intrusion showing up at a given workstation may be the result of the compromise of various intermediate machines. All implied machines may not show the same symptoms with some compromised machines not showing any symptoms at all. Cleaning up all machines that show symptoms is thus not enough. Malicious code may still be lingering deep in a machine and strike again when all other machines have been restored. Restoring all systems is not an option when there are too many machines or too many managers.

2. THE LARGE SCALE ATTACK RECORDER (LASCAR)

The large scale attack recorder provides a mean to find out exactly which machines have been hit by a given attack. LASCAR is divided into three parts, the recorder, the database and the analysis tool. The recorder consists in a module loaded on every hosts of the workstation cluster. This module tracks all the events in each workstation and logs the needed information to the central integrity database. Two types of events are logged by the recorder: process events and network events. The analysis tool parses all the events in the integrity database and reconstructs all propagation paths within and between the workstations. Once a potential integrity breach has been detected, the analysis tool is able to trace back the origin and the evolution of the breach in the cluster protected by LASCAR. In particular, it can determine which are the corrupted machines in the cluster and visualize the path and the propagation of the attack.

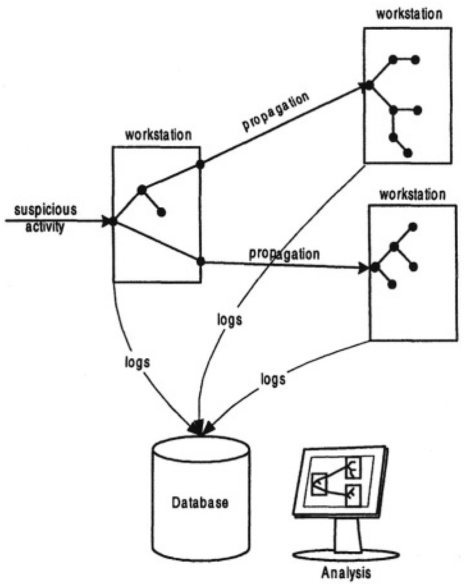


Figure 1. LASCAR at work

2.1 Existing Work

Our approach of recording the propagation of suspicious activities across processes and workstation is quite unique. Previous work that is related to our approach includes:

- The EMERALD system [2] is an advanced intrusion detection system aimed at large scale networks. It is composed of classical IDS systems distributed on all elements of the network and of an intelligent method to merge and correlate all information gathered. It is a heavy system in the sense that complex statistics have to be gathered in many places of the network and it suffers from typical shortcoming of IDS, namely that they generate false alarms and that they are not able to detect new classes of attacks that were not imagined by their conceivers. Our contribution is much more lightweight and humble, in the sense that we do not actually detect attacks but we gather evidence for later damage assessment and integrity restoration when some other system or person has detected an attack.
- For database security it has been proposed (e.g. in [1]) to tag data in order to indicate whether it is correct, damaged to some extend or even unsafe to use. The tags propagate when datasets are combined in calculations. The advantage of the method is similar to our case: In case of loss of integrity only the damaged data needs to be reconstructed rather than the whole database.
- In [3] the propagation of intrusions across multiple workstations is formally stuidied. The results of the paper could allow us to find out the intrinsic properties of an intrusion by matching our data to the mathematical models.

2.2 Tagging inside a workstation

Inside every machines of the LASCAR cluster, the recorder monitors network and processes activities. Using the Linux Kernel modules (LKM) ability to intercept system calls, it can log every incoming or outgoing connection on the machine and every new listening socket or session id (SID) change.

When the machine receives an incoming connection, LASCAR checks whether this connection is considered as trusted or not. In the latter case, the recorder logs the event and tag the process handling this connection with a "suspicious" flag. This flag indicates that the corresponding process is launched by an untrusted connection and thus can be potentially dangerous for the machine integrity. All child processes on the machine will automatically inherit this flag. In order to limit the volume of information logged, the recorder does not log every new process that inherits a "suspicious" flag but only processes that are created within a new session. A new session is typically created when a user launches a process in the background or when multiple interactive sessions are run in different windows. Lastly, the recorder logs all outgoing connections which are established by flagged processes.

A suspicious activity (and a potential loss of integrity) can thus be traced on the machine from the point where it entered the system up to all points where connections were made to other machines.

2.3 Tagging on the network

The previous method can be generalized to the entire LASCAR cluster. However in this case, the "suspicious" flag must now be remotely transmitted amongst machines of the LASCAR cluster.

The simplest way of propagating the flag is to set a bit in the IP header of the packets transmitted by a flagged process. This could for example be one of the Type Of Service (TOS) bits which are not usually used within a local network. (An alternative would be the "evil bit" suggested in [4] published on april fools day this year). The best way of using such a bit would be to use the default value of the bit for the flagged processes and to use a non-default value for all other processes. Thus, packets providing from machines not running the LASCAR recorder would automatically appear as suspicious. Since connections from outside the cluster are considered suspicious anyway and administrator privileges are required to set IP header bits, only machines within the cluster where the administrator account has been compromised could avoid logging by resetting the bit. A more secure approach would be to use a set of bits to carry the result of a cryptographic calculation. For example, the calculation of the IP packet ID field (16 bits) could include a secret key owned by the LASCAR recorder. Finally, to prevent manipulation of the flag in transit or creation of spoofed packets, the IPsec protocol could be used to authenticate all packets in a cryptographically secure way.

In summary, an incoming connection will be logged by the recorder only if it comes from a machine outside the LASCAR cluster or if its IP header indicate that the connection was generated by a process previously marked as "suspicious". All connections internal to the cluster and originating from clean processes won't be logged by LASCAR.

3. EXPERIMENTAL RESULTS

To illustrate the behavior of LASCAR we have chosen two scenarios that have been recorded by our implementation of LASCAR: a normal user activity and a computer worm.

3.1 Normal user activity

Figure 2 shows an output of LASCAR analyzer generated from the integrity database. It shows the connection of a user to the LASCAR cluster composed by lasecpc12, lasecpc15 and lasecpc16. The user launches an xterm window and from that window he connects to two other machines, maybe to launch two calculations. In the original session he starts the lynx browser to access a web site on the Internet (198.133.219.25)

Lasecpc12 receives the incoming ssh connection from an outside host at the IP address 128.178.73.68. It logs the connection attempt and its two subsequent sessions: xterm and lynx. We see that each session then connects to others machines.

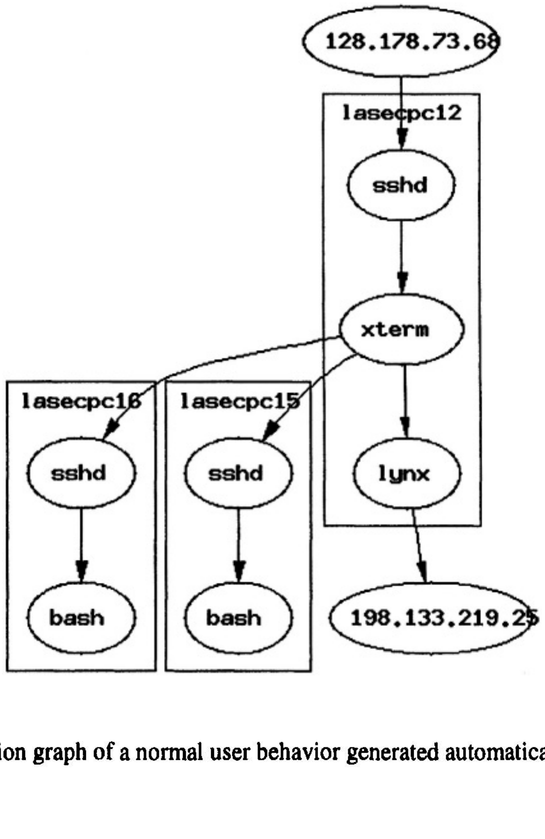


Figure 2. **Propagation graph of a normal user behavior generated automatically by LASCAR**

Lasecpc15 and lasecpc16 consider the incoming ssh connections as untrusted, even though they come from a machine inside the cluster. Indeed, connections were launched by sessions that were carrying the "suspicious" flag.

3.2 Computer worm behavior

The second example illustrates the evolution of a computer worm inside a cluster where machines are exposed to a common vulnerability of the secure shell program (SSH) on port 22. In this case, the worm enters through an external connection and corrupts a first machine of the cluster. It then tries to initiate each time two outgoing connections to replicate itself To simulate a work behavior we have manually connected a host by ssh and recursively opened two new sessions to random hosts from every ssh session.

With the help of LASCAR, we are able to trace back the evolution of the worm replication inside the cluster. In particular, we see that machines lasecpc15 and lasecpc16 are infected twice. When the attack has been detected, the LASCAR analyzer provides an exhaustive description of the attack path and scope with the id of the machines involved. This allows cluster administrators to quarantine only the corrupted machines of the cluster. In particular, the analysis of the propagation path may identify machines that have been infected but are not showing any symptoms of infection.

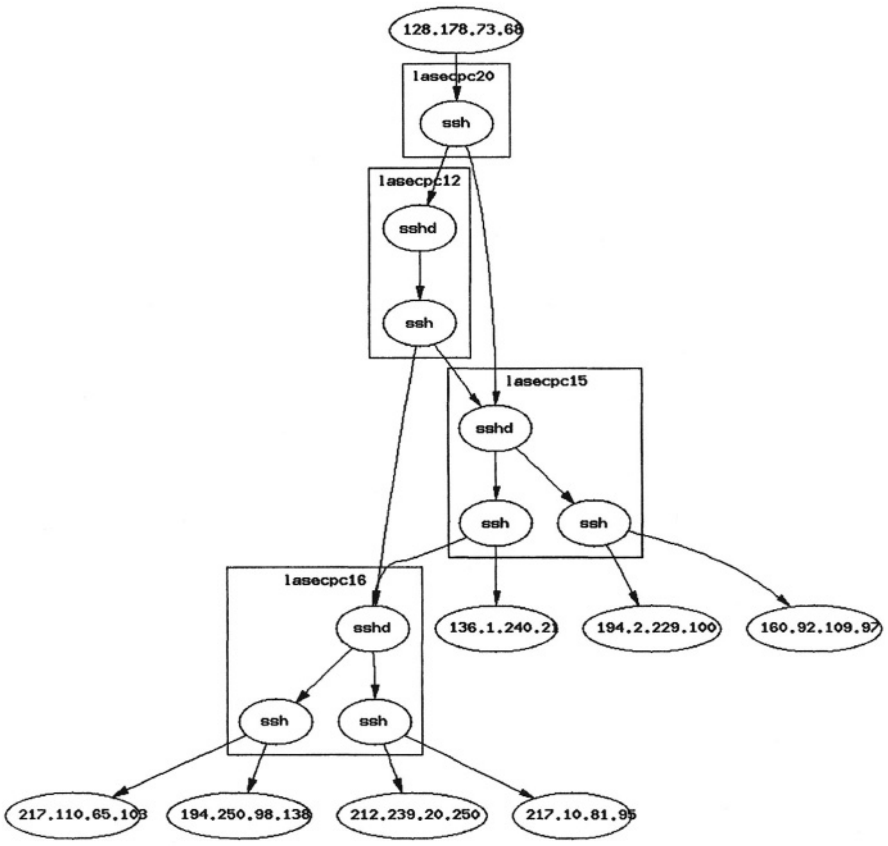


Figure 3. **Propagation graph of a simulated computer worm plotted by LASCAR**

4. IMPLEMENTATION DETAILS

LASCAR has been implemented on Linux using Linux Kernel Modules (LKM). These modules allow administrators to load and unload features on the fly inside the operating system kernel. LKM was the ideal solution for LASCAR since it combines the power and speed of being directly integrated into the kernel together with the flexibility of an independent program. Similar solutions are available on others UNIX systems.

The first implementation of the recorder was made specifically for the CERN network as a connection logger. After further research it has evolved to the actual LASCAR. Two functionalities were needed in order to implement the recorder: interception and modification of the connections and tagging of processes.

When a network event occurs or a new session is created, the corresponding system call is intercepted and LASCAR output is generated accordingly. The output can be customized, but it contains at least minimal security information such as connection IP addresses, ports numbers and processes info (session id, process id, user id) for the LASCAR analyzer. In the case of a process, we set a "suspicious" flag on the process called by the incoming connection. Every child process then automatically inherits this flag.

All the logs generated by the recorder are forwarded by the logging daemon to a remote integrity database, which centralize the logs of the LASCAR cluster needed by the analyzer to generate graphs of the LASCAR reports as shown in section 3. The analyzer currently uses timestamps to link the events between different machines (e.g. an outgoing connection and the matching incoming connection on another machine), since the concept of flagging trough TOS bits has not yet been implemented. Additionally, it has the ability, as a forensic tool, to regenerate graphs of a whole attack based on several criteria or to perform some basic intrusion detection using a blacklist.

5. DISCUSSION: LASCAR AND INTRUSION DETECTION

LASCAR is not an intrusion detection tool by itself. Its main function is to record traces of potential intrusions. Still, it can provide basic intrusion detection capabilities.

5.1 The suspicion criteria or identifying the potential troublemakers:

LASCAR only tags potentially harmful connections and processes. To do so it must have some criteria to identify potential harmfulness. In our case the criteria simply is the fact that a connection origins in a machine outside the cluster. This reflects the belief that the malicious activity that we want to trace enters the cluster from the internet, either because an attacker is targeting the cluster or because the attack propagates automatically and arbitrarily hits the cluster. If desired, the tagging criteria could be extended to include any process that acquires root privileges. More complex criteria could be made available by running a complete intrusion detection system on each workstation and using its output. The important point about our approach is that there is no need for such a complicated way of finding potentially harmful connections or processes. Indeed, we could even try to tag and log all connections and processes. The use of the suspicion criteria only serves to relieve the load on workstations and the main database.

5.2 The detection criteria or finding out when you have been hit:

Tagging and logging is only one part of LASCAR. The other part is the analysis of the gathered information in order to identify the machines that have lost integrity. To start an analysis we first need to know that we have been attacked. The most evident way of detecting an attack is when its first symptoms become visible (e.g. data loss, deterioration of service). Of course it would be more useful to detect an attack when it first appears. This is the goal of all intrusion detection systems. Alas, there is no way yet to build an IDS which will catch intrusions early and will not create a large number of false alarms. In a large setting like ours the amount of false alarms could be prohibitive. This is why we resort to logging suspicious activity such that when an attack is later confirmed we can go back and quickly get rid of its effects. Still, we have built a very pragmatic IDS capability into LASCAR. A criteria for attack detection used at CERN is a blacklist of IP addresses of servers hosting malicious code. Although not every attack will generate a connection to these addresses, any process that connects to the addresses of the black list must be malicious. Since connections made by potentially harmful processes are logged by LASCAR anyway, attempts to connect to addresses from the black list can be signaled with no overhead.

5.3 Using LASCAR as an IDS tool by itself:

The graphs that can be created from data gathered by LASCAR describe the propagation behavior of potentially harmful code or actions. This data contains metrics which are closely related to malicious activity and which could lead to an IDS system that would not have the high false alarm rates of other known systems. Interesting metrics would be the maximum depth in a propagation tree (how many times a suspicious activity has jumped to a next workstation), the maximum out degree of nodes in the trees (e.g. how many different workstations an activity has jumped to from a single workstation) or simply the number of nodes in a graph (how many machines at all have been hosts to the same activity).

6. CONCLUSIONS

Using a combination of process and connection tagging we have been able to create a system that can record the propagation of an activity through a cluster of machine and trace back all machines and processes that contributed to a loss of integrity. This information makes it possible to completely retrace the propagation of an attack and to eradicate compromised malicious code and compromised systems without having to restore large numbers of unconcerned hosts. Our system is particularly well suited for large networks of loosely managed systems as found in academic or research environments.

REFERENCES

1. P. Ammann, S. Jajodia, C. D. McCollum, and B. T. Blaustein, "Surviving information warfare attacks on databases," *Proc. IEEE Symp. on Research in Security and Privacy,* Oakland, Calif., May 1997, pages 31-42.
2. Peter G. Neumann and Phillip A. Porras. Experience with EMERALD to date. In *Proceedings of the 1st Workshop on Intrusion Detection and Network Monitoring.* 1999, USENIX, pages 73–80.
3. Sotiris Nikoletseas, Grigorios Prasinos, Paul G. Spirakis, and Christos D. Zaroliagis. Attack propagation in networks. In *ACM Symposium on Parallel Algorithms and Architectures,* pages 67–76, 2001.
4. S. Bellovin, The Security Flag in the IPv4 Header, *RFC 3514,* Internet Engineering Task Force, April 1st 2003

A SECURE MULTI-SITED VERSION CONTROL SYSTEM

Indrajit Ray and Junxing Zhang
Computer Science Department, Colorado State University

Abstract: The software development process is increasingly taking the form of a collaborative effort among several teams which are hosted at widely dispersed sites networked over the Internet. In this model of software development, multi-sited version control systems play a very important role to maintain the revision history of software and facilitate software evolution. In this paper we look into the security requirements of such multi-sited version control systems. We identify the security deficiencies in current systems and propose a new framework for secure multi-sited version control.

Key words: revision control, versions, software development, collaboration, integrity

1. INTRODUCTION

A version control system is used to maintain the revision history of software and facilitate software evolution. When software evolves it produces many revisions. A version control system stores these revisions, organizes them into meaningful structures that conform to software development principles, and provide operations to work with the different versions so that the integrity of the different versions of the software is independently ensured. A multi-sited version control (MVC) system is a version control system that operates at multiple sites to coordinate the software development effort of several teams that are collaborating with each other to develop a single piece of software. Even at the same geographical location such a system is useful to allow independent groups to work with the same development data, to enable interoperation in a heterogeneous environment, or to be a backup mechanism.

Version control systems were being used even when software development was primarily an individual effort. During that time the security of such systems was not of a major concern. This was because the data in the system was not shared. Later when version control systems became "team-ware" to share data among team members, security became somewhat more important. Still it was not considerably so; teams were located at the same site and the data sharing happened over a company's proprietary network. Since multi-sited version control system appeared, security concerns have become critical. This is because the data in the system now needs to be shared among multiple sites that are connected by open networks. Development teams may want to protect proprietary technologies from competitors. More important perhaps, is the need to ensure the integrity of the different revisions submitted over the Internet by different teams and the non-repudiation of their origin. This is particularly true about in the Open Source community. The various teams within this community generally do not care about the confidentiality of their software due to their commitment to the open source model; however they do need to ensure the integrity and non-repudiation of their code, and their developers do submit the contributed code via Internet.

In this work we propose a new secure multi-sited version control (MVC) system that can be easily implemented using COTS components. We utilize the Directory Information Tree (DIT) structure of X.500 directories [1, 2] to represent the MVC data model, network model and user account management model. Next we propose a novel authorization scheme for the MVC system that is based on integrating the access control list model [13] with the role-based access control model [12]. To support this new access model we propose to extend the Lightweight Directory Access Protocol (LDAP) [8, 9, 10, 15]. Finally we utilize the Simple Authentication and Security Layer (SASL) protocol [7] to solve the authentication problem and protect data confidentiality and integrity in the MVC system.

The rest of the paper is organized as follows. In section 2 we discuss some of the more well-known multi-sited version control systems to identify their security deficiencies. Section 3 develops the model for our multi-sited version control system. Section 4 describes our MVC framework. Finally we conclude in section 5.

2. RELATED WORK

There are a number of version control systems available today both commercial as well as open-source. The most widely used among them are ClearCase [14] (by the erstwhile Rational Software Corporation now owned by IBM), the UNIX Source Code Control System (SCCS) [16], the GNU Revision Control System (RCS) [17] and the Open Group's Concurrent Versions Systems (CVS) [11]. These differ from in each other mostly in terms of functionality and convenience. However, none of these systems were explicitly designed with security in mind. Consequently each of them have at least one of the following security deficiencies – (i) authentication deficiencies, (ii) authorization flaws, (iii) confidentiality and integrity problems and (iv) user account management problems – and cannot be used, without consequences, as multi-sited version control systems. In the following we discuss these deficiencies in more details.

Authentication Deficiencies - Most MVC systems use the native operating systems' authentication mechanisms for local access. This causes a dependency on the operating systems. Also remote access authentication varies among different systems. ClearCase doesn't have any support for remote access authentication, so do not SCCS and RCS. CVS supports several remote authentication mechanisms. Some of them are rather weak for authentication over open networks – such as connecting with *rsh* (which requires a machine at one site completely trust other machines at other sites) and simple password based authentication. Other remote authentication mechanisms that CVS supports, like authentication with GSSAPI [5] and Kerberos [6], are considerably stronger but are more suitable for authentication within a single administrative domain than over the Internet.

Authorization Flaws – We categorize authorization flaws into (a) missing dimensions, (b) encapsulation failures, (c) inconsistent controls, (d) write and checkout problems and (e) authorization coordination failure.

- Missing Dimensions – The most common authorization flaw is the authorization granularity is not as fine as the granularity of accessible objects. For example, CVS doesn't have access control at the version tree layer at all although it does support branches. ClearCase has controls at all three layers, but it doesn't control access at the version level.
- Encapsulation Failure – Many systems require users to check the permissions of the internal devices. SCCS, RCS, CVS users need to check the permission of data repository directories and even their subdirectories. ClearCase users need to check the permission of virtual workspace devices; in many cases these devices are not on the same machines where workspaces are used. Because these systems take internal attributes as the external attributes, users have to understand their internal mechanisms. This often confuses them and makes systems hard to use.
- Inconsistent Controls – Some systems are inconsistent in controlling different permissions. ClearCase doesn't control read and execute

access at branch level, but it does control the write access (checkout, checkin, and uncheckout) at the branch level.

- Write and Checkout Problems – Some systems (ClearCase, for example) do not treat the write permission in the same way as the checkout permission; instead they use the checkout permission to replace the write permission. The problem with this approach is that users cannot determine from the access control lists of files in the virtual workspace, who can change the content of the files or directories (elements). This is against the design purpose of using the virtual workspace to simulate the work environment in a file system.
- Authorization Coordination Failure – Some systems (CVS, for example) do not have an authorization coordination mechanism; so they use one central data repository to represent the whole repository family. This approach can't support local access model. Since every operation has to be done remotely, it incurs lots of resource overhead. ClearCase has the coordination scheme, but because it doesn't support remote access model it can only use the passive access mode of the scheme. More details will be discussed in the proposed framework.

Confidentiality and Integrity Problems – None of the four systems encrypt the messages exchanged in any manner by default for remote access. CVS has the provision for encrypting messages but does not do this without special configuration. This is a serious security problem; not only is the data in the messages endangered, attackers can also plant malicious code or data in the packets to damage the confidentiality and integrity of the importing data repositories.

User Account Management Concern – All current systems rely on the operating system to provide the network-wide database of user and group names. This causes a dependency on the operating system and undermines interoperability if the system needs to run in a heterogeneous environment.

3. MODELS OF THE MVC SYSTEM

The proposed MVC system is defined in terms of four different parameters (i) the data model, (ii) the network model, (iii) the access model and (iv) the user management model. We discuss each in details beginning with the data model.

The Data Model – The MVC system data model defines how the data is stored in the system. The data model defines four components – (i) data elements, (ii) data repository, (iii) data repository family and (iv) virtual workspace. We use the X.500 directories [1] to represent the data model.

A *data element* is a unit of data that is stored in the system. It can have different granularities. The smallest data element in the version control system is a *version.* A version is a particular revision of a file or directory. Versions of one line of development form a linear sequence called a *branch.* Branches are used to separate different development efforts and allow parallel development. For example, one branch may contain all the versions used to add a new feature to the software; another branch may be composed of the versions contributing to a software bug fixing. Branches of the same file or directory are organized into a version tree. Every tree has one main branch (called main), which represents the principal line of development. Files and directories are called *elements.* Unlike in a file system file and directory elements in a version control system are not flat. They have the version tree structure. Like in a file system, however, file and directory elements are also located somewhere in a directory tree.

A *data repository* is used to store different versions of files and directories. It also holds the derived data and meta-data associated with them. Data repositories at multiple sites form a *data repository family.* The data repository family stores data relevant to a single project – that is, the data elements in the repository family are all semantically related.

A *virtual workspace* is an environment where users can have access to a set of versions selected from the data repository. The versions are selected via a user-defined filter, which is a part of the environment. Most virtual workspace is used by one user for individual development. Some are shared by several users for integration or integration testing.

The data model is illustrated by an example in figure 1. In this figure, the directory element "/home/junxing" has one branch and two versions on this branch, 0 and 1. Version 1 contains a file element named "Hello.c". This element has four branches: main, bug102, feature23 and Linux_port. Each of its branches has several versions to store the revision history of a separate development line. As the diagram shows in the MVC system each element is represented as a version tree no matter it is a directory or file. The contents of directory versions indicate the sub directory and file elements they include. The directory tree is organized in this way just as it is designed in a file system.

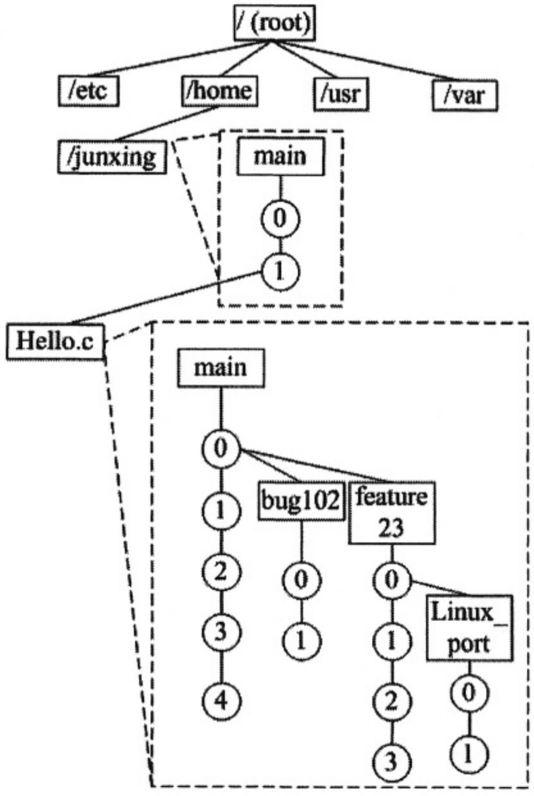


Figure 1. Data Model for the MVC System

The Network Model – The MVC system network model defines who accesses and/or manages the different versions of the data that are stored in the entire system. It comprises of *servers* and *clients* connected over a local area network at *local sites,* a number of which participate in the same MVC system.

We define a *local site* to be one that is responsible for one and only one data repository. The MVC system comprises of a number of such local sights. From the point of view of a local site, other local sites are called *remote sites.* Each local site consists of one server machine and several client machines that are connected in a local area network. A local site is under a single administrative control. The server at the local site maintains the data repository. The clients manage the virtual workspaces and accept and respond to users' requests. The clients also communicate with local or remote servers to retrieve, add, delete or update versions, meta-data and/or derived data.

A specific data repository at a server holds only one copy of the data. This copy has the latest local revision changes. However, it may not have the latest, up-to-date revision changes from other sites. In order to get the latest changes, repositories must be synchronized with each other. To synchronize the data at the local site with data at remote sites, each server communicates periodically with servers at remote sites. We adopt a peer-to-peer model for such server-server communication.

The network model for the MVC system is shown in figure 2.

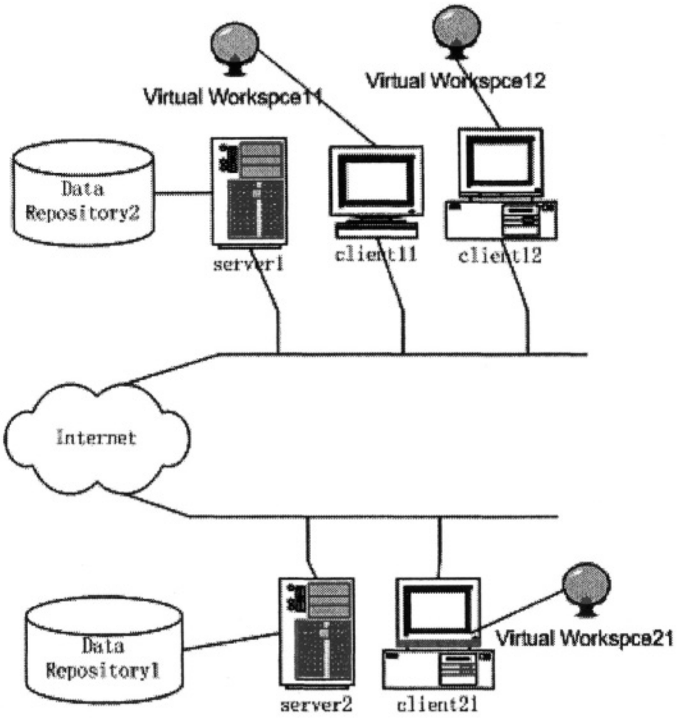


Figure 2. Network Model for the MVC System

The Access Model – The access model defines how the MVC system controls access to different versions in the data repository family. The access is of two types – local access and remote access. If a client contacts a local server, then this is a local access. For a local access a user must access the versions in a data repository via a virtual workspace. She needs to specify which versions of the files she wants to access. This is done by defining rules in the filter of a virtual workspace. We use a SQL like syntax for defining such rules. An example of such a rule on the data model shown in figure 1 is as follows.

SELECT Hello.c /main/LATEST

This rule will select the version numbered 4 (the latest version) on the main branch of Hello.c.

We define a *well-formed rule* to be one that retrieves only one version of any element. The virtual workspace is usable only after such well-formed rules are defined. When the user requests an operation on a file or directory element, the client that manages the workspace contacts a server to apply the operation to the version of the element selected by the filter. If the user wants to operate on a different version or on more than one version, she explicitly gives the version name in the request.

Sometimes the local server does not have the permissions to execute certain operations on the specified version. In this case a remote server has to be contacted; either the client can directly contact the remote server that has the permission to complete this request, or the client can request the local server to contact the relevant remote server. Now a server typically has to handle multiple client requests. So it is more efficient for a local server to gather together multiple client requests to access a remote server, and schedule it in a batch mode than to handle each client request as it comes. One the other hand, if we allow a client to directly access the relevant remote server then the client will be serviced promptly. For this reason we choose to have the client contact the remote server directly in the MVC system. This is termed as *remote client-server access.* The only time a local server needs to communicate with remote servers is to synchronize data in the containing repositories. This is called *remote server-server access.*

Access to different versions is achieved by executing a set of atomic operations. These are (i) get, (ii) checkout, (iii) check-in, and (iv) un-checkout.

- Get – Get is the action of getting the copy of a version from the data repository to a virtual workspace.
- Checkout – Checkout is the action of locking a branch in the data repository for adding a new version. This prevents other users from checking the same branch out, though they are not restricted from getting versions in the branch.
- Check-in – Check-in is the action of adding a new version to the branch locked by a checkout and then releasing the lock.
- Un-checkout – Un-checkout is the action of releasing the lock created by a checkout without adding a new version.

Check-in, checkout and un-checkout all are essentially write transactions. Checkout starts the transaction, check-in commits it, and un-checkout rolls back the transaction. The definition of checkout operation implies that a branch cannot be checked out by a user if it has already been checked out by another one. However, the data model of the repository family indicates that the local repository may not know if a branch has been checked out at other sites because it does not have the latest revision changes at other sites. This means the authorization need be coordinated among sites. A check-in or un-checkout operation is based on a previous checkout operation. The user who can check in or un-checkout a branch must be the one who checked it out. Optionally the system may allow others who are in charge of the branch or its contents, to execute the operation when the user is unavailable.

Checkin, checkout and un-checkout all require permissions similar to write permissions. The execute permission has additional connotation when it is applied to the objects in a MVC system. The execute permission of directory elements, file elements and branches is the permission to list or search their included objects, while the execute permission of versions is to run operations defined in owners' data. This is because owners' data is contained in files in a file system, but it is kept in versions in a MVC system.

A user is authorized at three layers in order to access a particular version. The first layer consists of virtual workspaces. Here the user needs permissions to operate on the temporary copies of versions she selected from the data repository. The second layer is the directory tree. Here she needs the permission to operation on the directory structure. The third layer is the version tree. Here she needs the permission to access a particular version on a specific branch, which presents a development line.

The authorization granularity is decided by the granularity of accessible objects. In the workspace layer the accessible objects are just virtual workspaces. In the directory tree layer the accessible objects are the data repository and elements (directories and files). In the version tree layer the accessible objects are branches and versions.

Finally, although in the local access model the MVC system may not need strong encryption mechanisms to ensure data confidentiality, they are needed for integrity and non-repudiation of data origin; strong cryptographic techniques are indispensable in the remote access model.

The User Account Management Model – This model defines how user account information is maintained across the entire MVC system. Every user in the system belongs to a primary site. All changes to user account information are made at the server at the primary site. To meet the requirements of parallel development at multiple sites, the account information at the primary site must be available at all sites and be consistent. This is achieved by a remote server-server access.

4. THE PROPOSED MVC SYSTEM

The proposed MVC system is developed by adopting and/or extending widely industry standards and COTS components.

We use X.500 directories [1] to represent the data model, network model and user account management model of the MVC system.

2. We extend the Lightweight Directory Access Protocol (LDAP) [8, 9, 10] to support the access model of the MVC system. In particular, we develop the get, checkin, checkout and un-checkout operations as extensions to LDAP.
3. We develop a novel authorization scheme for the MVC system by borrowing from and integrating the concepts of access control lists, role-based access control and the concept of mastership.
4. We adopt the Simple Authentication and Security Layer (SASL) [7] protocol to address the authentication problems in the MVC system. This protocol also helps to protect data confidentiality and integrity in the MVC system by virtue of its built in strong cryptographic techniques.

In the following we briefly discuss how we implement each of the above modules of the MVC system.

4.1 X.500 schemas for Data, Network and User Account Management Models

The MVC system stores information about version trees; thus it is natural to use another tree structure to implement the system. We use the Directory Information Tree (DIT) of X.500 directories to represent the system tree structure. The structure is defined X.500 schemas according to RFC 2252 specifications, which is based on BNF (Backus-Naur Form meta-language). For lack of space we show the declarations for only the security related elements. Since X.500 is used, each schema element for a project must have a globally unique Object Identifier (OID)[1]. For this discussion, the OIDs under 1.1 are used. This is in keeping with the conventions of ASN.1. Each element has at least one textual name. To reduce the potential for name clashes, the name prefix “mvc” is used in the convention. The following is an example of these definitions for a project eduColostate.

objectIdentifier eduColostateOID 1.1 ; Organization OID

[1] OIDs are basically strings of numbers. They are commonly found in protocols described by ASN.1. The formal definitions of OIDs come from ITU-T Rec. X.208 (ASN.1). OIDs are allocated in a hierarchical manner and managed by the IANA. For private experiments and research purposes the OIDs under the 1.1 branch are typically used.

```
objectIdentifier mvc eduColostateOID: 1 ; Name prefix and Application OID
objectIdentifier mvcObjClass mvc: 1    ; MVC object class
```

The Data Model – The data elements, data repository, data repository family and virtual workspace are all defined as object classes in X.500 directories. The type definition of the data element *version* is given in the following schema. Its object Id is derived by appending 1 to the object Id of "mvcObjClass". Its name is "mvcVersion". It has a text description and inherited from the superior object class "top". Each instance of this class must have five attributes: "mvcUserId", "mvcGroupId", "mvcUserPermission", "mvcGroupPermission" and "mvcOtherPermission". These attributes are introduced in the authorization scheme section. Other data model components can be defined similarly. For lack of space we omit them from here.

```
objectclass
  (
    mvcObjClass:1
    NAME 'mvcVersion'
    DESC 'Data Version Type'
    SUP top
    MUST  (
              mvcUserId $
              mvcGroupId $
              mvcUserPermission $
              mvcGroupPermission $
              mvcOtherPermission
          )
  ); Version Class
```

The Network Model – To conform to the MVC network model, each MVC site should have one server sub tree and multiple client sub trees. The server sub tree is composed of data model components: version, branch, element, data repository and data repository family. The client sub tree consists of one root element and several workspace elements. The root element is used to organize workspace elements and identify the client. Its type doesn't concern the MVC security, so the definition is not given here.

The User Account Management Model – User account management service required by MVC systems are represented using user schemas defined in RFC 2256. Since these are very well defined and standardized we do not discuss them here.

4.2 LDAP Extensions to Support the Access Model

The MVC framework uses X.500 directories to represent the MVC system structures. LDAP is the access protocol to X.500 directories. Thus it is reasonable to support the access model of the MVC system by extending LDAP in the secure framework. Another advantage gained by extending LDAP is the convenient access to SASL. Since SASL is the association security services of LDAP, the framework can use it directly instead of building a new one from scratch.

The MVC framework uses a special operation in LDAPv3 support the extension. This operation is defined in order to allow additional operations to be defined for services not available elsewhere in LDAP protocol. It is described in ASN.1 (Abstracted Syntax Notation 1) [3, 4], and is typically transferred using a subset of ASN.1 Basic Encoding Roles. The special operation that we use is as follows.

```
ExtendedRequest ::= [APPLICATION 23]
SEQUENCE
  {
    requestName [0] LDAPOID,
    requestValue [1] OCTET STRING OPTIONAL
  }

ExtendedResponse ::= [APPLICATION 24]
SEQUENCE
  {
    COMPONENTS OF LDAPResult,
    responseName  [10] LDAPOID OPTIONAL,
    response  [11] OCTET STRING OPTIONAL
  }
```

The MVC access model operations get, checkout, check-in and un-checkout, are implemented using the above LDAP extended operation definition.

4.3 Authorization Scheme for the MVC System

Problems with authorization are often caused by the intention to reuse the authorization scheme of a native operating system. To avoid them, we provide the MVC system with its own authorization scheme. Due to the system's special requirements, this scheme combines three authorization mechanisms: ACL (Access Control List), RBAC (Role Based Access Control) and mastership.

Access Control List – ACL is used as the basic authorization mechanism. There are two reasons to use ACL. Firstly, MVC systems are object centralized and subject distributed systems. Secondly,it makes simulating file systems easier. Most file systems use ACL, so if the MVC system also uses ACL it will be easier for it to translate the permissions of a specific version in the data repository into the permissions of the corresponding file in the virtual workspace. ACL is defined as attribute types in the following LDAP/X.500 schema. In the first line of the schema the object Id of “mvcAttType” is created by appending 2 to the object Id of “mvc”. There are two attribute types. One is used to represent the user Id, and another is for the user permission. Their object Ids are created by extending the Id of “mvcAttType”. They use the matching rule of numeric strings. They have the syntax of integers. And they may only hold one single value. Similarly the group Id, group permission and other’s permission can be defined. These definitions consist of an ACL implementation in X.500 directories.

```
objectIdentifier mvcAttType mvc:2 ; MVC attribute type
attributetype
(
        mvcAttType: 1
        NAME 'mvcUid'
        DESC 'User Id'
        EQUALITY numericStringMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
        SINGLE-VALUE
)   ; User Id Type

attributetype
(
        mvcAttType:2
        NAME 'mvcUperm'
        DESC 'User Permission'
        EQUALITY numericStringMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
        SINGLE-VALUE
)   ; User Permission Type
```

In the data model design, every object class has mandatory attributes of ACL attribute types. This means that every object in the MVC system is in

ACL control, so the system's access control granularity is as fine as the accessible objects.

Role Based Access Control – ACL is not enough to meet the MVC system's authorization requirements. Operations such as checkin and un-checkout require the system to check if the requestor has certain relationship with the user who executed the corresponding checkout operation. This kind of requirements calls for Role Based Access Control. This mechanism makes use of the attribute types and object classes defined in the schema below.

```
attributetype
(
        mvcAttType:6
        NAME 'mvcOperationId'
        DESC 'Operation Id'
        EQUALITY numericStringMatch
        SYNTAX
1.3.6.1.4.1.1466.115.121.1.27
        SINGLE-VALUE
); Operation Id Type
```

```
attributetype
(
        mvcAttType:7
        NAME 'mvcUids'
        DESC 'User Id List'
        EQUALITY numericStringMatch
        SYNTAX
1.3.6.1.4.1.1466.115.121.1.27
); User Id List Type
```

```
objectclass
(
        mvcObjClass:6
        NAME 'mvcOperationRoles'
        DESC 'Roles can execute the
Operation'
        SUP top
        MUST mvcOperationId
); Roles allowed for an operation
```

```
objectclass
(
        mvcObjClass:7
        NAME 'mvcRoleUsers'
        DESC 'Users in the Role'
        SUP top
        MUST mvcUids
); User Ids in a role
```

The above types and classes are used to build RBAC trees. A RBAC tree is a directory information sub tree that has only a root and leaves. The root is an entry of " mvcOperationRoles" type. It has an "mvcOperationId" attribute that identifies the operation. The leaves are entries of "mvcRoleUsers " type. Each of them has an "mvcUids" attribute that includes all user Ids in the role. RBAC trees are used to implement RBAC. One operation such as checkout creates and inserts a RBAC tree to the checked out branch. Another operation such as checkin or un-checkout examines the RBAC tree in the target branch. If there is no such tree or the requestor Id can't be found in the tree, the operation fails; otherwise the operation is executed and the RBAC tree is deleted.

Mastership – is introduced in the MVC system to coordinate the authorization among multiple sites. This is needed because changes made at different sites can potentially conflict when they are imported into one data repository; the authorization must be coordinated to prevent conflicts from happening. Mastership is "site based access control". It ensures that only one site in a repository family has the permission to change a controlled object (e.g. branch) at any given time. This exclusive permission to modify ensures that no parallel changes can be made to controlled objects; thus conflicts are avoided. Users at sites that do not have mastership-permission of an object have two access modes to make changes. The active mode allows one to modify the object using remote access model at the site that has the permission. The passive mode allows one to request the mastership-permission to be transferred to the local site. This scheme is made possible by the following attribute types and object classes:

```
attributetype
  (
     mvcAttType:8
     NAME 'MVC Mastership'
     DESC 'Master Site Id'
     EQUALITY numericStringMatch
     SYNTAX  1.3.6.1.4.1.1466.115.121.1.27
     SINGLE-VALUE
  ); A type used to define attributes that identify the sites that have the
mastership-permissions of the controlled objects
```

```
objectclass
  (
     mvcObjClass:8
     NAME 'MVC MastershipObject'
     DESC 'Mastership Container'
     SUP top
     MUST MVCMastership
  ); A class used to define entries that hold the attribute of the previous type.
```

4.4 Authentication, Confidentiality and Integrity Measures

Simple Authentication and Security Layer protocol is used to provide both strong authentication mechanisms needed in the remote access model and regular mechanisms required in the local access model of the MVC system. To ensure secure remote access, the MVC system imposes the following requirements to LDAP configuration: Remote MVC operations are allowed only when SASL mechanisms whose Security Strength Factor (SSF) are greater than 1, are in effect. Remote MVC operations are allowed only when mechanisms that are able to negotiate a security layer are in force. To ensure secure remote access, the mechanisms that support the security layer negotiates a privacy protection layer after the successful authentication for remote MVC operations. There are no mandatory configuration requirements for the local access model.

5. CONCLUSION

In this paper we look into multi-sited version control systems. Such systems are very important in distributed collaborative environments where many development teams work together on a large software system and the system goes over several versions before being finally available. We identify the security requirements of multi-sited version control systems in general. We observe that it is more important that such systems ensure the integrity of the software versions rather than their confidentiality. We then identify the security deficiencies in current implementations of multi-sited version control systems. We propose a new framework for secure multi-sited version control. The specific contributions include:

- Using X.500 directories and extending LDAP v3 to provide version control service at multiple sites.
- The application of SASL to improve the security of MVC systems.
- The application of the concept of mastership in the remote access model so that conflicts due to concurrent access is eliminated.
- An authorization scheme combining ACL, RBAC and mastership.

The paper also uses the LDAP extension for Internet Domain Name service as the reference for extending the protocol.

REFERENCE:

[1] ISO/IEC 9594-1, "X.500 The Directory: Overview of Concepts, Models and Services", International Standards Organization, 1993.

[2] ISO/IEC 9594-2, "X.501 The Directory: Models", International Standards Organization, 1993.

[3] ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", International Telecommunication Union, 1994.

[4] ITU-T Rec. X.690, "Specification of ASN.1 Encoding Rules: Basic, Canonical and Distinguished Encoding Rules", International Telecommunication Union, 1994.

[5] J. G. Meyers, "Simple Authentication and Security Layer (SASL)", RFC 2222, Network Working Group, The Internet Society, October 1997.

[6] M. Wahl, "A Summary of the X.500(96) User Schema for Use with LDAPv3", RFC 2256, Network Working Group, The Internet Society, December 1997.

[7] M. Wahl, A. Coulbeck, T. Howes and S. Kille, "Lightweight Directory Access Protocol (V3): Attribute Syntax Definitions", RFC 2252, Network Working Group, The Internet Society, December 1997.

[8] M. Wahl, T. Howes and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, Network Working Group, The Internet Society, December 1997.

[9] P. Cederqvist, et al. "Version Management with CVS", The Open Group, Available from http://www.cvshome.org/docs/manual/cvs-1.11.6/cvs.html

[10] R. Sandhu, "Role-Based Access Control", in Advances in Computers, volume 48, M. Zerkowitz editor, Academic Press, 1998.

[11] R. Sandhu and P. Samarati, "Access Control: Principles and Practice", IEEE Communications, 32(9), 1994.

[12] Rational Software Corporation, "Rational® ClearCase Multisite® Documentation", October 2001.

[13] S. Kille, M. Wahl, A. Grimstad, R. Huber and S. Sataluri, "Using Domains in LDAP/X.500 Distinguished Names", RFC 2247, Network Working Group, The Internet Society, January 1998.

[14] Unix SCCS, "Source Code Control System", The Regents of the University of California, 1986.

[15] W. F. Tichy, "RCS – A System for Version Control", Software – Practice and Experience, 15(7), 1985.

Integration of Integrity Constraints in Database Federations

Herman Balsters, Bert de Brock
University of Groningen, Faculty of Management and Organization, P.O. Box 800, 9700 AV Groningen, The Netherlands, {h.balsters, e.o.de.brock}`@bdk.rug.nl`

Abstract: A database federation provides for tight coupling of a collection of heterogeneous legacy databases into a global integrated system. A major problem in constructing database federations concerns maintaining quality and consistency of data on the global level of the federation. In particular, the integration of integrity constraints within component legacy databases into a single global schema is a process that is prone to incompleteness and inconsistency. Moreover, additional inter-database constraints between the various component legacy databases often also have to be taken into account to complete the integration process. Our approach to coupling of component databases into a global, integrated system is based on the concept of mediation. Our major result is that mediation in combination with a so-called integration isomorphism integrates component schemas without loss of constraint information; i.e., integrity constraints available at the component level remain intact after integration on the global level of the federated database. Our approach to integration also allows for specification of additional inter-database constraints between the various component legacy databases. We therefore can handle consistency not only on the local level of component databases, but also consistency on the global level between the various component databases. We shall describe a general semantic framework for specification of database federations based on the UML data model. This data model will prove to offer an elegant and powerful framework for analysis and design of database federations, including integration of integrity constraints.

Key words: Databases, distribution, integrity control, legacy systems, systems integration, semantics, consistency, data modeling, object-orientation, UML

1. INTRODUCTION

Modern information systems are often distributed in nature. Data and services are often spread over different component systems wishing to cooperate in an integrated setting. Cooperation of component systems in one integrated information system is becoming more and more important since information is often spread over different databases in one organization (or even spread over different organizations). Such information systems involving integration of cooperating component systems are called *federated information systems*; if the component systems are all databases then we speak of a *federated database system* ([ShL90]). This tendency to build integrated, cooperating systems is often encountered in applications found in EAI (Enterprise Application Integration), which typically involve several, usually autonomous, component systems (data and service repositories), with the desire to query and update information on a global, integrated level. In this paper we will address the situation where the component systems are so-called legacy systems; i.e. systems that are given beforehand and which are to interoperate in an integrated single framework in which the legacy systems are to maintain as much as possible their respective autonomy.

A major obstacle in designing interoperability of legacy systems is the heterogeneous nature of the legacy components involved. This heterogeneity is caused by the design autonomy of their owners in developing such systems. We are faced with major problems when trying to maintain data quality and data consistency in the integration process. Many of these problems deal with integration of the schemas of the component databases involved, and in particular with resolving consistency conflicts rising from integrity constraints ruling within the component databases.

To address the problem of interoperability the term *mediation* has been defined [Wie95]. A database federation can be seen as a special kind of mediation, where all of the data sources are (legacy) databases, and the mediator offers a mapping to a (virtual) DBMS-like interface. In our paper we will consider a *tightly-coupled approach* to database mediation, in which a global integrated schema of the federation is maintained. We base our approach on the *"Closed World Assumption"* (CWA, [Rei84]), where the integrated database is to hold -in some manner- the "union" of the data in the underlying component databases. The user of the federated system will be offered the impression that he is working with a monolithic homogeneous database system, while in fact this system basically resembles an interface, mapping interactions on the federated level to actions on the existing local database components. More precisely, the federated database will consist of an integrated *database view* on top of the existing legacy database components. We concentrate on problems concerning integration of component legacy schemas on the level of the mediator. Schema integration requires the definition of relationships between schema elements of component systems. Detection and definition of such relationships can be heavily complicated by so-called *semantic heterogeneity* [DKM93,Ver97]. Semantic heterogeneity refers to disagreement about the meaning, interpretation, or intended use of related data. In this paper we will focus on the UML/OCL data model to tackle the problem of integrating semantic heterogeneity. UML/OCL offers a high-level specification language and is equipped with a unique combination of high expressiveness with a large degree of precision. Also, UML is the *de facto* standard language for analysis and design in object-oriented frameworks, and is being employed more and more for analysis and design of information systems, in particular information systems based on databases and their applications. OCL ([WK03, OCL2.0]) complements UML by providing a language for formally expressing integrity constraints of a model. In this paper, we will assume that component databases -e.g. relational databases - can somehow be modeled in the UML/OCL-framework ([BP98]).

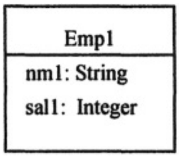
The paper describes a particular mediating system integrating component schemas without loss of constraint information; i.e., no loss of constraint information available at the component level may take place as result of integrating on the level of the virtual federated database. Furthermore, we will show how to deal with integrity constraints not only within, but also between various component databases (the so-called *inter-database constraints*). Integration conflicts are treated in a tightly-coupled environment, and are resolved by introducing a so-called *integration isomorphism* ([Bal03]).

The paper is concluded by a description of general design principles for database federations.

2. UML/OCL AS A SPECIFICATION LANGUAGE FOR DATABASES

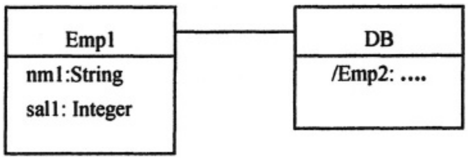
Recently, researchers have investigated possibilities of UML as a modeling language for (relational) databases. [BP98] describes in length how this process can take place, concentrating on schema specification techniques. [DH99, DHL01] investigate further possibilities by employing OCL for specifying integrity constraints and business rules within the context of relational databases. The idea is that OCL provides expressiveness in terms of relatively abstract set definitions that should prove to be sufficient to capture the general notion of (relational) database view. In the more specific context of relational databases and OCL, [DH99] offers a framework for representing integrity constraints within the relational data model ([GUW02]). Our application area is focused on Federated Databases, where legacy databases are to interoperate by employing a so-called mediating system. This mediating system can be considered as an integration of a set of certain database views defined on the component legacy database systems. In particular, we will concentrate on data representations in the UML data model, and employ OCL as a language for expressing accompanying constraint specifications, both on the component level and on the global level of a federation.

An important notion in modeling databases is that of a database view. To illustrate this notion, consider the case that we have a class called Emp1 with attributes nm1 (of type String) and sal1 (of type Integer), indicating the name and salary of an employee object belonging to class Emp1. In UML, this can be represented as follows



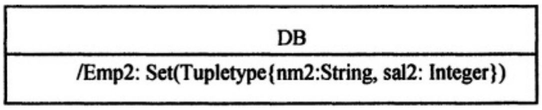
Now consider the case where we want to add a class, say Emp2, which is defined as a class whose objects are completely derivable from objects coming from class Emp1. The calculation is performed in the following manner. Assume that the attributes of Emp2 are nm2 and sal2 respectively (indicating name and salary attributes for Emp2 objects), and assume that for each object e1:Emp1 we can obtain an object e2:Emp2 by stipulating that e2.nm2=e1.nm1 and e2.sal2=(100 * e1.sal1). By definition the total set of instances of Emp2 is the set obtained from the total set of instances from Emp1 by applying the calculation rules as described above. Hence, class Emp2 is a *view* of class Emp1 (showing all salaries in cents), in accordance with the concept of a view as known from the relational database literature. In UML terminology ([BP98]), we can say that Emp2 is a *derived class,* since it is completely derivable from other already existing class elements in the model description containing model type Emp1.

We will now show how to faithfully describe Emp2 as a derived class in UML/OCL (version 2.0) in such a way that it satisfies the requirements of a (relational) view. First of all, we must satisfy the requirement that the set of instance of class Emp2 is the result of a calculation applied to the set of instances of class Emp1. The basic idea is that we introduce a (database-) class called DB that has an association to class Emp1, and that we define within the context of the database DB an attribute called Emp2. A database object will reflect the actual state of the database, and the system class DB will only consist out of one object in any of its states. Hence the variable *self* in the context of the class DB will always denote the actual state of the database that we are considering. In the context of this database class we can then define the calculation obtaining the set of instances of Emp2 by taking the set of instances of Emp1 as input.



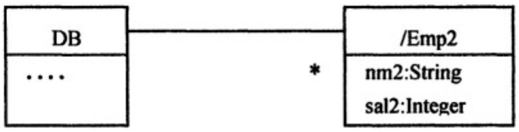
```
context  DB
def: Emp2: Set(Tupletype{nm2:String, sal2: Integer}) =
              (self.Emp1 -> collect(e:Emp1 |
              Tuple{nm2=e.nm1, sal2=(100*e.sal1)}))-> asSet
```

In this way, we explicitly specify Emp2 as the result of a calculation performed on the base class Emp1. Graphically, we could represent Emp2 as follows



(1)

where the slash-prefix of Emp2 indicates that Emp2 is a *derived* table. Since in practice such a graphical representation could give rise to rather large box diagrams (due to lengthy type definitions), we will use the following (slightly abused) graphical notation (2) to indicate this derived class

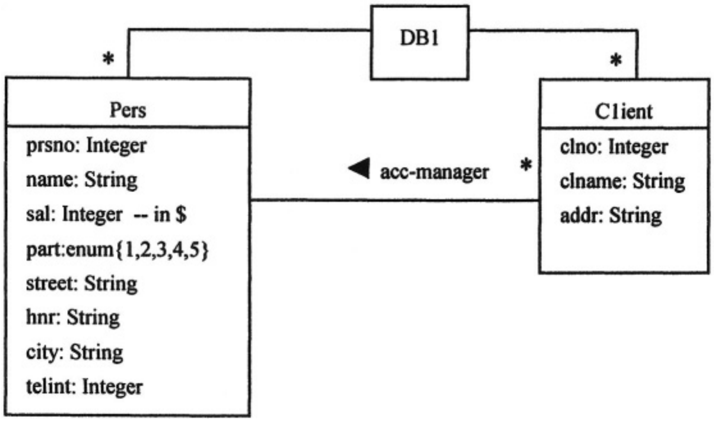


(2)

The intention is that these two graphical representations are to be considered equivalent; i.e., graphical representation (2) is offered as a diagrammatical convention with the sole purpose that it be formally equivalent *(translatable)* to graphical representation (1). Full details regarding representations of database views in UML/OCL can be found in [Bal03b].

3. COMPONENT FRAMES

We can also consider a complete collection of databases by looking at so-called component frames, where each (labeled) component is an autonomous database system (typicaly encountered in legacy environments). As an example consider a component frame consisting of two separate component database systems: the Client-Relationship-Management (CRM)-database DB1, and the Sales-database DB2 below



DB1 consists of two classes. For example, `Pers` is the class of employees responsible for management of client resource; `part` indicates that employees are allowed to work part time; hnr indicates house number; telint indicates internal telephone number; acc-manager indicates the employee (account manager) that is responsible for some client's account. The rest of the features of DB1 speak for themselves. We furthermore assume that database DB1 has the following integrity constraints

```
context Pers inv:
Pers.allInstances --> isUnique (p: Pers | p.prsno)
sal <= 1500
telint >= 1000  and  telint <= 9999

context Client inv:
Client.allInstances --> isUnique (c: Client | c.clno)
```

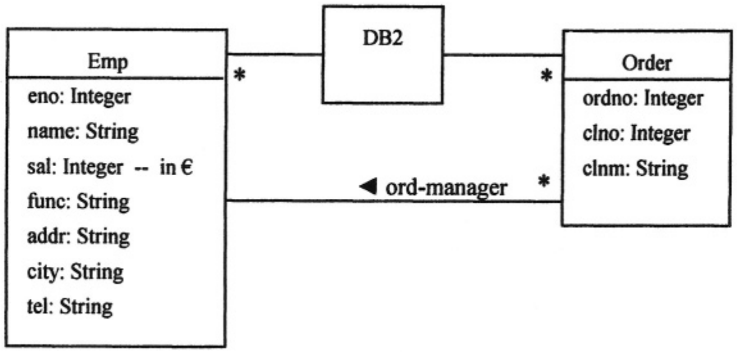
Informally, these integrity constraints state that

- Persons have unique `prsno`-values;
  their salaries may not exceed 1500 dollars;

their (internal) telephone numbers have values between 1000 and 9999

- Clients have unique `clno`-values

The second database is the so-called Sales-database DB2



Most of the features of DB2 also speak for themselves. We offer a short explanation of some of the less self-explanatory aspects: `Emp` is the class of employees responsible for management of client order; `func` indicates that an employee has a certain function within the organization; `Ord-manager` indicates the employee (account manager) that is responsible for some client's order. We assume that this second database has the following integrity constraints:

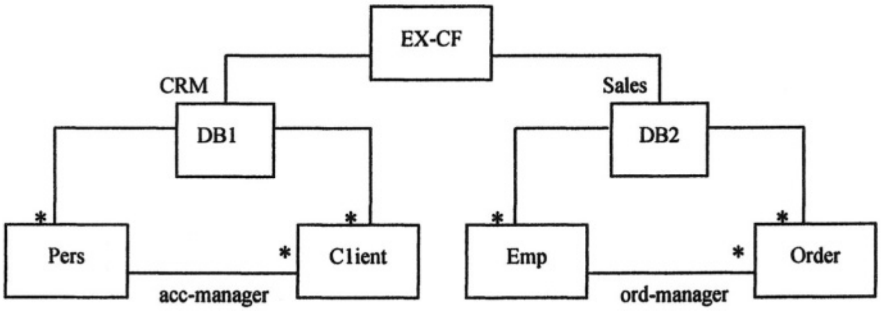
```
context Emp inv:
Emp.allInstances --> isUnique (p: Emp | p.eno)
sal >= 1000
tel.size <= 16

context Order inv:
Order.allInstances --> isUnique (c: Order | c.ordno)
Order.allInstances --> forall(c: Order | c.ord-manager.func =
'Sales')
```

Informally, these integrity constraints state that

- Employees have unique `eno`-values;
  their salaries are at least 1000 euros;
  their telephone numbers consist of strings of no more than 16 characters
- Orders have unique `ordno`-values;
  employees managing some order must have `'sales'` as `func`-value

We can now place the two databases DB1 and DB2 without confusion into one component frame EX-CF as seen below



4. SEMANTIC HETEROGENEITY

We are faced with major problems when trying to maintain data quality, data completeness and data consistency in the integration process. Many of these problems deal with integration the schemas of the component databases involved, and in particular with resolving consistency conflicts rising from integrity constraints ruling within (and possibly between) the component databases. The problems we are facing when trying to integrate the data found in legacy component frames are well-known and are extensively documented (cf. [ShL90]). We will focus on one of the large categories of integration problems coined as *semantic heterogeneity* (cf. [Ver97]). Semantic heterogeneity deals with differences in intended meaning of the various database components. Integration of the source database schemas into one encompassing schema can be a tricky business due to: *homonyms* (solved by mapping same name occurrences to different names); *synonyms* (mapping different names to one common name); *different representations* (data conversion to a common value); *default values* (adding default values for new attributes); *missing attributes* (adding new attributes in order to discriminate between certain class objects); *absent subclasses* (creation of a common superclass and subsequent accompanying subclasses).

By homonyms we mean that certain names may be the same, but actually have a different meaning (different semantics). Conflicts due to homonyms are resolved by mapping two same name occurrences to different names in the integrated model. In the sequel, we will refer to this solution method as **hom**. Synonyms, on the contrary, refer to certain names that are different, but have in fact the same semantics. Synonyms are treated analogously, by mapping two different names to one common name; this solution method is refered to by **syn**.

In the integration process, one often encounters the situation where two attributes have the same meaning, but that their domain values are differently represented. For example, the two attributes sal in the Pers and the Emp class of databases DB1 and DB2, respectively, both indicate the salary of an employee, but in the first case the salary is represented in the currency dollars ($), while in the latter case the currency is given in euros (€). What we can do, is to convert the two currencies to a common value (e.g. $, invoking a function convertTo$). Applying a conversion function to map to some common value in the integration process, is indicated by **conv**.

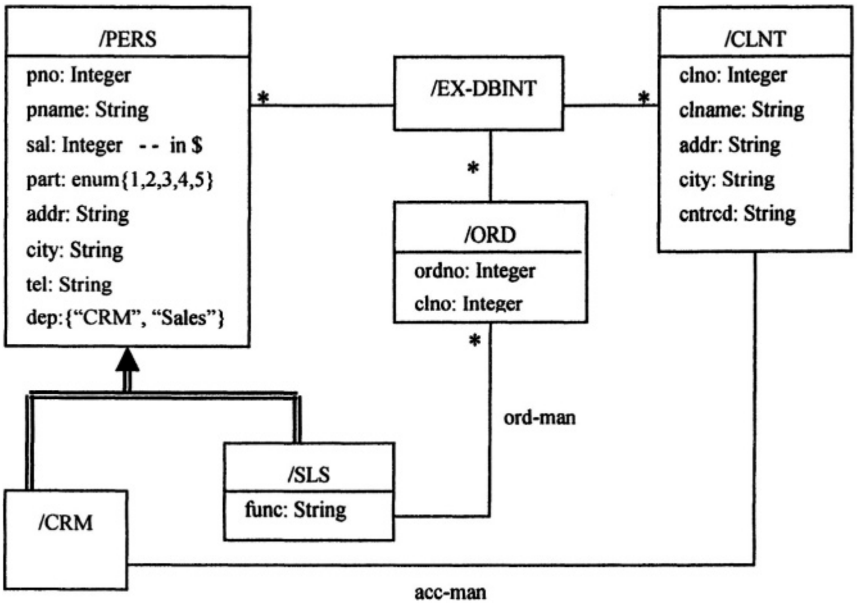
Sometimes an attribute in one class is not mentioned in another class, but it could be added there by offering some suitable default value for all objects inside the second class. As an example, consider the attribute `part` in the class Pers (in DB1): it could also be added to the class Emp (in DB2) by stipulating that the default value for all objects in Emp will be 5 (indicating full-time employment). Applying this principle of adding a default value in the integration process, is indicated by **def**.

The integration of two classes often calls for the introduction of some additional attribute, necessary for discriminating between objects originally coming from these two classes. This will sometimes be necessary to be able to resolve seemingly conflicting constraints. As an example, consider the classes Pers (in DB1) and Emp (in DB2). Class Pers has as a constraint that salaries are less than 1500 (in $), while class Emp has as a constraint that salaries are at least 1000 (in €). These two constraints seemingly conflict with each other, obstructing integration of the Pers and the Emp class to a common class, say PERS. However, by adding a discriminating attribute `dep` indicating whether the object comes from the CRM or from the SLS department, one can differentiate between two kinds of employees and state the constraint on the integrated level in a suitable way. Applying the principle of adding a discriminating attribute to differentiate between two kinds of objects inside a common class in the integration process, will be indicated by **diff**. The situation of a missing attribute mostly goes hand in hand with the introduction of appropriate subclasses. For example, introduction of the discriminating attribute `dep` (as described above), entails introduction of two subclasses, say CRM and SLS of the common superclass PERS, by listing the attributes, operations, and constraints that are specific to CRM- or SLS-objects inside these two newly introduced subclasses. Applying the principle of adding new subclasses in the integration process, is indicated by **sub**.

We note that in [Ver97] the problems regarding semantic interoperability were analyzed in a high-level object-oriented langauage called TM ([BBZ93], specifically tailored for object-oriented analysis and design of databases. In [BS98] this approach using TM was further pursued, providing a basis for automatic verification of transaction correctness in OO databases.

5. THE INTEGRATED DATABASE

We now offer our construction of a virtual database EX-DBINT, represented in terms of a derived class in UML/OCL. The database we describe below, intends to capture the integrated meaning of the features found in the component frame described earlier



We have introduced a superclass PERS with two subclasses CRM and SLS. A subclass inherits all of the attributes of a superclass, and possibly also includes (e.g., the class SLS) attributes of its own. Furthermore, this database has the following integrity constraints:

```
context PERS inv:
PERS.allInstances ->
forall(p1, p2: PERS | (p1.dep=p2.dep and p1.pno=p2.pno)
implies p1=p2)
PERS.allInstances ->
forall(p:PERS |
 (p.oclIsTypeOf(SLS) implies (p.sal >= 1000.convertTo$ and
part=5)) and (p.oclIsTypeOf(CRM) implies p.sal <= 1500 ))
```

```
tel.size <= 16

context CLNT inv:
Clnt.allInstances --> isUnique (c: CLNT | c.clno)

context ORD inv:
Order.allInstances --> isUnique (o: ORD | o.ordno)
ord-manager.func = 'Sales'
```

Informally, the integrity constraints on the PERS-class state that

- PERS-objects have unique values for the *combination* of attributes (dep,pno)
- If a PERS-object is also a SLS-object, then its salary is at least the dollar-equivalent of 1000 euros
- If a PERS-object is also a CRM-object, then its salary is at most 1500 dollars
- Its telephone number consists of strings of at most 16 characters

In order to control the quality, completeness, and consistency of the data, we shall now carefully analyze the specification of this (integrated) database EX-DBINT, and check whether it captures the intended meaning of integrating the classes in the component frame EX-CF and resolves potential integration conflicts.

Conflict 1: Classes Emp and Pers in EX-CF have partially overlapping attributes, but Emp has no attribute part yet, and one still needs to discriminate between the two kinds of class objects (due to specific integrity constraints pertaining to the classes Emp and Pers). Our solution in DBINT is based on applying **syn + def + diff + sub**: map to common class name (PERS); add a default value (to the attribute part); add an extra discriminating attribute (dep); introduce suitable subclasses (CRM and SLS).

Conflict 2: Attributes prsno and eno intend to have the same meaning (a *key constraint,* entailing uniquely identifying values for employees, for Emp- and Pers-objects). Our solution in DBINT is therefore based on applying **syn + diff** (map to common attribute name (pno); introduce extra discriminating attribute (dep)) and enforce uniqueness of the value combination of the attributes pno and dep.

Conflict 3: Attributes sal (in Pers) and sal (in Emp) partially have the same meaning (salaries), but the currency values are different. Our solution is based on applying **conv** (convert to a common value).

Conflict 4: The attribute combination of `street` and `hnr` (in Pers) partially has the same meaning as `addr` in Emp (both indicating address values), but the domain values are differently formatted. Our solution is therefore based on applying **syn + conv** (map to common attribute name and convert to common value).

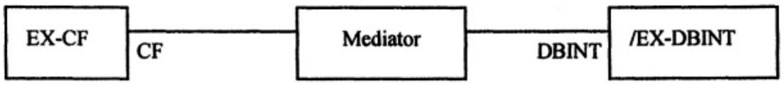
Conflict 5: Attributes `telint` (internal telephone number) and `tel` (general telephone number) partially have the same meaning, but the domain values are differently formatted. Our solution is therefore based on applying **syn + conv** (map to common attribute name and convert to a common format).

6. INTEGRATING BY MEDIATION

Our strategy to integrate a collection of legacy databases is based on the following principle:

An integrated database is intended to hold exactly the "union " of the data in the source databases in the component frame CF

This principle is also known as the Closed World Assumption for integrated databases (**CWA-INT**, cf. [Hull97]), and is a direct extension of the CWA first introduced in [Rei84] for monolithic databases. In [Bal03] we offer a mathematical basis for CWA-INT, by introducing a so-called *integration isomorphism,* which aims at correctly mapping elements of the component frame CF to the integrated database DBINT. In [Bal03] we furthermore offer an elaborate description of a UML/OCL model containing a class, called the *mediator,* explicitly relating CF to DBINT, as depicted below



Our construction of the mediator class closely reflects the basic ideas of mediation as first described in [Wie95]. Further elaboration on our construction of such a mediator class (and the accompanying integration isomorphism) is beyond the scope of this paper; the interested reader is refered to [Bal03] for more details.

7. INTER-DATABASE (OR COMPONENT-FRAME) CONSTRAINTS

Additional information analysis might reveal the following two wishes regarding the consistency of data in the component frame EX-CF:

(1) Nobody is registered as working for both the CRM and Sales department; i.e., these departments have no employees in common
(2) Client numbers in the Sales database should also be present in the CRM database

This entails that certain integrity constraints should be added, and in this case on the level of the class EX-CF, since these integrity constraints hold between two *databases* DB1 and DB2. Such integrity constraints are called *inter-database constraints,* or *component-frame constraints.* We now offer a specification of the two inter-database constraints mentioned above. We first offer some appropriate abbreviations (using a so-called **`let`**-construct), and then offer the two constraint specifications.

The following constraint states that there are no common `prsno`- and `eno`-values, and that each `clno`-value in the Order class corresponds to some `clno`-value in the Client class in the context of the class EX-CF

```
context  EX-CF  inv:
let  Pers-nrs = ((self.CRM.Pers.allInstances     ->
                collect (p:Pers | p.prsno))      -> asSet)
let  Emp-nrs  = ((self.Sales.Emp.allInstances    ->
                collect (e:Emp | e.eno))         -> asSet)
let  Cl-nrs   = ((self.CRM.Client.allInstances   ->
                collect (c:Client | c.clno))     -> asSet)
let  Ord-nrs  = ((self.Sales.Order.allInstances  ->
                collect (o:Order | o.clno))      -> asSet)
in
 (Pers-nrs->Intersect(Emp-nrs)) -> isEmpty  and              (1)
 Ord-nrs  ->  forall(o:Integer  |  (Cl-nrs  ->  exists(c:Integer  |
o=c)))                                                        (2)
```

We are now, of course, also faced with the obligation to suitably introduce these inter-database constraints in the integrated database DBINT. This can now be done in a straightforward manner, as illustrated below for the first constraint

```
context  EX-DBINT  inv:
let  X =  (self.CRM.allInstances -> collect(c:CRM | c.pno))
                                  -> asSet
let  Y =  (self.SLS.allInstances -> collect(s:SLS| s.pno))
                                  -> asSet
in
 (X->Intersect(Y))  -> isEmpty
```

Informally, this constraint states that there are no `pno`-values common to the CRM-class and the SLS-class.

As the examples offered above illustrate, OCL offers a powerful means to specify inter-database constraints in a very general manner in the context of our approach.

8. HEURISTICS: FROM SPECIFIC EXAMPLES TO A GENERAL APPROACH

This section concerns a discussion on methodology, in which we attempt to move from specific examples to a general approach in constructing a database federation from a collection of legacy databases, in order to maintain quality, completeness, and consistency of the data.

We adopt the following strategy to integrate a collection of legacy databases (collected in a component frame) into a virtual integrated database: (1) create a tightly-coupled architecture of the federated system, and (2) abide to the principle of the Closed World Assumption of Database Integration (CWA-INT). CWA-INT is established by supplying an *integration isomorphism,* mapping from the component frame to the virtual integrated database ([Bal03]).

In practice, fulfilling this strategy can often be a challenging demand, but without eventually realizing both aspects, the resulting federated database will fall short due to incorrect query results and inadequate constraint integration (resulting in loss of completeness and/or consistency).

We now offer some heuristics concerning the realization of the isomorphic mapping from component frame to integrated database. The construction of this isomorphic mapping from the component frame to the virtual integrated database cannot –in principle- be determined beforehand in algorithmic terms. By this we aim to say that given some set of conflicts in moving from the components to the integrated federated schema, it is usually an illusion to state that there exists an indisputable *algorithm* determining how those conflicts are resolved. On the contrary, establishing the specification of a suitable integration isomorphism usually, in terms of a formal specification (cf. [Bal03]), demonstrates the mostly *ad hoc* nature of resolving the conflicts at hand, reflecting the need for a *business semantics* to reach an eventual solution. For example, the resolution of the conflict of the currencies dollar ($ in DB1) and euro (€ in DB2), deciding which currency is to be taken on the common integrated level is basically *ad hoc,* and has to be offered by the business. In general, the process of constructing the formal specification of an isomorphism between the component frame and the virtual integrated database constantly has to be guided by knowledge of relevant business semantics.

Equipped with knowledge of relevant business semantics, we can proceed by following a short step-by-step guideline (constituting a heuristics, *not* an algorithm) for constructing a virtual integrated database from a collection of legacy databases, as described below:

1. Specify the details of the Component Frame **CF**
2. Analyse semantic heterogeneity: detect conflicts due to *Homonyms, Synonyms, Different representations, Default Values, Missing Attributes,* and *Subclassing*
3. Construct an integrated schema DBINT (applying the principles of **syn, hom, conv, def, diff,** and **sub)**
4. Introduce a **mediator class**
5. Enforce ***CWA-INT,*** by constructing an ***integration isomorphism*** (via the mediator class) between CF and DBINT
6. Add possible **inter-database constraints** in CF, and incorporate them into DBINT

This guideline –though systematic- is not algorithmic in nature. Applying the principles set out in this guideline will often demand the necessary creativity from the database modeler in close cooperation with a business expert, who has sufficient knowledge of the specific business domain. Apart from these challenges (which apply to most modeling methodologies), our guideline can offer a powerful methodology in moving from a collection of legacy systems to a correctly integrated database system maintaining quality, completeness, and consistency of the involved data.

REFERENCES

[AB01] Akehurst, D.H., Bordbar, B.; On Querying UML data models with OCL; «UML» 2001 4th Int. Conf., LNCS 2185, Springer, 2001

[Bal03] Balsters, H.; Object-oriented modeling and design of database federations; SOM Report, University of Groningen, 2003

[Bal03b] Balsters, H. ; Modelling database views with derived classes in the UML/OCL-framework ; 6th UML-conference, San Francisco, LNCS 2863, Springer, 2003

[BB01] Balsters, H., de Brock, E.O.; Towards a general semantic framework for design of federated database systems; SOM Report 01A26, University of Groningen, 2001

[BBZ93] Balsters, H., de By, R.A., Zicari, R.; Sets and constraints in an object-oriented data model; Proc. 7th ECOOP, LNCS 707, Springer 1993.

[BP98] Blaha, M., Premerlani, W.; Object-oriented modeling and design for database applications; Prentice Hall, 1998

[BS98] Balsters, H., Spelt, D.; Automatic verification of transactions on an object-oriented database, 6th Int. Workshop on Database Programming Languages, LNCS 1369, Springer, 1998.

[DH99] Demuth, B., Hussmann, H.; Using UML/OCL constraints for relational database design; «UML»'99: 2nd Int. Conf., LNCS 1723, Springer, 1999

[DHL01] Demuth, B., Hussmann, H., Loecher, S.; OCL as a specification language for business rules in database applications; «UML» 2001, 4th Int. Conf., LNCS 2185, Springer, 2001

[DKM93]P. Drew, R. King, D. McLeod, M. Rusinkievicz, A. Silberschatz; Workshop on semantic heterogeneity and interoperation in multidatabase systems; SIGMOD RECORD 22, 1993

[GUW02] Garcia-Molina, H., Ullman, J.D., Widom, J.; Database systems Prentice Hall, 2002

[Hull97] Hull, R.; Managing Semantic Heterogeneity in Databases; ACM PODS'97, ACM Press 1997.

[OCL2.0] Response to the UML 2.0 OCL RfP, Revised Submission, Version 1.6, January 6, 2003

[Rei 84] Reiter, R.; Towards a logical reconstruction of relational database theory. In: Brodie, M.L., Mylopoulos, J., Schmidt, J.W.; On conceptual modeling; Springer Verlag, 1984

[ShL90] A.P. Sheth, J.A. Larson; Federated database systems for managing distributed and heterogeneous and autonomous databases; ACM Computing surveys 22,1990

[Ver97] M. Vermeer; Semantic interoperability for legacy databases. Ph.D.-thesis, University of Twente, 1997.

[Wie95] G. Wiederhold; Value-added mediation in large-scale information systems FIP Data Semantics (DS-6), 1995

[WK03] Warmer, J.B., Kleppe, A.G.; The object constraint language, 2nd ed.; Addison Wesley, 2003

REDUCING DISRUPTION IN TIME-TABLED CONDITION MONITORING

Binling Jin and Suzanne M. Embury
Department of Computer Science,
University of Manchester,
Oxford Road, Manchester M13 9PL
United Kingdom
{bjin, sembury}@cs.man.ac.uk

Abstract Condition monitoring typically has a high processing overhead and can thus disrupt the processing of business transactions by the monitored systems. Time-tabled condition monitoring aims to overcome this disadvantage by allowing the system owners to specify the times at which condition monitoring may take place, and the times when the system is too busy with revenue generating (or otherwise mission critical) processing. Unfortunately, however, when this approach is applied to a distributed system, the complex interactions between the component sites during monitoring mean that some interference with normal business processing does occur. In this paper, we present five methods for reducing this interference at the expense of the accuracy of the results of condition monitoring. We describe the outcome of an experimental evaluation that has helped us to characterise each method in terms of its effect on disruption and accuracy. We conclude by presenting heuristics to help in choosing which method to adopt in a given situation.

Keywords: Condition monitoring, distributed query evaluation, integrity monitoring.

Introduction

Organisations are becoming increasingly dependent on their information systems (ISs) in supporting many key business functions. In addition to the traditional uses of information in facilitating and guiding the day-to-day operations of the business, it is also being applied in other contexts, such as marketing, customer relationship management and strategic decision making. However, access to information comes with an associated cost. Execution of queries and transactions (especially in high volumes) can require non-negligible amounts of time. Because of this, the number and variety of these longer term applications of data that can be supported by a company's ISs are limited by the amount of processing resources that can be spared from the task of managing mission critical/revenue generating business processes.

One solution to this problem is to replicate the data in a new information system (e.g. a data warehouse [1]) but this is expensive, both in terms of the initial set-up and the continued maintenance of the replicated data. Few applications will warrant the expense of full replication, even though they may be very worthwhile and of potential value to the organisation.

However, less drastic solutions are also possible. The owner of an IS may be reluctant to allow unlimited access by a new application (since it may interfere with the rate at which revenue generating transactions can be processed, for example). However, he or she may be more willing to release some processing resources, on the proviso that control over exactly when this happens and for how long remains in his or her hands. For example, the manager of a busy IS may be very reluctant to take on additional processing burdens during office hours, but may be happy to allow a new application access to the data between the hours of 3.00am and 5.00am, once the business of the day has been concluded and the backup procedures have run to completion.

Time-tabled condition monitoring (TTCM) is a method of tracking the state of conditions expressed over a distributed information system (DIS) that attempts to make use of these less-heavily-loaded time periods. It allows the owner of each component system (or portion of data) to specify when that system is available for processing queries relating to condition monitoring. Effectively, the system owner provides a timetable (of arbitrary length) that states when the system is busy with other, more important tasks, and when it is free to undertake work for the TTCM engine. The aim is to allow the long term monitoring of conditions over distributed systems without also disrupting the normal business processing rates at the component ISs [11].[1]

The price to be paid for the resulting lack of disruption is that the record of the data items that satisfy the condition over time will not be accurate. Since the TTCM engine must wait until the component systems are free for use, some changes in conditions will not be detected until after they have occurred. If a change is short-lived, it may not be

[1] Although at present there is little demand for distributed condition monitoring in industry, we have focused on distribution as this is the more general case, with centralised systems being the special case. Moreover, with the recent rise of data sharing technologies such as the Grid, we believe that the need to monitor conditions that span several distinct systems will grow.

detected at all. At one end of this spectrum, we have the situation where the owner is so concerned about disruption to the system that he or she forbids all condition monitoring. Naturally, the levels of accuracy in this case will be very poor. Similarly, we can achieve extremely accurate results if we are not concerned about how much disruption we cause to other forms of information processing on the system in question (for example, monitoring the conditions as soon as updates occur, using an active rule mechanism).

Ideally, of course, we would like to maximise the accuracy of the results while also minimising the amount of disruption that occurs at each site. In our previous work, we have explored the use of temporal logic techniques for increasing accuracy, and have characterised them according to the degree of disruption they impose on the DIS [12]. However, the ability to trade disruption for better accuracy is only half of the story. In order to allow full control over the TTCM system, we also need to provide the system administrator with ways to reduce disruption at the expense of poorer accuracy. It is this latter form of control that we turn our attention in this paper.

The remainder of the paper is organised as follows. In Section 1, we discuss the various approaches to condition monitoring proposed in the literature and the means by which researchers have attempted to limit the processing overheads they impose. Next, in Section 2, we describe the TTCM approach in more detail and pinpoint the causes of disruption and inaccuracy within it. Section 3 presents the five different methods we have proposed for reducing disruption, and characterises them in terms of their expected effects on both disruption and accuracy. The results of an experimental evaluation of the five methods are discussed in Section 4, while Section 5 concludes.

1. Approaches to Condition Monitoring

The term *condition monitoring* refers to the continuous scrutiny of an IS in order to discover whether any data items exist within it that satisfy a particular condition. In some cases, the purpose of this scrutiny is to raise a warning when some undesirable or illegal condition has arisen (for example, monitoring of medical equipment). In other cases, the properties of data items which satisfy the condition are recorded, along with the times at which they were entered into and removed from the IS, in order to analyse long term trends (for example, in fraud detection and quality control). In this paper, we are concerned with this second kind of condition monitoring.

Experience has shown that condition monitoring can be highly expensive in terms of the processing resources it demands. Since the monitoring is *continuous,* we must re-evaluate the query that searches for matches with the condition every time a (relevant) update occurs. To make matters worse, since updates happen most frequently during periods of peak business activity, condition monitoring imposes its worst overheads at times when processing resources can least be spared.

A number of researchers have studied the problem of reducing the overheads of condition monitoring. The most common approach is to evaluate the condition query incrementally relative to each update that occurs, rather than repeating the work of checking data items that cannot possibly have been affected by the change [4, 14, 16, 19, 20]. Another common approach focuses on avoiding unnecessary recomputation of unchanged parts of the condition view by materialising it (either in whole or in part) [2, 5, 8, 17, 18].

Efficiency is an even greater concern when the condition to be monitored is distributed across several databases, because of the significant overheads imposed by the need to ship data sets between sites. Most proposals for improvements to distributed condition monitoring have therefore focussed on reducing the costs of data communications. For example, Mazumdar [13] proposed a technique for reducing the number of sites involved in the distributed query evaluation, on the grounds that this would also reduce the need for data shipping. Other authors (e.g. [10, 9, 15]) have focused on the more fundamental problem of how to keep the data sets that must be communicated between sites small and manageable.

All the methods described above assume that accuracy of condition monitoring is paramount and therefore that conditions should be checked immediately when data is updated (or transactions committed). Even taking into account the efficiency measures we have mentioned, the overhead imposed by immediate condition monitoring is too great for many system owners to feel that the benefits are worth the resultant costs. In some cases, however, we may be prepared to accept some limited reduction in accuracy, if by that means we can reduce the disruption to key business functions. A small number of authors have considered this line of argument, and have proposed a periodic approach to condition monitoring, in which condition re-evaluation is not triggered by each and every update [3, 6, 7]. Instead, the conditions may be monitored at some frequency set by the user (e.g. every hour or after every 100 transactions) or perhaps only when the user requests the current value.

These methods provide system owners with some measure of control over the resources that are given over to condition monitoring, but

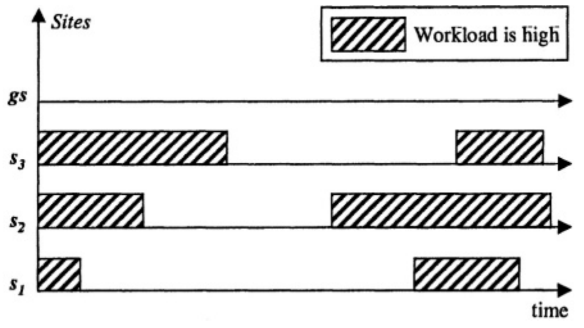


Figure 1. **An Example Set of Timetables Showing when TTCM may Operate**

only in a rather coarse-grained fashion. Moreover, they do not consider the additional problems raised by distributed condition monitoring, where each component site imposes its own constraints on resource usage. Time-tabled condition monitoring (TTCM) takes the notion of periodic monitoring a stage further, in allowing the owner of each site involved in distributed condition monitoring to specify exactly when that system can perform work on behalf of the TTCM system and for how long [11]. The TTCM system then attempts to choose an optimal plan for distributing the work of query evaluation to the component sites, in such a way that accuracy of the results will be maximised. However, because of the complex interactions between the timetables specified by the individual sites, it is not always possible to avoid all disruption at all sites. In the following section, we will present a more detailed description of the workings of TTCM and outline the causes of this unwanted disruption.

2. Time-Tabled Condition Monitoring

Figure 1 shows example timetables for a DIS consisting of three component sites (s_1, s_2 and s_3). A fourth site (gs) is also shown, representing the TTCM system itself (where the results of condition monitoring are recorded). The blank periods at each site indicate times when condition monitoring queries may be evaluated by that site, while the shaded areas indicate periods when the system is busy with other work.

When a query is submitted to the TTCM system for monitoring, it is translated into an execution plan, which describes how the work of each of the operators within the query are to be allocated across the various sites of the system. By implication, therefore, the execution plan also determines which data sets will have to be shipped between sites for condition monitoring to take place. Each site maintains an operation queue, which indicates the operations that are due for re-evaluation at that site. Operations are ordered within the queue according to decreasing length of time between their last evaluation and the most recent update to their input relations. As well as the usual relational algebra operations, operation queues may also include ship operations (which request that a local relation be transmitted to another site) and load operations (which request that data shipped from a remote site be loaded into a local relation). The final results of condition re-evaluation are sent to the TTCM site, as a set of timestamped tuples, where they are recorded for future analysis.

The reasons for the inaccuracy of the results from TTCM should now be clear. Data items which satisfy the condition will often be introduced into the IS during the periods when it is busy with normal business processing (i.e. when update rates are high). The TTCM system, however, must wait until the next free period before it can begin the work that will detect the satisfaction of the condition. Therefore, the timestamp that will be associated with the new data item will be later than the time of its actual entry into the system. Similarly, when updates to a data item mean that it no longer satisfies the monitored condition, the timestamp recorded by TTCM for its removal will also be inaccurate. It is even possible that the presence of data items that satisfy the condition may be missed altogether, if they are removed again before the TTCM system has a chance to identify them. In our previous work [12], we have shown that it is possible to reduce these inaccuracies by the inclusion of temporal reasoning in the query evaluation process, but some inaccuracy will still remain.

The causes of disruption, on the other hand, are less obvious to the casual observer. By disruption in this context we mean a reduction in the amount of non-condition monitoring work that the component sites are able to undertake while TTCM is in operation. If the timetables for TTCM have been set accurately by the system owners, and very little business processing occurs during the periods marked as "free" for TTCM, then we would expect TTCM to have no negative effect on the transaction rates exhibited by the monitored system. Yet our experiences have shown that such disruption does occur.

Why should this be so? On investigation, we discovered that the problem arises due to the complex interactions between the component systems, each of which may be operating to a different timetable for the purposes of TTCM. We can distinguish two significant forms of disruption:

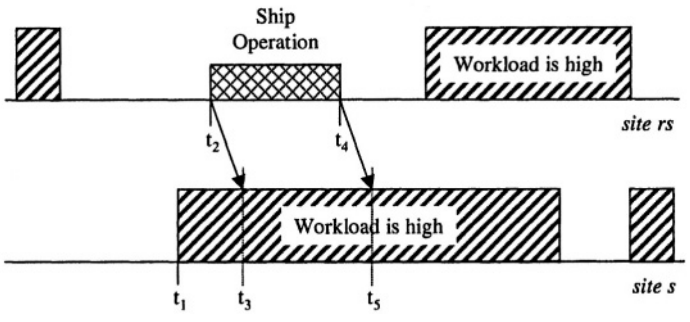


Figure 2. **Disruption Caused by Remote TTCM Processing During Busy Periods**

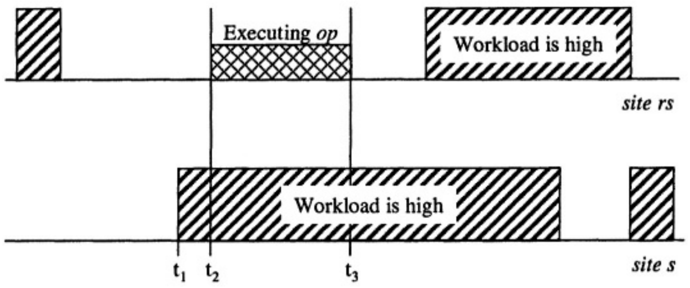


Figure 3. **Disruption Caused by Delays in Processing Requests for Remote Data**

- *Direct Disruption,* which is caused when a ship operation arrives at its destination site during a busy period for that site (Figure 2). Although the receiving site would not attempt to load the shipped data into the database (since TTCM is not enabled during busy periods), there is still some small amount of overhead involved in handling the receipt of the data and storing it locally on disk for later processing.

- *Indirect Disruption,* which is caused when a component system that is performing normal business processing (i.e. is in a busy period) needs to access data held at another site that is currently engaged in TTCM. In such a case, the processing of the transaction at the busy site will be delayed until the site which holds the data completes its current TTCM operation (Figure 3).

3. Methods for Reducing Disruption

One way to combat the disruption inherent in TTCM is to make a careful choice of the allocation of work to the various sites. However, such an approach cannot adapt to the local conditions that occur at runtime. The alternative is to modify the way in which TTCM operations are executed by each individual site, so that each site has available the complete context in which to make decisions about the best way to avoid disruption. We will now present five run-time strategies for reducing disruption in this way, and will discuss how far each of them can help to avoid disruption and what effect they each have on the accuracy of the final TTCM results.

Method 1: Ship Data Only When Both Sites are Free. The first method is aimed at reducing direct disruption. As we have seen, this form of disruption results when data is shipped to a site at which the workload is high. This suggests the simple expedient of delaying ship operations until the destination site has entered a free period and is ready to receive the data. Where this can be done, the direct disruption should be completely eliminated.

The main advantages of this approach is that it is very simple and straightforward to implement. The only information required to determine whether to ship data or not is the timetable of the destination site, which can be cached locally for easy access. The main disadvantage is the effect on accuracy, which is likely to be significant unless both sites involved in the ship operation have many free periods in their timetables. If they do not, then there is a chance that the shipment of data may have to be delayed for a very long time until the timetables of the sending and the receiving site happen to coincide.

Method 2: Ship Data via a Free Intermediate Site. The second method is a variant on the first. It aims to reduce delays in shipping data while still eliminating the direct disruption. As with the first method, method 2 begins by examining the timetables of the sending and receiving sites and calculating the next time at which both will be free. However, it also looks for a third site that can potentially be used as an intermediate holding site for the data while waiting for the destination site to become free. Once again, data is only shipped when both sites involved are free for TTCM. The difference this time is that we have two ship operations to consider: the transfer from the sending site to the intermediate site, and the transfer from the intermediate site to the destination site. If the use of an intermediate site will cause the data to arrive at the destination site sooner than with method 1, then the necessary ship operations are inserted into the operation queues at the source site and the intermediate sites.

Since delays in evaluation are the cause of inaccuracy, where there are several candidate intermediate sites, we choose the one that will result in the earliest arrival time for the data at the destination site. In a complex system, with very few free periods, it might also be worthwhile to consider the use of multiple intermediate sites, although we have not explored this option ourselves (since the search space size grows rapidly with the number of hops considered and the anticipated benefits are not all that great).

There is also the added complication that, using this method, it is possible that an older version of the relation might arrive at the destination site *after* the arrival of a newer version (which has travelled by a different route). However, this problem can be solved by shipping timestamps with each version of the relation, so that the destination site can check whether the data that has just arrived is more recent than that received previously or not.

The advantage of this method is that it should result in more accurate condition monitoring than method 1, while having much the same effect on disruption. The disadvantage is that it is slightly more complicated to implement.

Method 3: Monitor Conditions Periodically. This third method aims to reduce indirect disruption by cutting down on the number of times that condition monitoring operations are performed during TTCM. The rationale for this is that if the TTCM is issuing fewer queries then there will be more bandwidth available for handling requests for data from other (busy) sites.

Ideally, we would prevent only those executions of operations that do not result in any new information regarding the data items that satisfy the condition. For example, suppose during a free period at site s_1, the relation r is shipped twice to the destination site s_2. (This could happen if r was updated just after the first ship operation had been completed, for example.) However, at the time of the first ship, s_2 is busy and it does not process the new set of tuples for r until after the second ship operation has occurred. In this case, the first ship operation was a waste of time, because its results were not used by the TTCM system.

Unfortunately, identifying such redundant operations before they occur is impossible because we do not know when the next update to the input relations of each operation will occur. We must therefore adopt some more artificial mechanism for determining how often each oper-

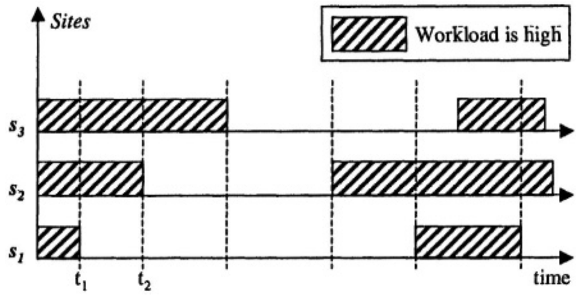


Figure 4. **Example Timetables Showing the Points at which Re-evaluation of Operations is Allowed**

ation can productively be re-evaluated. Since the majority of updates occur during busy periods, one approach is to re-evaluate each operation only when some site in the system has changed its status from busy to free. However, we also need to take into account changes that are made as a result of TTCM itself, so we should also be prepared to re-evaluate operations when a site changes from free back to busy again.

This strategy is best explained by an example. Figure 4 shows the timetables of three sites involved in condition monitoring. Initially, all the sites are busy but soon site s_1 becomes free and begins to process the operations in its queue. If an operation *op* is evaluated when s_1 becomes free at time t_1 then, under this method, we will artificially block its re-evaluation until time t_2, when one of the other sites changes its state from busy to free. The vertical lines in Figure 4 indicate the times after which operations may be re-evaluated thanks to a change in the status of some site or other.

In general, one would expect this method to have only a limited effect on disruption, since it is still possible for an operation to be evaluated many times during any one free period. However, by the same token, the negative effects on accuracy can also be expected to be limited, since the condition is still being monitored on a regular basis. Moreover, although this method was designed with the intention of reducing indirect disruption, it should also show some beneficial effects in terms of direct disruption, since the number of ship operations executed during any one free period may also be reduced.

Method 4: Monitor Conditions once per Free Period. If method 3 is expected to have only a limited effect on disruption, then an obvious way to improve it is to impose a more stringent criterion for re-evaluation that further reduces the frequency with which operations are executed. Method 4 does exactly this by ensuring that each operation is executed at most once in each free period.

The principal advantage of this method is its simplicity. However, its beneficial effects could also be significant, especially in systems which have a high update rate and/or timetables containing many long periods of free time. By the same token, we would expect to see a corresponding reduction in accuracy of condition monitoring with this method.

Method 5: Monitor Conditions During Global Free Periods. The final method aims to reduce both types of disruption, by taking the rather drastic step of only performing condition monitoring work when *all* sites are free at the same time. If our understanding of the causes of the disruptive effects of TTCM is correct then this method should eliminate both direct and indirect disruption completely. However, unless the timetables for all systems include an unusually high proportion of overlapping free time, this method is also expected to have an extremely severe effect on the accuracy of condition monitoring.

4. Experimental Evaluation

Having implemented the five methods within the TTCM system, we next undertook an experimental evaluation to determine whether our predictions as to their effectiveness were correct. In order to do this, it was necessary to define quantitative measures of the two characteristics of interest: namely, disruption and accuracy. We define the disruption of a method in terms of its effect on the total number of business transactions that are processed over the course of an experiment. We refer to the number of transactions executed during an experiment as the *TransactionNumber.*

We define the *inaccuracy* of a method to be equivalent to the total amount of time during the experiment for which the condition monitoring system has recorded a false positive or a false negative result for the condition. A false positive result occurs when the condition monitoring systems states that, for some period of time, the condition was true, when in fact it was false. Similarly, a false negative result occurs when the condition monitoring system states that the condition was false when it was in reality true. By totalling up the length of time during which an inaccurate result was recorded, we can gain some indication of the inaccuracy inherent in the condition monitoring method used.

In order to measure these quantities when the methods were in action within the TTCM system, we developed a simple order handling system, distributed over three sites. We also developed an experimental framework which allowed us to execute transactions against the database and record the actual state of the monitored condition, while using one of our methods to record the state of the condition as seen by the TTCM system for comparison. Each run of the system lasted for a period of 12 hours.

We used this experimental framework to assess the benefits and costs of each of the five methods presented above. In order to provide upper and lower baselines against which to compare our methods, we also measured the disruption and inaccuracy in two extreme cases. Lower bounds were provided by measuring the effects of running the basic TTCM evaluation method, which executes the condition monitoring operations during free periods without making any additional attempt to control disruption. For convenience, we refer to this as method 0.

The upper bounds were found by executing the sequence of business transactions with TTCM switched off (method 6). The results of this process gave us the maximum number of transactions that could be processed during the 12 hour period of the experiment. Similarly, since no condition monitoring was taking place, the record of the data items that satisfy the condition will be empty. The inaccuracy of an empty TTCM result does not tell us what the maximum possible level of inaccuracy is since, for many data items, an empty condition satisfaction record is an entirely accurate result. However, it does give a useful benchmark figure against which to compare the inaccuracy that results from the other methods, since it is almost certainly going to be worse than trying to perform some kind of TTCM, however limited.

Since the success of each individual method is dependent upon the exact form of the timetable in use at any one time, we have performed two sets of experiments with two different sets of timetables, so that we could get some initial idea of the stability of our results. The key factor in determining how hard or how easy it is to avoid disruption seems to be the ratio of:

- the average length of the free periods over all sites (*AvgLengthLFP*), to
- the average cost of condition monitoring for each site (*AvgCostCM*).

The two sets of experiments use timetables which were deliberately designed to give very different values for the ratio of these two quantities (*AvgLengthLFP/AvgCostCM* = 17 and = 2.5). Figures 5 and 6 illustrate the results in each case. In the graphs, inaccuracy is given in terms of minutes, while transaction number is a simple count of the transactions that were completed.

As we predicted, method 5 does indeed result in the least disruption, coming close to the upper bound in terms of the number of transactions

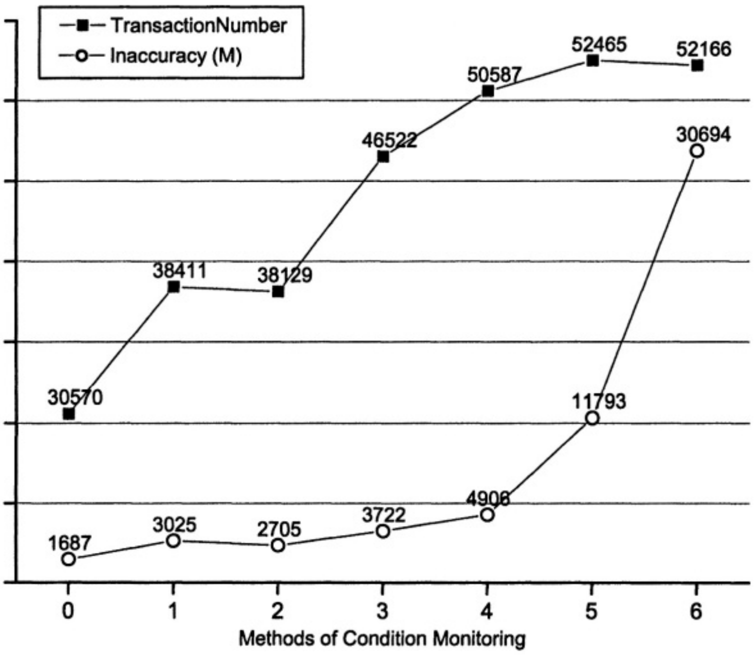


Figure 5. **Disruption and Inaccuracy Resulting from the Five Methods:** ***AvgLengthLFP/AvgCostCM*** **= 17**

that could be executed. However, this benefit comes at a heavy cost in inaccuracy, which is four to six times the inaccuracy of the basic TTCM method (method 0).

Both methods 1 and 2 have a positive effect on disruption and a surprisingly small negative effect on accuracy. On the basis of these results, there is very little to choose between them, which perhaps suggests that the extra complications of method 2 are not worth the time and trouble it takes to implement them.

Methods 3 and 4 are both extremely successful when *AvgLength-LFP/AvgCostCM* is higher; that is, when the free periods are significantly longer than the amount of time required to check the condition as a whole. The number of transactions that can be processed when using these methods is almost as high as that when no condition monitoring is taking place. In addition, there is a surprisingly small effect on inaccuracy, with the amount of false positives and false negatives being

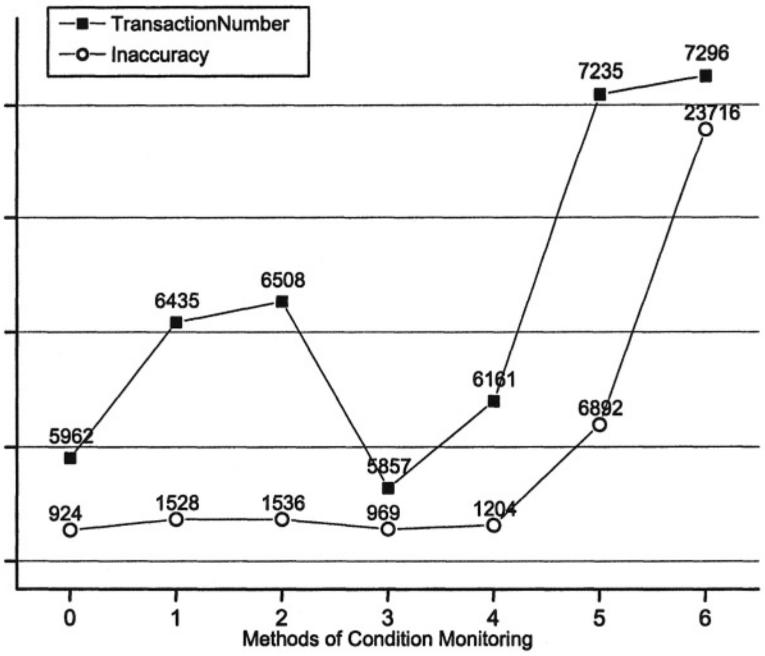


Figure 6. **Disruption and Inaccuracy Resulting from the Five Methods: *AvgLengthLFP/AvgCostCM* = 2.5**

scarcely larger than the lower baseline figure itself. However, neither method is very successful in the context of a lower value for *AvgLength-LFP/AvgCostCM.*

This is reasonable. When the free periods are only just long enough to allow the execution of one TTCM operation, then attempts to reduce the frequency of execution of those operations is unlikely to make much of a difference on the overall outcome. Effectively, in this experiment set, methods 3 and 4 were operating very much like the basic TTCM evaluation strategy; hence the limited improvement in disruption and the low levels of increased inaccuracy.

5. Conclusions

We have presented five different methods for reducing the disruption caused by time-tabled condition monitoring, and have outlined how we have evaluated their varying effectiveness using our experimental framework. None of the methods emerges as a clear winner from this evaluation. Either the positive effect on disruption is limited or else there is a substantial negative effect on the accuracy of the results. Methods 3 and 4 performed excellently in the first experiment, but were much less successful in the second. Presumably, it would be possible to construct scenarios in which each of these methods could perform well. The question as to which method to use in practice, therefore, depends in part on the particular characteristics of the DIS to which TTCM is to be applied and in part on whether lack of disruption or high accuracy is of most importance to the owners of the system.

For example, if absence of disruption is the key priority, and accuracy is considered to be of secondary importance, then method 5 is the obvious choice. It will eliminate almost all the disruption, but the results of condition monitoring will contain many phantoms and omissions. If, on the other hand, accuracy of results is a major concern, over and above disruption, then the basic TTCM method should be adopted.

More typically, however, system owners will be looking to achieve a compromise between accuracy and disruption. In such a case, perhaps the best approach is to try to determine the principal causes of disruption within the system and to choose the most appropriate method for reducing it. In addition, system owners should take into account the characteristics of their workload timetables and the conditions that are to be monitored and thus determine the value of *AvgLengthLFP/AvgCostCM* for their system. If this value is high (say, greater than 10) then there is a good chance that methods 3 and 4 may be effective in reducing disruption without too serious an effect on disruption. Otherwise, these methods should probably be avoided. Tables 1 and 2 summarise our recommendations for choosing a specific method based on system characteristics and stakeholder priorities.

TransactionNumber \ Accuracy (SuggestedMethod)	*VeryImportant*	*Important*	*NotImportant*
VeryImportant	3 or 4	3 or 4	5
Important	0	3 or 4	5
NotImportant	0	0	*

Table 1. Choice of Method when $AvgLengthLFP/AvgCostCM > 10$

TransactionNumber \ SuggestedMethod \ Accuracy	*VeryImportant*	*Important*	*NotImportant*
VeryImportant	2	2	5
Important	0	2	5
NotImportant	0	0	*

Table 2. Choice of Method when $AvgLengthLFP/AvgCostCM \leq 10$

Acknowledgments

The authors wish to thank Mark Greenwood and Ben Senior for their helpful discussions and suggestions on the work described in this paper.

References

[1] R. Barquin, and H. Edelstein. *Building, Using, and Managing the Data Warehouse.* Prentice Hall, 1997.

[2] Philip A. Bernstein, Barbara T. Blaustein, and Edmund M. Clarke. Fast maintenance of semantic integrity assertions using redundant aggregate data. In *Sixth International Conference on Very Large Data Bases, October 1-3, 1980, Montreal, Quebec, Canada, Proceedings,* pages 126–136. IEEE Computer Society, 1980.

[3] Stephanie J. Cammarata, Prasadram Ramachandra, and Darrell Shane. Extendeing a relational database with deferred referential integrity checking and intelligent joins. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989,* pages 88–97. ACM Press, 1989.

[4] Stefano Ceri and Jennifer Widom. Deriving production rules for constraint maintainance. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings,* pages 566–577. Morgan Kaufmann, 1990.

[5] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In Philip S. Yu and Arbee L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan,* pages 190–200. IEEE Computer Society, 1995.

[6] Latha S. Colby, Timothy Griffin, Leonid Libkin, Inderpal Singh Mumick, and Howard Trickey. Algorithms for deferred view maintenance. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996,* pages 469–480. ACM Press, 1996.

[7] Armin B. Cremers and G. Doman. Aim - an integrity monitor for the database system ingres. In Mario Schkolnick and Costantino Thanos, editors, *9th International Conference on Very Large Data Bases, October 31 - November 2, 1983, Florence, Italy, Proceedings,* pages 167–170. Morgan Kaufmann, 1983.

[8] Stefan Grufman, Fredrik Samson, Suzanne M. Embury, Peter M. D. Gray, and Tore Risch. Distributing semantic constraints between heterogeneous databases. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K,* pages 33–42. IEEE Computer Society, 1997.

[9] Ashish Gupta. *Partial Information Based Integrity Constraint Checking, PhD. Thesis.* Stanford University, USA, 1994.

[10] Ashish Gupta and Jennifer Widom. Local verification of global integrity constraints in distributed databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993,* pages 49–58. ACM Press, 1993.

[11] Binling Jin and Suzanne M. Embury. Non-intrusive assessment of organisational data quality. In E.M. Pierce and R. Katz-Haas, editors, *The 6th International Conference on Information Quality (IQ-2002), Cambridge, MA, USA, November 2001,* pages 398–411, 2001.

[12] Binling Jin and Suzanne M. Embury. Increasing the accuracy of time-tabled condition monitoring. In *Integrity and Internal Control in Information Systems V, IFIP TC11/WG11.5 Fifth Working Conference on Integrity and Internal Control in Information Systems (IICIS), November 11-12, 2002, Bonn, Germany,* volume 251 of *IFIP Conference Proceedings,* pages 21–35. Kluwer, 2003.

[13] Subhasish Mazumdar. Optimizing distributed integrity constraints. In Song C. Moon and Hideto Ikeda, editors, *Proceedings of the 3rd International Conference on Database Systems for Advanced Applications (DASFAA), Daejeon, Korea, April 6-8, 1993,* volume 4 of *Advanced Database Research and Development Series,* pages 327–334. World Scientific, 1993.

[14] Jean-Marie Nicolas. Logic for improving integrity checking in relational databases. *Acta Informatica,* 18(3):227–253, 1982.

[15] Xiaolei Qian. Distribution design of integrity constraints. In Larry Kerschberg, editor, *Expert Database Systems, Proceedings From the Second International Conference, Vienna, Virginia, USA, April 25-27, 1988,* pages 205–226. Benjamin Cummings, 1989.

[16] Xiaolei Qian and Gio Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Date Engineering,* 3(3):337–341, September 1991.

[17] Eric Simon and Patrick Valduriez. Design and implementation of an extendible integrity subsystem. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984,* pages 9–17. ACM Press, 1984.

[18] Divesh Srivastava, Shaul Dar, H. V. Jagadish, and Alon Y. Levy. Answering queries with aggregation using views. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings*

of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India, pages 318–329. Morgan Kaufmann, 1996.

[19] Michael Stonebraker. Implementation of integrity constraints and views by query modification. In W. Frank King, editor, *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 14-16, 1975,* pages 65–78. ACM Press, 1975.

[20] Jennifer Widom, Roberta Cochrane, and Bruce G. Lindsay. Implementing set-oriented production rules as an extension to starburst. In Guy M. Lohman, Amílcar Sernadas, and Rafael Camps, editors, *17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings,* pages 275–285. Morgan Kaufmann, 1991.

A SERVICE ORIENTED SYSTEM BASED INFORMATION FLOW MODEL FOR DAMAGE ASSESSMENT

Yanjun Zuo and Brajendra Panda
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701
Email: {yzuo,bpanda}@uark.edu
Phone: (479) 575-2067, Fax: (478) 575-5339

Abstract: The process of damage assessment and recovery in case of an information attack is time consuming. Any delay during recovery process may lead to system unavailability and is unacceptable in many time-critical applications. In this research we have developed a model to reduce this delay and aid in prompt assessment of damage, which is essential for faster recovery. In case of an attack notification, the goal of this model is to identify the affected data items quickly without having to evaluate too many data items and then make the unaffected data items available as soon as possible. This model can be used along with other intrusion detection mechanisms.

Key words: Information flow, damage assessment

1. INTRODUCTION

Damage assessment and recovery are time consuming processes. They could lead to denial-of-service in some situations and that is unacceptable. Any uncertainty in intrusion response can make the situation worse [3]. Some of the recent developments on damage assessment and recovery from information attacks are ([6], [8], [7], [10], [11], [13]). This paper focuses on achieving faster damage assessment by analyzing the patterns of information flow within an organization (or closely related organizations), which we call a domain. A very common situation in a domain is the time lag between the moment when an attack took place and the time when the attack is actually detected. During this period, any other legitimate programs or transactions may still read damaged data items and use their values to update other data items, thus, affecting latter. In database systems, damage may also spread via other means as discussed in [1] and [4]. Before the recovery process it is necessary to make accurate evaluations to determine the sets of data items that have been damaged or the sets that are clean so that clean sets of data items can be made available to user while feeding the damaged sets to recovery module. This can effectively reduce system down time after an attack. The approach presented in this paper employs a simple method to make preliminary damage assessment and can be used with other intrusion response systems.

1.1 The Data Model

A domain is an abstract entity, which is made up of multiple units. A unit, denoted by U, is a multi-service entity involved in multiple types of services in a domain. Each type of service is associated with and supported by a set of relevant data items distributed in multiple units. A unit could be a business department, a functional office, or a service center within an organization. Each of these units communicates with other units in the same domain and share information with them. We consider each unit to consist of multiple abstract entities, called service groups (SGs). For the service group in unit U, which is involved in the type of service S, we denote it as $\mathbf{SG_U(S)}$. Each SG owns and manipulates a group of relevant data items, which are involved only in that type of service. There may be a public group corresponding to a unit in the domain, which is kept in a public group area and is available for SGs of that unit. Each public group owns and manipulates public data items for the corresponding unit. That public group area in the domain keeps all public groups for all units. Figure 1 shows the structure of a domain, which consists of units $\mathbf{U_1}$, $\mathbf{U_2}$, ..., $\mathbf{U_n}$, and a public group area containing public group 1 $\mathbf{(PG_1)}$ for unit 1, public group 2 $\mathbf{(PG_2)}$ for unit 2, and so on. Figure 2 illustrates the typical structure of a unit U, which has multiple service groups involved in service $\mathbf{S_1}$, $\mathbf{S_2}$, ..., $\mathbf{S_n}$.

Data items in public groups can only be updated by authorized administrators in controlled environments. Within a domain any data item in a unit is only owned and manipulated by one service group (SG) and each service group corresponds to only one service type. Information in each public group (PG) may be released to the corresponding SGs of the unit or available for other PGs of units under administrative control. However, no information flow is allowed between any two SGs of the same unit.

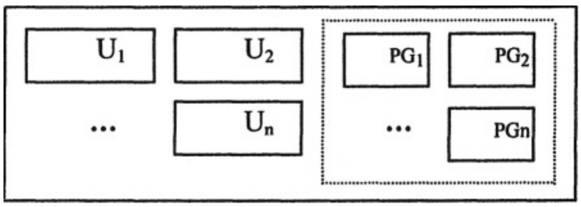


Figure 1: **The Structure of a Domain**

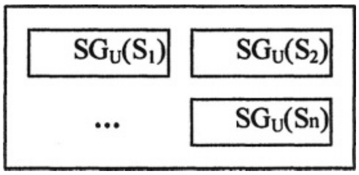


Figure 2: **The Structure of a Unit**

1.2 Separation of Service Data Items

The concepts of domain, unit and service group separate data items in a social network based on their services. Instead of grouping data items based on conflict of interests as done in the Chinese Wall Security Policy [2], our model separates data items based on relevant types of services provided in a domain. This restriction of possible information flows not only prevents manipulations of unnecessary data items but also helps perform quick damage assessment in case of an attack. The following examples would clarify the idea. Consider a domain such as a national administrative system, which consists of government bodies at federal, state, and city levels. Each government agency can be viewed as a unit. Each unit provides social services and manipulates data items related to military and civil sectors. There may be no (at least no need to allow) information flow from military sector to civil sector for each government unit. For those information needed by both sectors, a public group can be set up in the public group area within the domain. Information flow could happen from a civil service group in a state government unit to civil service groups in other state, federal or city government units, or vice versa. As another example consider a supply chain, which consists of wholesalers, retailers, customers, and vendors. Each component in this domain can be viewed as a unit. A wholesaler unit may have different service groups, each of which handles different product lines. For each product line, it has business connections with some vendors in that industry. There is no need to allow information flow between different service groups in this whole seller unit, say, from food group to baby cloth group. Common information such as financial data, or human resources data, can be retrieved from public group. The last example is about an academic body such as a university or a college, which consists of academic departments, administrative offices, and other agents. If we view each academic department as a unit, then it may have multiple abstract SGs, such as student academic group, faculty administrative group, computing facility group, and others. The department unit may keep administrative data such as faculty payroll information separated from student academic records and computer lab information from faculty research projects.

2. DAMAGE ASSESSMENT MODEL

The following definitions would be useful in explanation of the damage assessment model. The first definition has been taken from [9] with slight modification.

Definition 1: Data item *a* "can-influence" data item *b* if *a* is allowed to be used to update *b*. This permission to update is determined by the SG of the unit, which owns *a*, based on the SG's functional roles and policies or mutual agreements of relevant units. This relationship is denoted as $a \longrightarrow b$. If *b* also can-influence *a*, we denote the relationship as $a \longleftrightarrow b$.

Definition 2: $SG_{U1}(S)$ "can-influence" $SG_{U2}(S)$ if a data item of $SG_{U1}(S)$ can-influence a data item of $SG_{U2}(S)$, where U_1 and U_2 are two units within one domain and S is a type of service. This relationship is denoted as $SG_{U1}(S) \longrightarrow SG_{U2}(S)$. It must be ensured that this type of relationship only exists between SGs involved in the same type of service.

Property 1: It is obvious that $SG_{U1}(S_1)$ does not influence $SG_{U2}(S_2)$ and $SG_U(S_1)$ does not influence $SG_U(S_2)$. We denote these as $SG_{U1}(S_1) \not\longrightarrow SG_{U2}(S2)$ and $SG_U(S_1) \not\longrightarrow SG_U(S_2)$ respectively.

Definition 3: We use the representation $SG_{U1}(S) \blacklozenge\!\!\longrightarrow SG_{U2}(S)$ for actual information flow from $SG_{U1}(S)$ to $SG_{U2}(S)$. Moreover, when there is two way information flow between $SG_{U1}(S)$ and $SG_{U2}(S)$, we use the symbol $SG_{U1}(S) \longleftarrow\!\!\blacklozenge\!\!\longrightarrow SG_{U2}(S)$.

Information flow has the following characteristics:

1) Reflexive: $X \blacklozenge\!\!\longrightarrow X$;

2) Non-commutative: $X \blacklozenge\!\!\longrightarrow Y \mathrel{!\!\Rightarrow} Y \blacklozenge\!\!\longrightarrow X$;

3) Transitive: $X \blacklozenge\!\!\longrightarrow Y$ and $Y \blacklozenge\!\!\longrightarrow Z \Rightarrow X \blacklozenge\!\!\longrightarrow Z$.

While the "can-influence" relationship is a direct reflection of units' functional polices, the "information flow" relationship is the actual functional activity between two SGs. For example, if unit U_1 provides certain type of service S in a domain and allows unit U_2 to use its data items related to that service, then a "can-influence" relationship exists between the corresponding two SGs of U_1 and U_2 respectively. When unit U_2 actually updates its data items using information from unit U_1, then the information flow takes place (via SGs).

2.1 Probability Graph Analysis

In this model, we assume the attack has been detected by using some intrusion detection methods. The separation of service data items facilitates the speedup of the damage assessment process. If we can determine the service group, which a damaged data item belongs to, then any other data items belonging to different type of services can be made free and available to users.

2.1.1 Probability Graph

For those data items involved in the same type of service as the damaged one, a probability graph can be used to make quick damage assessments. We recognize the dynamic nature of a domain in a time span. Given a type of service S, each unit (or SG) may change its policies dynamically affecting the "can-influence" relationships with SGs of other units. A static period *t* for S refers to the time duration when no unit (or SG) changes its "can-influence" policies and the domain is in a static period. A time period refers to a static time period in the following sections unless stated otherwise. An evaluation time period for S is the time range from the time point when the attack started (as detected) until its effects were stopped (usually the time when the damage assessment process begins). During this period, legitimate transactions may still read dirty data.

Definition 4: A probability graph for a given type of service S in a time period t_1 is a directed graph representing can-influence relationships among SGs of different units in a domain involved in S. It is denoted as $Pt_1(S)$. A node in a probability graph denotes an SG of a unit and an edge represents can-influence relationship between the two connected nodes. An edge can be unidirectional or bi-directional. An unit has no more than one SG as a node in a probability graph for a given service type S and static time period t_1. Figures 3(a) and (b) show two probability graphs for the static time period t_1 and t_2 respectively.

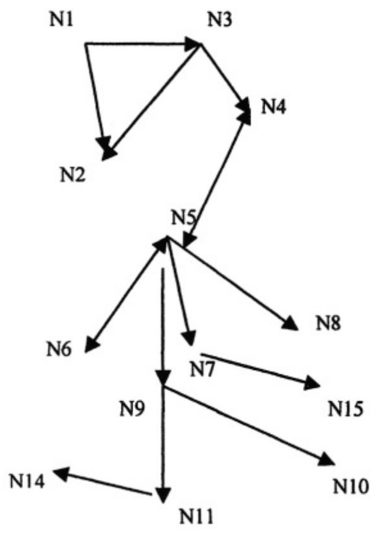


Figure 3(a): **A Probability Graph with Service S for Time Period** t_1

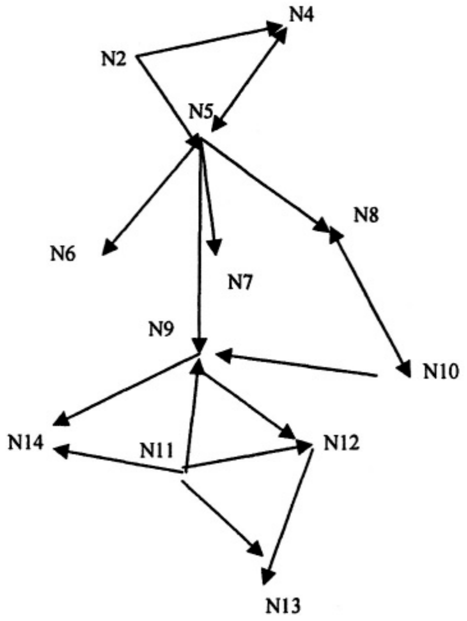


Figure 3(b): **A Probability Graph with Service S for Time Period** t_2

Definition 5: Pt(S) is a cumulative probability graph for an evaluation period *t* for a given type of service S and $Pt(S) = Pt_1(S) + Pt_2(S) + \ldots + Pt_k(S)$ based on the following rules:

1) Period $t_1, t_2, \ldots, t_k$ are continuous, $t = t_1 + t_2 + \ldots + t_k$, and *t* has the same beginning time as t_1 and the same ending time as t_k;
2) If a node $N \in$ Pm(S)'s node set, where $m = t_1, t_2, \ldots,$ or t_k, then $N \in$ Pt(S)'s node set;
3) If an edge $E \in$ Pm(S)'s edge set, where $m = t_1, t_2, \ldots,$ or t_k, then $E \in$ Pt(S)'s edge set.

A cumulative probability graph reflects any possible "can-influence" relationships among units during a cumulative time period *t* for a given service type. It is calculated based on the probability graphs in each time period (these time periods are continuous). Figure 4 represents such a cumulative probability graph for the evaluation time period *t*, where t_1 and t_2 are continuous and *t* has the same starting time as t_1 and the same ending time as t_2, which is the end time of spreading of damage.

If we can effectively reduce the evaluation time period (e.g., detect the intrusion earlier), then the cumulative probability graph may be simpler. On the other hand, if a domain is less dynamic in term of changes of "can-influence" relationships, then the needs to accumulate probability graphs could also be reduced (or eliminated).

Definition 6: If G' is a sub graph of G (where G and G' are built based on the same time period), then the graph(s) G-G' contains only the nodes in G but not in G'. Further, all edges in G except those with a vertex in G' are still in G-G'. If a node N is in G, and {N} denotes a single-node graph, G-{N} will be the result if we delete N from G.

Property 2: If we consider graph accumulation as a special operation of graph "addition", as denoted as $G = G_1 + G_2$, then $G = G_1 + G_2 \; !\Rightarrow G_1 = G - G_2$.

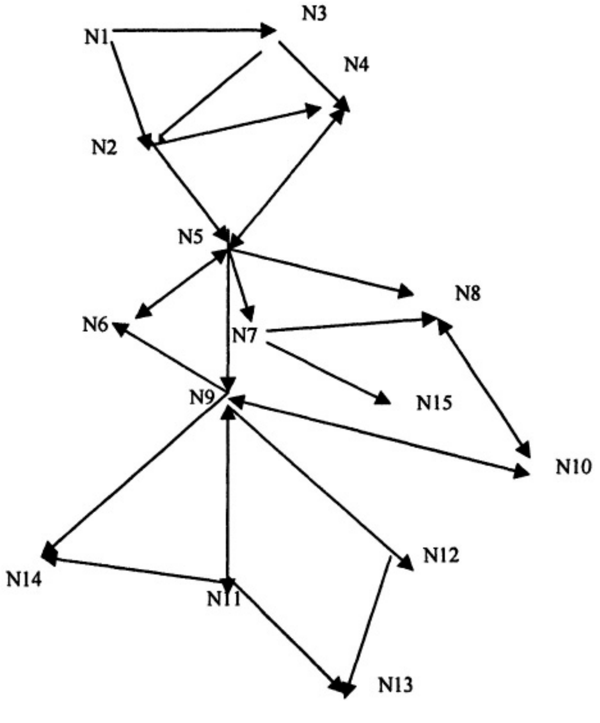


Figure 4: **A Cumulative Probability Graph with Service S for an Evaluation Time Period $t = t_1 + t_2$)**

Definition 7: A critical node in a probability graph or an information flow graph (see section 2.2) is a specific connecting point, removal of which may divide the graph into multiple connected graphs (the original graph is no longer connected). In Figure 4, nodes 5, 7, and 9 are critical nodes. A critical node in this paper is similar to an articulation point in a graph as described in [12]. Critical nodes in a graph can be found using the corresponding algorithm presented in [5]. But not every probability graph or an information flow graph has critical nodes. If there is a cycle in a graph visiting every node, then this graph has no critical node.

Definition 8: A distance of a critical node in a probability graph or an information flow graph is the number of nodes along the shortest path from the node (SG), which contains the initially detected damaged data item(s), to that critical node.

All the critical nodes for a given graph form a critical node set, denoted as CNS. We record CNS in increasing order. In Figure 4, the distance of node 5 from node1 (source of the damage) is 2, the distance of node 7 is 3, and the distance of node 9 is also 3. So CNS = {5, 7,9} or {5,9,7}.

Definition 9: A sector is a connected graph formed by nodes as obtained after the removal of a critical node from a probability graph. Several sectors may be resulted after removal of one critical node. The sector, which contains the initially known damaged data item, is called dirty sector. Figure 5 depicts two sectors produced after removal of critical node N9 from the graph shown in Figure 4. Sector 1 will be marked as a dirty sector if we identify the initially detected damaged data in N1, for example.

For any critical node, if we can determine it is free of damage, then all of the nodes it "can-influence" can be ensured not damaged. If the considered critical node cannot be eliminated, we then pick up the next critical node with smallest distance value and perform the same evaluation.

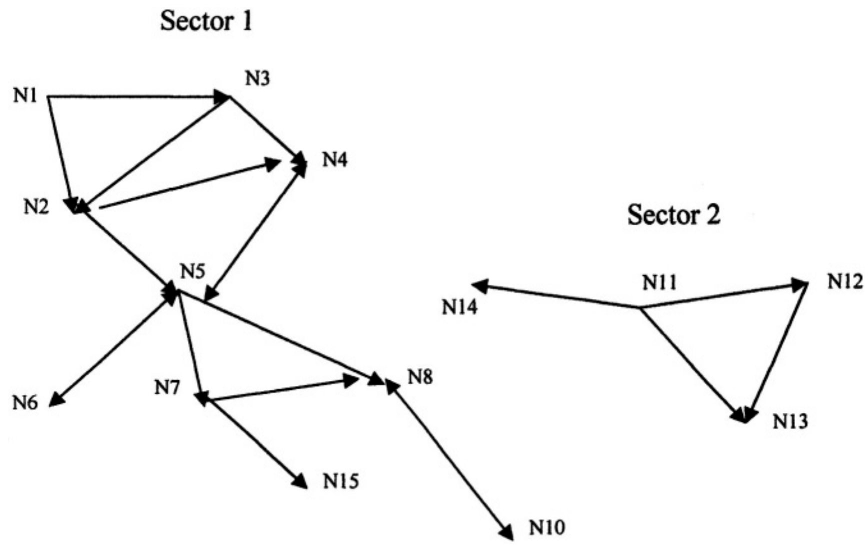


Figure 5: Two Sectors after Removal of N9 from Figure 4

2.1.2 Evaluation of a Critical Node

It is possible for a critical node to be quickly assessed to determine whether it contains damaged data items without reading all other nodes in a probability graph. This could be done by several means, such as by carefully studying the legitimate transactions and calculating the "should-be" values for data items in that critical node. Then comparisons can be made to decide the cleanness of that critical node. This paper introduces a method to determine whether any data item in a critical node has been damaged without analyzing other nodes.

2.1.2.1 Release Log and Update Log

For every critical node two logs are maintained: release log and update log. An entry in each log for a critical node has two fields, namely, SG_id and timestamp. However, these fields mean differently for the two logs. We use an example of a service group, *sg,* to clarify these concepts.

For *sg*'s update log, the first field, "SG_id", represents the ID of an SG, from which a data item was read earliest or latest by *sg*. We consider the possible earliest time since a base point when the domain has no damaged data item for the given type of service (e.g., organization startup time, or the last time when the intrusion was detected and cleaned). Before an entry is added, if there has been no more than one entry for the SG in *sg*'s update log, then the new entry is inserted. Otherwise, overwrite the most recent entry for the SG.

For *sg's* release log, the "SG_id" field represents the ID of an SG, into which a data item in *sg* flows most recently. Before an entry is added, if there is already an entry for the SG, then overwrite that entry using the most recent timestamp.

The second field, "timestamp" records the moment when the information flow takes place for each recorded entry in both release and update logs. The complete format for this field should be *year:month:date:hour:minute:second.* This paper uses the format *hour:minute* for simplicity.

Property 3: A node N*'s update log always keeps the earliest and the most recent time for incoming information flows from each other SGs (if there is only one entry for the SG, the earliest and the most recent time are the same). N*'s release log only keeps the most recent outgoing information flow from N* for each other node.

For each SG, we don't keep information flow for each individual data item. Rather we keep information for communications among SGs. An example is given in Table 1 to show the update log for node N9 based on Figure 4 and 5 as well as actual information flow records. In addition, we assume that an attack was detected as taking place at 11:00.

Table 1: Update Log for N9 in Figure 4

SG id	Timestamps
N5	7:00
N10	10:10
N10	9:00
N11	15:00
N11	6:00

2.1.2.2 Latest In-flow Timestamp

As mentioned previously, if a critical node N* in a probability graph or an information flow graph, G, is removed, several sectors are formed. In G, some nodes of a sector have incoming and outgoing edges from and/or to N*. We are interested in finding one parameter related to a sector *s* for N* as follow. Latest in-flow timestamp from *s* to N* is the largest timestamps in N*'s update log entries, which are associated with any node in *s*. This value represents the time for the most recent read operation by N* from a node in *s*. Based on Table 1 and Figures 4 and 5, L(in, N9, Sector 1) = max{7:00, 10:10, 9:00} = 10:10. Since this time is earlier than the time when the attack took place, which is 11:00, we can determine for sure that N9 is damage free. This way, preliminary assessment of a critical node is done.

2.1.2.3 Algorithm to Evaluate a Probability Graph

Based on the analysis for a critical node, there are two cases as discussed below in analyzing the probability graph (we use node N5 in Figure 4 as an example).

Case1: If we cannot evaluate any data item in N5 as damage free or we can determine at least one data item in N5 has been damaged, we will then move to the next critical node, such as N7.

Case 2: If we can determine that no data item in N5 was damaged during the evaluation time *t,* this will lead us to remove the critical node N5 (along with all its edges) resulting in two sectors as shown in Figure 5. Since sector 2 is not the dirty sector, we can determine all data items in sector 2 are clean. For sector 1, which contains the source of damaged data item, if it still contains critical nodes and one of these critical nodes has not been evaluated, then the same procedure is applied to that critical node. If there is no unevaluated critical node in the dirty sector, we stop. Next we present an algorithm that evaluates a probability graph to identify the damage.

Algorithm 1

CNS: a set of all ordered critical nodes in G based on their distance values
iNode = CNS[0]; //assigns the first node in CNS to an initial critical node to be analyzed
ESet={}; //ESet holds only evaluated critical nodes. Initially it is empty
FreeSet={}; //FreeSet holds all of the nodes which are determined free of damage
G: the cumulative probability graph for an evaluation period t
(All the above variables have global scopes for the algorithm)

```
Eval_Prob_Graph(G, iNode)
{
 1 If iNode is Null then exit    //no critical node
 2 If iNode is damage free, then
     2.1 Add iNode to ESet
     2.2 Obtain sectors $s_1, s_2, \dots s_k$ if iNode is removed from G
     2.3 For each $s_m$ in $\{s_1, s_2, \dots, s_k\}$
         2.3.1 if $s_m$ is not the dirty sector, then
             2.3.1.1 Put all nodes of $s_m$ into FreeSet
             2.3.1.2 Put all critical nodes of $s_m$ into ESet
         2.3.2 else //$s_m$ is the dirty sector
         2.3.2.1 If any node N in $s_m$ has only an incoming edge from
                 iNode, put N into FreeSet
             2.3.2.2 $s_m = s_m - \{N\}$
             2.3.2.3 $G = s_m$
             2.3.2.4 Let Nod is the critical node in $s_m$ with smallest
                     distance value
             2.3.2.5 Call Eval_Prob_Graph(G, Nod)
 3 else //iNode can not be determined damage free
     3.1 Add iNode to Eset
     3.2 Find next critical node Node in CNS but not in Eset (if no such
         node, set
             Node = Null)
     3.3 INode = Node
     3.4 Call Eval_Prob_Graph(G, iNode)
 } //end of Eval_Prob_Graph()
```

If any nodes are left with at least one critical node after the analysis of probability graph, then an information flow graph should be built and analyzed. Although probability graphs for each time period can be built well ahead of time, a cumulative probability graph for an evaluation period should be calculated whenever an attack is detected. This cumulative probability graph reflects possible information flows in this domain during the period between attacks took place and the attacking effects are stopped. Furthermore, we are only interested in part of this cumulative probability graph. When an attack took place and some data items in an SG are detected as damaged, this cumulative probability graph can eliminate any node, which has no direct or indirect information flow from the damaged SG. For example, if node A has only an outgoing edge, which is towards the damaged SG node, it can be eliminated from this cumulative probability graph.

2.2 Information Flow Graph Analysis

In this section, we analyze an information flow graph, which could be built after a probability graph.

2.2.1 Information Flow Graph

Definition 10: An information flow graph for a given type of service S in an evaluation period *t* is a directed graph representing actual information flow related to S in a domain. It is denoted as **IFt(S)**. A node in an information flow graph represents an SG and an edge represents information flow relationship between the two connected nodes. An edge can be unidirectional or bi-directional. An information flow graph is created based on actual information flow records kept by the domain. A critical node can be defined and found for an information flow graph in the same way as that for a probability graph.

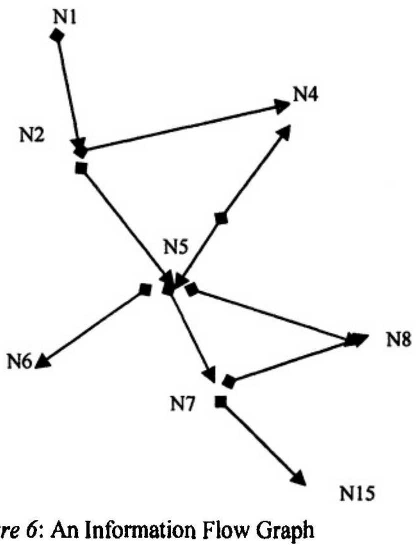


Figure 6: An Information Flow Graph

Consider the information flow graph in Figure 6 obtained by assuming that the critical node N9 in Figure 4 is determined damage free. Figure 7 shows three sectors if the critical node, N5, is removed from the graph in Figure 6.

Property 4: An information flow graph is a sub graph of the corresponding probability graph.

Property 5: Any critical node that remains in an information flow graph cannot be evaluated to be damage free.

Property 6: For a critical node N* in an information flow graph G', L(in, N*, DS) > *st,* where DS is the dirty sector after removal of N* from G', and *st* is the time when an attack was made. If this is not hold, then N* is damage free as shown in section 2.1.

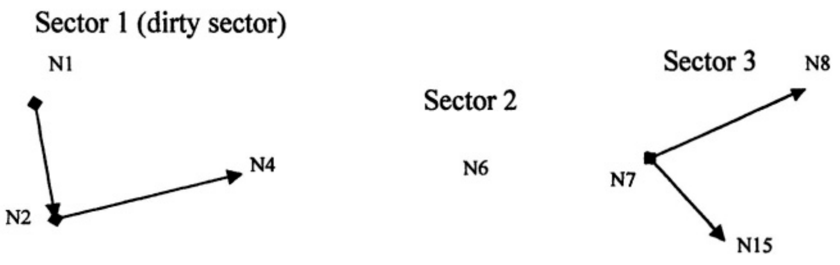


Figure 7: Three Sectors after Removal of Node N5

2.2.2 Analysis of an Information Flow Graph

Elimination of nodes based on critical nodes in an information flow graph is a complex process. An efficient analysis of the information flow and timestamp at each step is required to identify any clean nodes, if there are any.

Table 2: Release Log for N5 in Figure 6

SG id	Timestamp
N4	9:10
N7	11:15
N8	12:30
N6	10:10

Table 3: Update Log for N5 in Figure 6

SG id	Timestamps
N2	12:00
N2	13:15
N4	14:00
N4	15:30

An example of release log and update log for node N5 related to Figure 6 is given in Table 2 and Table 3 respectively.

In section 2.1, we discussed the concept "latest in-flow timestamp from *s* to N*", where *s* is a sector, which consists of multiple nodes, and N* is a critical node. We discuss other two concepts, "earliest in-flow timestamp from *s* to N* since the attack" and "latest out-flow timestamp from N* to *s*".

"Earliest in-flow timestamp from s to N* since the attack" is the smallest timestamp value (earliest time) in N*'s update log entries associated with an SG in *s*, which is later than the time when the attack occurred. It is denoted as E(in, N*, *s*). This parameter represents the earliest time when N* read a data item from an SG in *s* after the attack occurred. Related to Figures 6 and 7, E(in, N5, Sector 1) = min{12:00, 13:15, 14:00, 15:30} = 12:00, where {12:00, 13:15, 14:00, 15:30} is the set of all timestamps related to nodes in Sector 1 in Table 3 and the timestamp is later than the time of occurrence of attack (which is 11:00 as mentioned in 2.1).

Latest out-flow timestamp from N* to *s* is the largest timestamp value in N*'s release log entries, which are associated with an SG in *s*. It is denoted as L(out, N*, *s*). This parameter represents the latest time when N* wrote a data item to one SG in *s*. Related to Figures 6 and 7, L(out, N5, Sector 2) = max{10:10} = 10:10, where {10:10} is the set of all timestamps related to nodes in Sector 2 in Table 2. Similarly, L(out, N5, Sector 3) = max{1 1:15, 12:30) = 12:30, where {11:15, 12:30} is the set of all timestamps related to nodes in Sector 3 as shown in Table 2. Similar concepts can be applied to a node N relative to the critical node N*, such as E(in, N*, N) and L(out, N*, N).

There are two steps to analyze the sectors in Figure 7. The first step is to analyze a sector s as a whole. Relative to the critical node N*, if the latest read time for all nodes in *s* from N* is still earlier than the earliest write time from a node of the dirty sector to N*, we can guarantee that all nodes in *s* are clean. In Figures 6 and 7, since L(out, N5, Sector 2) = 10:10 and E(in, N5, Sector 1) = 12:00 as we calculated above, hence L(out, N5, Sector 2) < E(in, N5, Sector 1) and all nodes in Sector 2 are damage free. Hence, node N6 can be released (N6 is the only node in Sector 2). Because L(out, N5, Sector 3) > E(in, N5, Sector 1), however, we can't eliminate any node in Sector 3 at this time.

The second step follows if step 1 fails for a sector and it attempts to evaluate individual node in a sector. Any node, N, in a sector with a sole incoming edge from N* (in term of the original information flow graph), is analyzed first. If there is no such node, the analysis stops. If L(out, N*, N) < E(in, N*, DS), where DS is the dirty sector, then node N is determined free of damage. Then node N and all of its outgoing edges within that sector can be removed. Consider sector 3 in Figure 7. Since L(out, N5, N7) =11:15 found in Table 2 and E(in, N5, Sector 1) = 12:00 as calculated before, thus L(out, N5, N7) < E(in, N5, Sector1). So node N7 is damage free. Hence

N7 and its outgoing edges (from N7 to N8 and from N7 to N15) can be removed. For any other node in the considered sector, if it has only one incoming edge and that edge is from the removed node (in term of the original information flow graph), it can also be removed. Hence, node N15 in Figure 7 can be removed. However, we cannot remove N8 since N8 also has another incoming edge from N5 in addition to N7. Since L(out, N5, N8) = 12:30 found in Table 2 and E(in, N5, Sector 1) = 12:00 from Table 3, L(out, N5, N8) > E(in, N5, Sector 1). We cannot remove N8 during the information flow graph analysis. After this step, only node N8 is left in Sector 3. N8 (and its incoming edge from N5) should be forwarded for analysis by other means such as using information flow based on data items instead of SGs.

From the above example, we can deduce that for any "information flow chain",
N1'$\bullet\!\!\longrightarrow$ N2'$\bullet\!\!\longrightarrow$ ...$\bullet\!\!\longrightarrow$ Nk' in the information flow graph, if N1' can be removed, then N2', ..., Nk' can also be removed (such as N5, N7, N15 in Figure 7 as discussed above). The algorithm to analyze an information flow graph is given below.

__Algorithm 2__

```
CNS': a set of all ordered critical nodes in G' based on their distance values
iNode' = CNS'[0]; //assigns the first node in CNS' to the initial critical node to be analyzed
ESet'={}; //ESet holds the critical nodes already evaluated. Initially it is empty
FreeSet'={}; //FreeSet holds all of the nodes which are determined free of damage
G': the information flow graph for an evaluation period t
(All of the above variables have global scopes for the algorithm)

Eval_IF_Graph(G', iNode')
{
1  If iNode == Null then exit            //no critical node
   2  Add iNode into ESet'
   3  Obtain sector $s_1$, $s_2$, ..., $s_k$ if iNode is removed from G'
   4  Let $s_n$ = DS and DS is the dirty sector (1<=n<=k)
   5  For each $s_m$ in {$s_1$, $s_2$, ..., $s_k$}
      5.1  If (E(in, iNode', DS) > L (out, iNode', $s_m$))
           5.1.1 Add all nodes of $s_m$ to FreeSet'
           5.1.2 Add all critical nodes of $s_m$ to ESet'
      5.2  Else
           5.2.1  For any node N $\in$ $s_m$, where N has only one incoming
                  edge in G', which is from iNode'
```

```
            5.2.1.1   If ( E(in, iNode', DS) > L(out, iNode', N))
               5.2.1.1.1  Add N to FreeSet'
              5.2.1.1.2   If there is a chain in $s_m$ with nodes N, N1,..., Nt
                          and information only flows from N to N1, N1 to
                          N2, ... and Nt-1 to Nt
                  5.2.1.1.2.1  Add N1, N2 ..., Nt to FreeSet'
                  5.2.1.1.2.2  $s_m = s_m - \{N, N1, N2 ..., Nt\}$
                  5.2.1.1.2.3  If any element N' in (N1, N2, ..., Nt} is a
                                critical node
                    5.2.1.1.2.3.1   Add N'to ESet'
               5.2.1.1.3  else $s_m = s_m - \{N\}$
        5.3  G' = G' – $s_m$
    6     For any critical node Node in CNS' but not in Eset' (if no such
          node, set Node = NULL)
      6.1  iNnode' = Node
      6.2  Call Eval_IF_Graph(G', iNode')
  }  //end of Eval_Graph(G', iNode')
```

3. USING THE MODEL

The goal of this model is to reduce the "denial of service" and free as many data items as possible in a timely manner. The general guideline to use this model is described below.

3.1 Preconditions

(a) Probability graphs $G_1, G_2, \ldots, G_k$ for time periods $t_1, t_2, \ldots, t_k$, where $t_1, t_2, \ldots, t_k$ are continuous in time line, i.e., without any time gap between two time periods. Usually, t_k ends at the time when the spread of damage is stopped.

(b) An evaluation time period $t = t_1 + t_2 + \ldots + t_k$ and t has the same beginning time as t_1 and the same ending time as t_k.

(c) The damaged data items are identified in a node, such as N1 (SG).

(d) Each critical node (SG) keeps an update and a release log for a time period covering *t*.

3.2 The Procedure to Use this Model

(a) Identify the type of service that the damaged data item is involved in. All the data items in SGs that are involved in other types of services are put into set FreeSet1. If the damaged data item is in a PG, then multiple probability graphs might be needed to make accurate assessment.

(b) For the service type that the damaged data item is related to, build the cumulative probability graph G for the time period *t* based on $G_1, G_2, \ldots, G_k$.

$G = G_1 + G_2 + \ldots + G_k$

Call Eval_Prob_Graph() algorithm to generate FreeSet2.

(c) Nodes in G – {all nodes of FreeSet2} are used to generate an information flow graph G' based on actual information flow information.

Call Eval_IF_Graph() algorithm to generate FreeSet3. A set T = G' – {all nodes of FreeSet3} is left.

3.3 Post Conditions

Any data items in a node of FreeSet1, FreeSet2, or FreeSet3 are guaranteed to be free of damage and then can be available to users immediately. Evaluating the set T is beyond the scope of this model and other methods should be used to analyze it.

4. CONCLUSIONS

The effects of this model are largely dependent on the structure of a domain. If there are some "central points" in the probability (and/or information flow) graph and information flow tends to be in one direction, then the model has a good chance to eliminate large sets of data items, which are free of damages, at the earliest. In many cases, organizations have such structures with a central point, which acts as a critical node. Evaluating this central point often can release sets of nodes that are damage free.

This model requires that there is no information flow across the boundary of multiple service groups (SGs) of the same domain. If several SGs must share information, we can either fuse them together into a bigger SG, or put the shared data items in the public group kept in one area of that domain.

ACKNOWLEDGEMENTS

We are thankful to Dr. Robert L. Herklotz for his support, which made this work possible. This work was supported in part by US AFOSR under grant F49620-01-10346.

REFERENCES

[1] P. Amman, S. Jajodia, C. D. McCollum, and B. Blaustein, "Surviving Information Warfare Attacks on Databases", In Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 1997.

[2] D. Brewer and M. Nash, "The Chinese Wall Security Policy", In Proceedings of the 1989 IEEE Symposium on Security and Privacy, pp. 206-214, May 1989.

[3] C.A, Carver, J. M. D. Hill, and U. W. Pooch, "Limiting Uncertainty in Intrusion Response", in Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, June 2001.

[4] R. Graubart, L. Schlipper, and C. McCollum, "Defending Database Management Systems Against Information Warfare Attacks", Technical report, The MITRE Corporation, 1996.

[5] E. Horowitz, S. Sahni, S. Rajasekaran, "Computer Algorithms/C++", Computer Science Press, pp. 329-336, 1998.

[6] S. Jajodia, C. D. McCollum, and P. Amman, "Trusted Recovery", Communications of the ACM, 42(7), pp. 71-75, July 1999.

[7] P. Liu and X. Hao, "Efficient Damage Assessment and Repair in Resilient Distributed Database Systems", In Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Database and Application Security, July 2001.

[8] P. Liu, P. Ammann, and S. Jajodia, "Rewriting Histories: Recovering from Malicious Transactions", Distributed and Parallel Databases, 8(1), pp. 7-40, January 2000.

[9] B. Panda, S. Tripathy, "Data Dependency Based Logging for Defensive Information Warfare", Proceedings of the 2000 ACM Symposium on Applied Computing.

[10] B. Panda and J. Giordano, "Reconstructing the Database After Electronic Attacks", Database Security XII: Status and Prospects, S. Jajodia (editor), Kluwer Academic Publishers, 1999.

[11] P. Ragothaman, and B. Panda, "Modeling and Analyzing Transaction Logging Protocols for Effective Damage Assessment", In Proceedings of the 16th Annual IFIP WG 11.3 Working Conference on Data and Application Security, King's College, University of Cambridge, UK, July 2002.

[12] R. Sedgewick, "Algorithms", Addison Wesley, pp. 324-330.

[13] R. Sobhan and B. Panda, "Reorganization of Database Log for Information Warfare Data Recovery", In Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Database and Application Security, Niagara on the Lake, Ontario, Canada, July 15-18, 2001.

AN EFFICIENT OODB MODEL FOR ENSURING THE INTEGRITY OF USER-DEFINED CONSTRAINTS

Belal Zaqaibeh[1], Hamidah Ibrahim[2], Ali Mamat[2], and Md Nasir Sulaiman[2]
[1]*Faculty of Information Technology, Multimedia University, 63100 Malaysia,*
[2]*Faculty of Computer Science and Information Technology, University Putra Malaysia*

Abstract: In this paper, a new structural model is proposed to ease the checking of the integrity constraints in an object-oriented database system. The structure accepts declarative global specification of constraints including user-defined constraint, and an efficient representation that permits localized constraints checking. A new method called "Detection" is added to the structure to check the status of violation of the relations in an object-oriented database. The new approach is demonstrated using ALICE rule.

The notion of the constraints is used to define the connectivity between objects required for the valid expression of constraint and rule conditions. The event in Integration Rules (IRules) that defines the active behavior of an application specifies an operation to be monitored, such as modifying a data value. The semantic analysis process applies a concept known as object-centered conditions during the compilation of ALICE rules to detect semantically incorrect rules at compile time.

Keywords: Constraint, Maintenance, Object-Oriented Database, ALICE, Inter-Object Constraint.

1. INTRODUCTION

Object-oriented databases are rapidly gaining popularity, and show a promise of supplanting relational databases. It is imperative that explores the maintenance of integrity in object-oriented databases. By virtue of object orientation, some integrity constraints are represented naturally and maintained *for free* in an object-oriented database, the system type and the object class hierarchy will directly captured. Typical example of this sort is the constraint that every employee is a person and that every child of a person is a person. Other forms of integrity constraints apply to a single object, and clearly belong as part of an object class specification. An example of such a constraint for a person object is that years-of-schooling must be at least 5 years old.

As known, the integrity constraints that involve the monitoring of user updates on data items in a single object called as intra-object constraint. The following example shows the domain constraints that specify legal values in a particular domain: *Sex in* [*F, M*]. In other hand the integrity constraints that involve objects from more that one class known as inter-object constraints. For example of inter-object constraints is the association between *student* and *course* classes is expressed as each student must register in at least three courses, it will be represented as *count (student.course)* >= *3,* assuming that *course* in *student* is a data structure that holds the object identifiers of the courses taken by the particular student.

Traditionally, integrity constraints in object-oriented database systems are maintained by rolling back any transaction that produces an inconsistent state for the intra-object constraint, or disallowing or modifying operations that may produce an inconsistent state for the inter-object constraint. An alternative approach is to provide automatic repair of inconsistent states using production rules. For each constraint, a production rule is used to detect constraint violation and to initiate database operations that restore consistency. The maintenance system consists of a set of constraint services with different solving capabilities and complexity. Each service can be connected to maintain constraint relationships independently.

Object-Oriented Database Systems (OODBs) have been designed to support large complex programming projects. OODBs divide the definition and maintenance of all structures into inheritance encapsulated method, with information shared between classes kept to a minimum, since maintenance required code in multiple classes. The monitor specific state is changed in instances of the external classes. This is in contrast with relational database systems, which were designed to provide a large data repository accessible to the user through a general purpose, and declarative-style query language. Declarative query languages are well suited to handling arbitrary queries presented by an end user, but they introduce a burdensome impedance mismatch when embedded within application code. A language for the expression of Integration Rules (IRules) is an important part of an Object-Oriented Database Management System (OODBMS) that defines the active behavior of an application [7,11]. The event in an IRules specifies an operation or a situation to be monitored, such as modifying a data value.

2. PRELIMINARIES

The collection objects are the elements that allow an object to contain multiple values of a single property. The collection objects identified by the proposed standard including *set* that contains an unordered group of objects of the same type, no duplicates are allowed so this will help to reduce the number of constraints checking when an event happens, *bag* contains an unordered group of objects of the same type, duplicates are allowed, *list* is an ordered group of objects of the same type, *array* is an ordered group of objects of the same type that can be accessed by position, and *dictionary* is like an index [1, 3, 7]. Collection objects are made up of ordered keys, each of them is paired with a single value. From the collection objects a search can be performed using the keys to find the full information about any object in the database. Those keys are known as Object Identifiers (OID). OID is an internal database identifier for each individual object and this might include the page number and the offset from the beginning of the page for the file in which the object is stored [2, 3, 7].

The major types of classes used in object-oriented are control classes which manage data and have visible output, it controls the operational flow of the program [1, 2, 7]. Entity classes are used to create objects that manage data. Most object-oriented programs have at least one entity class from which many objects are created. In fact, in its simplest sense, the object-oriented data model is built from the representation of relationships between objects created from entity objects [2, 11]. Container classes existed to "contain" or manage, multiple objects that created from the same type of class [2]. Because they gather objects together, they are also known as aggregations.

Entity integrity normally enforced through the use of a primary key or unique index [2, 3]. However, it may at times be possible to have a unique primary key for a tuple (row) a relation and still have duplicate data in its' fields. This simply means that in any given relation, every tuple is unique. In a properly normalized, relational database, it is of particular importance to avoid duplicate tuples in a relation because users will expect that when a tuple is updated, there are no other tuples that contain the same data.

In the domain integrity the values in any given column fall within an accepted range [2, 3]. It is important to make sure that the data entered into a relation is not only correct, but also appropriate for the columns it is entered into. The validity of a domain may be as broad as specifying only a data type (text, numeric, etc.) or as narrow as specifying just a few available values. Referential integrity, here is the foreign key value points to valid rows in the referenced table or points to, a related tuple in another relation. It is absolutely imperative that referential integrity constraints be enforced [6, 8]. While it is possible that foreign key values may be null, it should never be invalid. If a value of foreign key is entered, it must reference a valid row in the related relation.

There are two models that deal with object-oriented database; Abstract Data Type (ADT) and Object-oriented System Model (OSM). The proposed approach is built on OSM because it has some similar features to formal syntax and semantics based on a temporal, first-order logic and it allows for multitude of expressive, and high-level view [2].

3. OUR CONSTRAINTS DOMAIN

In general there are two types of integrity constraints: static constraint and dynamic constraint. The static constraints are known and controlled by the OODBMS in addition to some dynamic constraints. Most of the dynamic constraints are complicated and unpredictable because it depends on the users needs. Figure 1 shows that the subclass *employee* inherits *classes* and *details* from *manager* and *secretary* in which both inherited from their superclass *name.*

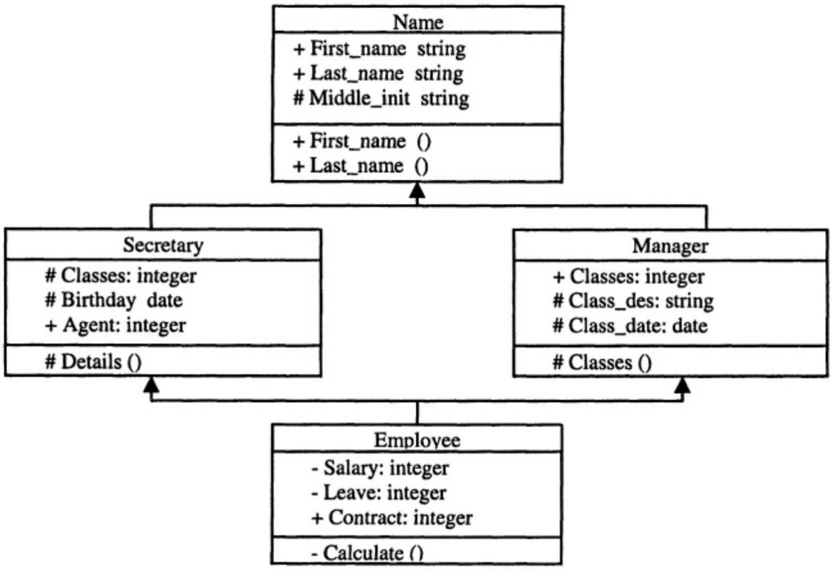


Figure1. The multi inheritance relationship

4. RELATED WORK

The most relevant researches on expression capabilities for active rule languages that are designed to maintain the database integrity. High Performance ACtive DBMS (HiPAC) is one of the first projects to extensively address active database issues and it is proposed for coupling [7, 9]. Active DBMS allows rules to be fired and executed automatically. One of the first occurrences of active capabilities was the use of ON conditions. Triggers and assertions are used to maintaining integrity constraints [5].

ARIEL rules are based on the relational data model extended with a production rule system. Rules in ARIEL are triggered based on specific events or pattern matching, as in expert systems. Rule condition testing in ARIEL is implemented by use of a variation of the Rete algorithm to improve performance [5, 7, 9].

The Object Database and Environment (ODE) represents some of the more efforts in the development of active rule languages [5]. Unlike most of the other systems, it supports a rule language within an OODB, the expression of constraints which are triggered by database updates, and the expression of rules triggered by condition monitoring [9].

STARBURST also represents one of the more developments in active rule languages [9]. Conditions in STARBURST are expressed by use of SQL with additional syntax for the expression of events, conditions, actions, and rule priorities.

Based on the active rule languages described above, a summary of the basic features associated is presented in the Table 1.

Table 1. **Comparison of constraint rules**

Features	HiPAC	ARIEL	ODE	STARBURST
*** *Temporal events***				
Absolute points	✓	✓	✓	✓
Relative points	✓		✓	✓
Periodic points	✓	✓	✓	✓
*** *Integrity of constraints***				
State constraints	✓	✓	✓	✓
Transition constraints	✓	✓		
*** *Conditions***				
Quantified variables				✓
Negated conditions	✓	✓	✓	✓
Aggregate functions	✓			✓
User-defined functions	✓	✓	✓	✓

Features	HiPAC	ARIEL	ODE	STARBURST
*** *Actions***				
Single database operations	✓	✓	✓	✓
Compound operations	✓	✓	✓	✓
User-defined operations	✓	✓	✓	
*** *Triggers***				
Event	✓	✓	✓	✓
Pattern		✓	✓	
*** *Data model***				
Relational		✓		
Extended relational				✓
Object oriented	✓		✓	

Assertion Language for Integrity Constraint Expression (ALICE) is an expression active rule language that is designed to maintain constraints over the expression of complex database.

5. OVERVIEW OF ALICE

ALICE was developed as a declarative constraint language for the expression of complex and for stating constraints in an object-oriented environment, and logic-based constraints in an object-oriented environment. As in an object model, it assumes that the existence of objects with unique object identifiers. Objects of a similar type are organized into classes, which are organized into ISA relationships; one class is a subclass of another class. If an object is an instance of a subclass, then the object must also be an instance of all of its superclasses [5,7]. The immediate subclasses of a superclass can be specified as disjoint subclasses as a way of imposing additional constraints on the classes in which an object participates.

Each class is described through the use of property definitions. Properties can be single or multi-valued. Inverse property relationships are also supported. As additional semantic detail, properties can be specified as required (no null values) and/or unique (establishing a one-to-one relationship between an object and its property values). A subclass inherits all of the property definitions of its superclasses [4, 5, 7].

ALICE provides a tool for expressing generalized, logic-based constraints against an object-oriented schema. In addition, its constraints are analyzed by a constraint explanation tool (CONTEXT) and are subsequently transformed into active database rules. The rules generated by CONTEXT provide a way to recover from constraint violations at execution time and also provide tools for translating constraints into active database rules [7, 9].

An important aspect of ALICE is representing a rule language that can be applied within an object-oriented model of data. Figure 2 shows the syntax of ALICE rule.

```
rule-name [IN rule-set-name ] [LEVEL n ] IS
 EVENT:          database-operation | user-defined-operation | time-specifier
 [CONDITION:     logical-expression| quantification]
 ACTION:                {sequence-of-operations}
END OF RULE
```

Figure 2. **ALICE rule format**

ALICE is strictly a constraint language [7, 9]. It cannot be used to directly express constraints in rule form, and does not support the expression of rules in general. It is restricted to the expression of static constraints among objects. The condition expression capabilities of ALICE are also limited because the original work on ALICE focused on the mapping of ALICE constraints to first-order logic [7, 9]. Also it does not support the expression of transition constraints or the use of external functions for complex conditions.

6. METHODOLOGY

The proposed structure exhibits some properties as it provides the capability to represent complicated relationships as it is based on a generic object system, so that some special relationships can be specified and maintained uniquely such as the relationships between domains and tasks. It can also maintain relationships between a set of distributed objects. This object model makes it possible to implement concurrent or distributed maintenance services. When the sets of objects or the constraints relationships of different types are completely independent, the constraints can be maintained by multiple processes simultaneously. This improves performance, which makes the new model constraint system easy to extend and can be integrated with any existing or specialized constraint services.

For example, assume that some information about two types of employees in a company is needed. The first type is a manager with (ID, name, classification ID, classification description, and classification date). The second type is a secretary with (ID, name, classification ID, birthday, and recruiting agent). Figure 3 shows the data definition language of the above example.

```
CREATE TYPE Name AS OBJECT (
 First_name char (15),
 Last_name char (15),
 Middle_init char (1);
 MEMBER PROCEDURE initialize);

CREATE TYPE BODY Name AS
    MEMBER PROCEDURE initialize IS
 BEGIN
             First_name := NULL;
             Last_name := NULL;
 END;

CREATE TYPE Secretary AS OBJECT (
 Classes integer,
 Birthday date,
 Agent integer,
 S_Name Name;
 MEMBER PROCEDURE initialize);

CREATE TYPE BODY Secretary AS
   MEMBER PROCEDURE initialize IS
 BEGIN
             Details := NULL;
 END;

CREATE TYPE Manager AS OBJECT (
 Classes integer,
 Class_description char (15),
 Class_date date,
 M_Name Name;
 MEMBER PROCEDURE initialize);

CREATE TABLE Employee (
 Person_ID integer,
 Recruiting_agent integer,
 Classes Manager,
 Details  Secretary,
 PRIMARY KEY (Person_ID),
 FOREIGN KEY (Classes) REFERENCES Manager);
```

Figure 3. A relation and its constraints

As shown earlier in Figure 3, the two reserved words OBJECT and BODY are used to declare the relations (*Name, Secretary,* and *Manager*). OBJECT considers as a class that contains the declaration of the attributes of the relations *Name, Secretary,* and *Manager.* BODY considers as a relevant container that contains methods and constraints. Class *Name* contains three attributes *(First_name, Last_name,* and *Middle_init)* and two methods, in our example we just assigned initial values but it could be function or procedure. Class *Secretary* contains the attributes *(Classes, Birthday, Agent, S_Name.First_name, S_Name.Last_name,* and *S_Name.Middle_init)* and a method *Details.* Class *Manager* contains the attributes *(Classes, Class_description, Class_date, M_Name.First_name, M_Name.Last_name,* and *M_Name.Middle_init).*

The previous code appears as an efficient way to declare different types of objects in OODB. The problem is not easy to detect the violation of the database and check the integrity of the data because it's very difficult to detect all constraints that appear as a result of composite inheritance. The integrity constraints in OODBSs are maintained by rolling back any transaction that produces an inconsistent state, or disallowing or modifying operations that may produce an inconsistent state. Maintaining the violation of constraints needs to know which constraint violates the database and what is the covered solution. So it is suggested that an efficient way to help in maintaining the constraints when any violation or inconvenient circumstances appears.

Our suggested model is illustrated in Figure 4, is says that to combine all related attributes together under the reserve word ATTRIBUTE, same thing with METHOD and CONSTRAINT in one class. We added another method called *Detection* to express the status of the database.

```
CLASS class_name
 ATTRIBUTE      {user defined attributes}
 METHOD         {user defined operations}
 CONSTRAINT     {user defined operations}
 Detection      {it will be hidden}
END CLASS
```

Figure 4. **The general structure of the model**

Figure 5 shows the grammar of the suggested model, notice here the reserved word *Detection* should be hidden from the user because it will be declared by the OODBMS automatically for every class when puts the class declaration. An initial value zero will be assigned to the *Detection* and the OODBMS changes its value depending on the database status.

The access of the specifier CONSTRAINT is declared the relationships and the constraints on the attributes of a class or between classes when inherit a subclass from a superclass. So the method *Detection* will contain the code that refers to the constraints status. The initial value is proposed to be zero to say that is no violation, and if an unexpected error happens then a code should be assigned to the *Detection* method.

```
        S            →  <classes>
        <classes> → CLASS  <name>  <members>
    <members>    → ATTRIBUTE <attributes> METHOD <methods>   CONSTRAINT
                      <constraints> | ATTRIBUTE <attributes> CONSTRAINT
                      <constraints> | ATTRIBUTE <attributes> METHOD  <methods> |
                      ATTRIBUTE  <attributes>
<attributes>    → <attributes>  <name>  <data type>  |  <name>  <data type>
        <methods>           → <methods> <operations> | <operations>
        <constraints>       → <constraints> <condition> | <constraints>
        <data type>         →  integer | char | date | <classes>
        <name>    → set of alphabet characters
        <operations>        → functions
        <condition>         →  special rules
```

Figure 5. The grammar for the suggested model

By using the "Detection" the status of the constraints can be checked at any time. The constraints are separated from the methods and the attributes. The constraints will be maintained or at least a message can be displayed about any error may appear as a result of data violation, then gives the programmer the choice to modify the constraints or refine the class members to avoid the violation.

7. IMPLEMENTATION

Using our proposed model the relation will be recreated as shown in Figure 6 the relation has been created to replace the code that was shown earlier in Figure 3.

```
CLASS Name
 ATTRIBUTE:
          First_name char (15),
            Last_name char (15),
            Middle_init char (1);
 METHOD:
            First_name := NULL;
            Last_name := NULL;
END CLASS;

CLASS Secretary
 ATTRIBUTE:
            Classes integer,
            Birthday date,
            Agent integer,
            S_Name Name;
 METHOD:
            Details:= NULL;
 CONSTRAINT:
            PRIMARY KEY (Classes);
END CLASS;

CLASS Manager
 ATTRIBUTE:
            Classes integer,
            Class_description char (15),
            Class_date date,
            M_Name Name;
 CONSTRAINT:
            PRIMARY KEY (Classes);
END CLASS;

CALSS Employee
 ATTRIBUTE:
            Person_ID integer,
            Recruiting_agent integer,
            Classes Manager,
            Details  Secretary;
 CONSTRAINT:
            PRIMARY KEY (Person_ID),
            FOREIGN KEY (Classes)
END CLASS;
```

Figure 6. Classes using the proposed model

Referring to the example in Figure 6, two subclasses *Manager* and *Secretary* are inherited from the superclass *Name. Employee* inherited from *Manager* and *Secretary.* The composite inheritance for attributes *M_Name* and *S_Name* from class *Name* gave them the same data type of *Name.* The *First_name, Last_name,* and *Middle_init* are added to the classes *Manager* and *Secretary* as it is inherited from same class. The constraints are inherited too and collected to be under CONSTRAINT so by grouping them the conditions that have been grouped can be checked easily. This effective especially when the user declares two constraints which conflict with each other as shown in Figure 7.

```
"All students that getting average more than 80 should be given $10"
C1: All s in student (where s.average > 80)
Implies: (all s in s.student get $10)

"All students that have average more than 90 should be given $20"
C2: All s in student (where s.average > 90)
Implies: (all s in s.student get 20$)
```

Figure 7. ALICE constraints for student schema

Suppose that C1 is a constraint over class A and C2 is a constraint over class B, and class C inherits the constraints from A and B (multiple inheritance). Constraints violate if the students grade is 95 because his grad is grater than 80 as C1 and grater than 90 as C2 (no violation between the conditions) so the student may get $10 or $20 (incorrect result) as a prize? It depends on which constraint will be enforced. A special code will be assigned to the *Detection* method. Figure 8 shows the mechanism of getting the code, assume *r1* is the rule that will be checked when inheritance happens. The involved constraints will be grouped and checked, if conflict exists then *Detection* method will be assigned with detection code and message will be displayed. The detection code is needed to modify or maintain the constraint.

```
    r1 IS
        EVENT:          when inherit any subclass
        CONDITION: conflict between constraints
        ACTION:     display a warning message
END OF RULE
```

Figure 8. ALICE rule example over inheritance

The model will be activated during the classes' creation. When inheriting class from other class, the model will work and collects the involved constraints then checks the violation wither exist or not, then informs the programmer with the detection code if violation exists by displaying a warning message. So it will ease to avoid any database crash or data corruption.

8. CONCLUSION

Typically object-oriented databases lack the capability for an ad-hoc declarative specification of maintaining the integrity constraints [3, 6]. A new model to check and maintain the violation or unexpected circumstances of the object-oriented database is proposed. The model depends on the Assertion Language for Integrity Constraint Expression (ALICE) rule language to find the suitable way to maintain the violation by designing an efficient structural model to create the relations and its constraints.

The model that has been developed to detect the constraints over the relations is presented, to ensure global declarative specification and consistency maintenance using IRules in object-oriented database environment by applying ALICE rule. Supporting integrity constraints in object-oriented database systems requires a high integration of the constraints with the rich concepts available.

With the rich semantics of object-oriented paradigm a lot of work remains to be done for future work. In particular, more optimization techniques can be developed for constraint compilation. Object-oriented databases made new challenges to semantic integrity especially to both constraint representation and constraint maintenance.

REFERENCES

[1] Ina Graham, *Object-OrientedMethods Principles & Practice*. England: Addison-Wesely, 2001.

[2] David W. Embley, *Object Database Development Concepts and Principles*. England: Addison-Wesely, 1998.

[3] Bindu R. Rao, *Object-Oriented Database Technology, Applications, and Products*. US: McGraw-Hill, 1994.

[4] Setrang Khoshafian, *Object-Oriented Databases*. New York: John Wiley & SonsInc, 1993.

[5] Urban. ALICE: An Assertion Language for Integrity Constraint Expression. Proceedings of the Thirteenth Conference on Computer Software and Applications, 1989; pp. 292-299.

[6] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. Proc. 16th Int'l Conference Very Large Data Bases, 1990; pp. 566-577.

[7] Susan D. Urbana and Anne M. Wang. The Design of a Constraint/Rule Language for an Object-Oriented Data Model. Elsevier Science Inc., J.system software 28, 1995; pp. 203-224.

[8] Urban, Karadimce, and Nannapaneni. The Implementation and Evaluation of Integrity Maintenance Rules in an Object-Oriented Database. Proceedings of the Eighth International Conference on Data Engineering, 1992; pp. 656-572.

[9] Urban. Desiderio. CONTEXT: A Constraint Explanation Tool. Data and Knowledge Engineering, North-Holtand, 1992; pp. 153-183.

[10] Michael Sipser, *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[11] Ying Jin, Amy Sundermier, and Suzanne W.Dietrich. An Execution and Transaction Model for Active, Rules-Based Component Integration Middleware. Springer-Verlag Berlin Heidelberg 2002; pp. 403-417

FROM SECURITY CULTURE TO EFFECTIVE E-SECURITY SOLUTIONS

Prof. Solange Ghernaouti-Hélie
University of Lausanne, Switzerland
Tel. 00 41 21 692 34 21
Email: sgh@hec.unil.ch
Web: http://inforge.unil.ch/sgh

Abstract: Stakes and challenges of e-security are analyzed to point out key issues in mastering information technologies risks. Lessons from the past are summarized to explain why security solutions are not effective. The global deployment of the information society is constrained by the development and overall acceptance of an international e-security framework. The validity of such model requires a challenging multidimensional approach of e-security. Several reflection axis and recommendations to guide the conceptualization of a unified e-security framework are proposed.

Key words: Information society, threats, risks management, security needs and challenges, computer crime, multidimensional approach, unified e-security framework.

1. INFORMATION AND COMMUNICATION TECHNOLOGIES REQUIEREMENTS FOR THE INFORMATION SOCIETY

1.1 Basic requirements

Information and communication technologies become a new kind of mediators for the information society and knowledge economy. These technologies must be:

- Accessible;
- Timely useable (timeliness);
- Interoperable;
- Scalable and flexible;
- Affordable;
- Open to party control;
- Trustworthy.

Doing activities with information and communication technologies suppose that three major issues have been resolved.

First, network infrastructure must exist, be accessible, available, reliable and secure. Networks must offer as much bandwidth as necessary to support user's activities.

Systems and network management approaches and solutions could contribute to achieve this issue. Moreover, the cost of use must be in correlation with the performances and quality of services obtained. That supposes a valid underlying economical model and an effective cost management process.

Second, contents and services must answer the user's needs in term of quality, integrity, confidentiality and accessibility. That could be achieved trough improving quality and security of software development, reverse engineering processing and by management. As previously, cost must be effective.

Third, a consistent international well-known regulatory framework must have been define specially to clarify the responsibility of each actors involved.

1.2 Security and trust requirements

It is not enough to promote development of connecting points to the Internet for accessibility. The information infrastructure must be reliable. Had hoc performances, services continuity and quality of services as quality of data must be guaranteed.

User's confidence in information and communication technologies will be achieved by addressing in complementary way: security and privacy protection issues.

The underlying problem lie on the level of security and trust offered and guaranteed by access, services and information and communication technologies providers. That could be sum up by the question: who controls infrastructures, accesses, uses, contents and security? Security technologies solutions and management provide part of the answer.

E-trust could be achieved through e-security and e-security would contribute to define a trusted environment. The notion of trust is central for computer security and does not rely only on technology tools. If trust is well placed, any system could be acceptably secure. If it is misplaced, the system cannot be secure. Security is a relative notion but security and trust are critical factors of success and enablers for the information society.

2. INFORMATION TECHNOLOGIES RISKS AND SECURITY THREATS

Focusing on security within open environments means to define the targets of threats. Without doing a risk analysis survey, the main targets of security threats are: end-user, network access point, network and all the infrastructures connected to the network as servers and information systems.

2.1 Increasing ICT dependency and vulnerability

Information and communication technologies are not reliable and not secure. Sources of vulnerabilities of the Internet are known. Multiple threats can occur at the environmental, physical, logical, informational and human levels.

Existing security technologies are fallible or could be circumvented, moreover it is difficult to define and support an effective security management process.

As the Internet as grown, so has connectivity, enabling also attackers to break into an increasing numbers of systems.

This is possible because more often non-secure systems are used and most systems cannot resist a determined attack if there are not well protected and monitored. Public attack tools are available. Security solutions or patches are not implemented. Management procedures and controls or system configuration and administration are defective. Human weaknesses are a reality.

A lack of an overall, global consistent and dynamic security approach and a lack of a good software development and implementation quality exist.

Opening computers and information resources through Internet, imply increasing dependency and vulnerability, so doing activities over the Internet is risky. Organizations and individuals can be hit by e-insecurity.

2.2 Cybercrime impacts

The negative impacts of security threats will affect not only systems or individuals actors but in a chained way organizations and society. Cybercrime is a reality. Usual criminals have gained new capabilities and e-delinquency could impact the economic actors. For example, economic crime can be enhanced such as: information warfare, competition distortion, internal theft, stock exchange influence, accountability unfairness, money laundering, etc. The market regulation can be weakened because traditional law enforcement is less effective, economic advantages can be given to unfair competitors, enterprise competitiveness can be reduced or canceled by unfair information access.

The growing strength of criminal organizations that caring out large scale information technology crime is alarming.

3. SOME UNSATISFIED SECURITY NEEDS

E-security fundamentals are well known: availability, confidentiality, integrity, authentication, and non-repudiation. Master technological and informational risks have to be done in allowing an efficient use of information and communication technology, and also allowing privacy in respect of fundamental human rights.

3.1 Difficulty to secure dynamic and complex environments

Information and communication technologies form complex environments. It is complex because several independents and correlated infrastructures constitute them: human infrastructure, software, data, application infrastructures, hardware infrastructure, network infrastructure, maintenance infrastructure and environmental infrastructure. Each one is dynamic and can evolve separately, has its specific vulnerabilities, security requirements and its particular security solutions.

Moreover, security solutions have they own life cycle including several stages: risks analysis, policy specification, security implementation, maintenance, evaluation and optimization.

In this context some unsolved questions are raised, with no clear answers today, among them:

- How to obtain a minimum certified security level for each infrastructure?
- How to obtain a certified global and consistent security level?

- What can be certified?
- What will be the validity of a static certificate in a dynamic environment?
- What would be the dynamic certification process able to realize and to guarantee certification in a dynamic environment?
- Who will be the certified authority (or authorities) authorized to deliver such certification at an international level?
- Who will control and manage certification processes, certification authorities and certificates?
- Who will pay for certification?
- Etc.

3.2 Lesson from the past

A wide range of stakeholders and players is present on the market such as: engineers, architects, developers, integrators, system administrators, managers, officers, lawyers, auditors, investigators, suppliers, manufactures, providers, clients and ends users.

Each one has diversity of interests, visions, solutions and languages. This reflects the evolution of the perception of handling security issues but does not lead to a better resolution of these issues.

Security is becoming more and more complex. From an historic perspective, security has been handling only through its technological dimension, and then others as managerial or legal dimensions have been taken into consideration. That is a good point, but the fact is, that they have been taken into account in an independence way instead of, a systemic and multidisciplinary approach.

More often we dispose of inefficient solutions, which introduce new weaknesses and vulnerabilities, or shift the responsibility of the security on others actors or entities, and produce a false sense of security.

Security solutions exist but are inefficient because:

- We think about tools only, not about tools, process and management;
- Tools are not enough simple and flexible;
- Tools offer a static and punctual answer to a dynamic and global problem;
- Security international standards or recommendations exist but are not implemented;
- There is no clarified share of responsibility and it is more easier to move the responsibility of the security to the end-user,
- Lack of training and competencies;
- End-users have not an e-security culture;

- Legal dispositions have been specified by people that does not fully integrated the user point of view and the technological, managerial or economical issues (mainly because it is too complex);
- No one wants to support the security cost.

3.3 E-security challenges

The real challenge is to keep simple security handling.

That means that the security responsibility must be well defined at national and international levels and that security solutions must be:

- Transparent and cost effective for the end – user;
- Cost - effective for the organization;
- Enforceable for the regulator;
- Flexible for information technologies provider.

Without offering simple and clear answers to this needs, e-security will remained an abstract concept of no use.

4. MULTI DIMENSIONAL AND GLOBAL APPROACH

Security issues and stakes are human, technological, economical, legal and political.

Security tools can't replace an ethical behavior and codes of conduct and an appropriate legal framework.

2002 OECD guidelines on the security of information systems and networks are a starting point to take into consideration security issues. But security is not only a cultural problem that has a technological dimension.

It is also a regulatory problem by the fact that technologies:

- Have become news kinds of mediators, which cannot be ignored at the individual, enterprise, organization or society levels;
- Are used to conduct criminal behaviors;
- Are the targets of criminality actions.

4.1 E-security issues from an user point of view

Security rules and tools must be usable and cost effective. That means that security mechanisms must be:

- Readily understood;
- Configured with a minimum of effort by untrained users;
- Designed with the right balance between efficiency, configurability, usability and costs.

Needs of awareness rising about risks assessments and risks management Needs of culture and education are real.

4.2 E-security issues from a managerial point of view

Information and communication services must be based on the use of secure systems, certified products and services.

That means to enforce use of standardized and certified solutions. ISO 15408 (Common Criteria) seems to be the best way to strengthen the fairness of the security market and that effective security offers exist.

Information technologies managers have to:

- Consider security as a permanent process that take into consideration resources, costs, and processes optimization within a risk management framework;
- Define specifics security policies to support business activities (security reference model) and a crisis management policy (back-up solutions),
- Configure and manage hardware and software securely;
- Be aware of they own penal responsibility in cases of major security incidents or crisis;
- Develop information assurance and legal conformance;
- Manage human resources (check personal background, define responsibility).

4.3 E-security issues from a technology point of view

To answer the need of monitoring and reaction, auditing mechanisms should be designed into critical systems. These mechanisms may report violations of a defined policy or actions that are considered to be security threats. The use of strong identification and authentication solutions, operational cryptographic mechanisms and one-time password are hardly recommended. Automated or semiautomatic techniques for guiding the selection of mechanisms for enforcing security policies and rules previously defined have to be designed. When necessary, certified and recognized third party authorities that have a regional, national and international recognition could be used. That means that such authorities exist with defined collaborative rules between them.

It is not necessary to define more security standards, but to promote certification processes by public institutions, based on the International Standard ISO 15408 - Common Criteria. That seems to be the best way to strengthen the fairness of the security market. Certification processes should be adapted to be more accurate for dynamic environment like Internet.

Certified e-security solutions must be supported in a native node by information technologies.

Security can be improved by:

- Monitoring vulnerabilities and security solutions;
- Finding computer and network security flaws;
- Avoiding single points of failure;
- Having an adaptability defensive mode.

4.4 E-security issues from a legal point of view

Law, legal institutions must exist to dissuade criminal behaviors and to pursue people who act in illegal ways. Security solutions can protect a given environment in a particular context, but cannot prevent criminal behavior.

More often, computer and cyber crime are poorly pursued because:

- They can be automated, software embedded, and remotely realized;
- The transnational dimension of that kind of crime requires an international and cooperative judicial system;
- Criminals can also use someone else identity making their identification difficult;
- It is difficult to qualify the facts;
- Crime and evidence are related to immaterial resources.

A regulatory framework must be enforceable and effective both at the national level and at the international level. It had to be defined and supported by governments.

4.5 E-security from a market point of view

Technologies must have reduced vulnerabilities and improved quality and security code. The market must increase product liability, take into consideration the mobile world and must enforce authentication and privacy.

Only a parallel development of security control and privacy protection will allow confidence into information and communication technologies and in e-activities or e-transactions.

Market forces do not drive sufficient investment for:

- Users and contents identification and authentication;
- Watermarking and fingerprinting;
- Digital signature;
- Public Key Infrastructure;
- Tracking;
- Confidentiality and Privacy management;
- E-transaction payment;
- User interface.

5. AREAS FOR IMPROVEMENT OF E-SECURITY

There is no real technical obstacle to further development of e-security but the scope of deployment of effective local and international e-security services is very complex and the technical and management costs are not trivial.

Private and public partnership is desirable on a National, European and International levels to integrate security into infrastructures and to promote security culture, behavior and tools.

Business, financial and organizational models are to be found to support effective deployment of security that could benefit to each one.

It is fundamental that the international community:

- Propose an unified e-security framework which take into consideration, in a complementary way the human, the regulatory, the organizational and economical, the technical and operational dimensions of e-security;
- Promote an e-security culture (information on stakes and risks, diffusion of simple recommendations as for example: use secure systems, reduce vulnerability in avoiding dangerous situations or behaviors, etc.);
- Train and inform on security, privacy or data protection issues, existing solutions, legal dispositions, etc.;
- Train and inform on information and communication technologies;
- Force information technologies and contents providers to improve security of their products and services;
- Products or services must integrate in native simple and flexible security measures and mechanisms; they must be well documented and comprehensible (security mechanisms must be readily understood and configured easily by untrained users);
- Security must not be considered anymore as an option. As we trust in air transportation (in these contexts security is not an option for fortunate passengers), we must have confidence into information technologies;
- Techniques must be define to guide the selection of mechanisms that enforce a security policy;
- In integrating at the beginning of their products development life cycle security processes, measures and solutions.

6. CONCLUSION

It is our responsibility to promote a safe and reliable cyberspace environment to contribute to design the emerging information society. A minimum level of security for information and communication technologies must be provided with an affordable cost. Security must not become an exclusion factor for everyone that would like to conduct private or business activities over the Internet.

Efficient e-security will result of a balance between security needs, financial and human processes, viable technological and legal solutions, to be put in operation to satisfy e-security needs.

Security is a compromise between cost, security service level and time to deliver them. It is illusive to believe that these three factors could be satisfied together; choices have to be made between cost, level of security and time to deliver security. After a one-privileged criterion has been chosen, the others have to be adapted.

ACKNOWLEDGEMENTS

The author acknowledges Professor Stefano Spaccapietra to have given her the opportunity to share security global issues with people who are involved in the technological dimension of security.

REFERENCES

1. M. Bishop. Computer security; Art and Science. Addison Wesley 2002.
2. S. Ghernaouti-Hélie. Stratégie et ingénierie de la sécurité des réseaux. InterEditions - Dunod 1998.
3. S. Ghernaouti-Hélie. Internet et sécurité. Que-sais-je? n°3609. PUF 2002.
4. S. Ghernaouti-Hélie. Challenges to develop and deploy a unified e-security framework. UNECE Workshop on E-Security and Knowledge Economy, 12 February 2003, Geneva, Switzerland.
5. International standards ISO 15408, ISO 13335.

CONSISTENT QUERY ANSWERING

Recent Developments and Future Directions

Jan Chomicki
Department of Computer Science and Engineering
University at Buffalo, SUNY
Buffalo, NY 12260-2000
USA
chomicki@cse.buffalo.edu

Abstract We summarize here recent research on obtaining consistent information from inconsistent databases. We describe the underlying semantic model and a number of approaches to computing consistent query answers. We conclude by outlining further research directions in this area.

Keywords: Databases, integrity constraints, queries, inconsistency.

Background

We are concerned here with relational databases and knowledgebases. A *database* consists thus of *facts* and *integrity constraints.* A *knowledgebase* can additionally contain *rules* (not necessarily Horn).

A knowledgebase is inconsistent if it implies, in the classical sense, every formula, leading to the trivialization of reasoning. An inconsistent knowledgebase has no model and thus can hardly be viewed as a representation of the real world. However, inconsistency is a common phenomenon in knowledgebases and databases. This apparent paradox can be explained by observing that the stored information is not necessarily *correct* and *complete.*

Inconsistency is often viewed as a *defect,* to be avoided at all costs. Knowledgebases are assumed to be consistent, with their consistency preserved by updates. Moreover, different forms of nonmonotonic reasoning or truth maintenance make sure that inconsistencies do not occur during reasoning.

EXAMPLE 1 *The* Closed World Assumption (CWA) *[49] is a form of nonmonotonic reasoning that infers $\neg L$ for every atom L not implied by the knowledgebase. CWA preserves the consistency of the knowledgebase if the latter contains only atomic facts and Horn rules. In the presence of disjunction, however, it is well known that CWA can lead to inconsistencies. Consider the knowledgebase $p \vee q$. It does not imply p or q separately, and therefore CWA derives $\neg p$ and $\neg q$. But $\{p \vee q, \neg p, \neg q\}$ is an inconsistent set of formulas. Consequently, weaker versions of CWA were studied [45].*

In relational databases, inconsistencies appear as violations of integrity constraints by the current database instance (a set of facts). This can be viewed as an instance of the more general logical notion of inconsistency, under the Closed World Assumption. Database systems typically *prevent* such violations by cancelling the offending updates to the database. Another possibility is to revise the database by *resolving* inconsistencies.

EXAMPLE 2 *Assume a database contains a fact p and an integrity constraint $\neg p \vee \neg q$. Inserting q leads to an integrity violation. A typical DBMS would reject the insertion. Another option is to replace p by $p \vee q$ but that requires moving to a richer representational framework in the form of* disjunctive databases [53], *which is beyond the capabilities of real DBMS today.*

However, present-day database applications have to consider a variety of scenarios in which data is not necessarily consistent. Integrity violations may be due to the presence of multiple autonomous data sources. The sources may separately satisfy the constraints, but when they are integrated the constraints may not hold. Moreover, because the sources are autonomous, the violations cannot be simply fixed by removing the data involved in the violations. Integrity constraints may also fail to be enforced for efficiency (e.g., denormalization) or other reasons. Finally, it may often be the case that the consistency of a database is only temporarily violated and further updates or transactions are expected to restore it. In all such cases, traditional approaches to deal with integrity violations fail, and a new approach is required.

Consistent query answers

Bry's approach

Bry [13]was the first to note that the standard notion of query answer needs to be modified in the context of inconsistent databases[1]. He proposed the notion of a *consistent query answer* – a query answer that is unaffected by integrity violations present in the database. The set of such answers forms a conservative estimate of the reliable information content of a database. Bry's proposal does not require that the database be modified to remove the inconsistencies and thus is suitable to the scenarios, outlined above, in which the removal of inconsistencies is impossible or undesirable.

Bry's definition of consistent query answer is based on provability in minimal logic and expresses the intuition that the part of the database instance involved in an integrity violation should not be involved in the derivation of consistent query answers. This is not quite satisfactory, as one would like to have a semantic, model-theoretic notion of consistent query answer that parallels that of the standard notion of query answer in relational databases. Moreover, the data involved in an integrity violation is not entirely useless and some reliable partial information can be extracted from it.

EXAMPLE 3 *Assume that an instance of the relation* Student *is as follows:*

Name	*Address*
Smith	*Los Angeles*
Smith	*New York*

Assume also that functional dependency $Name \rightarrow Address$ *is given. In Bry's approach, no positive information can be derived from this inconsistent database, although such information is clearly present, for example, we know that there is a student named Smith, and that Smith lives in Los Angeles or New York.*

The crucial observation to address the shortcomings of Bry's approach is as follows: even in the above simple example there is more than one minimal way to restore the consistency of the database. Therefore, *all such ways* should be considered. This brings us to the realm of *belief revision* [29] where minimal change has been extensively studied. Choosing *model-theoretic* revision operators can also yield the desired semantic definition of consistent query answers. These insights form the basis of the approach of Arenas, Bertossi and Chomicki [2], discussed below. The paper [2] provided a framework on which most of the subsequent work in this area is based. We discuss it in detail next.

Repairs and query answers

We assume a fixed relational database schema and a set of integrity constraints *IC* over this schema. We say that a database instance r is *consistent* if $r \models IC$ in the standard model-theoretic sense (i.e., *IC* is true in r), and *inconsistent* otherwise.

DEFINITION 4 *Given a database instance r, we denote by $\Sigma(r)$ the set of formulas $\{P(\bar{a}) | r \models P(\bar{a})\}$, where the P is a relation name and $\bar{a}$ a ground tuple. $\Sigma(r)$ is a set of facts corresponding to the instance r. The* distance *$\Delta(r, r')$ between instances r and r' is the symmetric difference:*

$$\Delta(r, r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r)).$$

For the instances r, r', r'', $r' \leq_r r''$ if $\Delta(r, r') \subseteq \Delta(r, r'')$, i.e., if the distance between r and r' is less than or equal to the distance between r and r''.

DEFINITION 5 *Given database instances r and r', we say that r' is a* repair *of r if r' is consistent ($r' \models IC$) and r' is $\leq_r$-minimal in the class of consistent database instances.*

EXAMPLE 6 *Suppose the results of an election in which two candidates, Brown and Green are running, are kept in two relations:* BrownVotes *and* GreenVotes.

BrownVotes

County	*Date*	*Tally*
A	*11/07*	*541*
A	*11/11*	*560*
B	*11/07*	*302*

GreenVotes

County	*Date*	*Tally*
A	*11/07*	*653*
A	*11/11*	*780*
B	*11/07*	*101*

Vote tallies in every county should be unique: thus the functional dependency $County \rightarrow Tally$ should hold in both relations. On the other hand, we may want to keep multiple tallies corresponding to different counts (and recounts). Clearly, both relations will have two repairs, depending on whether the first or the second count for county A is picked. So overall there are 4 repairs of the entire database.

To define consistent answers to queries defined in some query language, we assume that this language has already a well-defined notion of query answer in a database instance. For example, a k-tuple $(a_1, \ldots, a_k)$ is an answer to a relational calculus query $\phi(x_1, \ldots, x_k)$ in an instance r if $r \models \phi(a_1, \ldots, a_k)$. SQL2 has also a well-defined notion of when a tuple is a query answer. We define by $ans_Q(r)$ the set of all answers to a query Q in an instance r.

DEFINITION 7 [2] *Assume the set of all answers to a query Q is a k-ary relation. A k-tuple $\bar{t}$ is a* consistent query answer *to Q in an instance r (in symbols: $\bar{t} \in cns_Q(r)$) if for every repair r' of r, $\bar{t} \in ans_Q(r')$.*

EXAMPLE 8 *Returning to Example 6, we can see that the only consistent answer to the query:*

```
SELECT *
FROM BrownVotes
```

is the tuple

B	*11/07*	*302*

since the remaining two tuples are in conflict and will not appear in every repair.

Similarly, the only consistent answer to the query

```
SELECT County
FROM BrownVotes
WHERE Tally > 400
```

is A. Note also that the latter answer is not obtained if the conflicting tuples are blocked in the derivation of consistent answers (as in Bry's approach).

From the belief revision point of view, the problem addressed here is that of *revising* the database with the integrity constraints. The definition of repair (Definition 5) follows Winslett's semantics [57]. The notion of consistent query answer (Definition 7) corresponds to the notion of *counterfactual inference* in that semantics. We note, however, that research in belief revision has addressed mostly revising arbitrary prepositional theories with prepositional formulas and focused on the semantics of revised theories. On the other hand, the relational database context requires a first-order approach but assumes a restricted form of the revised theory which is just a set of facts. The latter restriction makes possible an efficient derivation of consistent query answers even for large databases.

Note. A *literal* is of the form $P(x_1, \ldots, x_k)$ (a positive literal) or $\neg P(x_1, \ldots, x_k)$ (a negative literal), where *P* is a database relation. We will denote by L_i literals and by ϕ a quantifier-free formula containing only built-in predicates. We consider the following classes of integrity constraints:

- *universal* constraints: $\forall\ (L_1 \vee \cdots \vee L_n \vee \phi)$ (*binary* if $n = 2$);
- *denial* constraints: universal constraints with only negative literals;
- *inclusion* dependencies: $\forall\ (L_1 \vee \exists\ L_2)$, where L_1 is a negative and L_2 a positive literal.

Functional dependencies (FDs) are a special case of denial constraints.

EXAMPLE 9 *Consider a relation Student with two attributes Name and Address. The (key) functional dependency $Name \rightarrow Address$ can be written as the following denial constraint*;

$$(\forall x)(\forall y)(\forall z)(\neg Student(x,y) \vee \neg Student(x,z) \vee y = z).$$

Computing consistent query answers

We discuss here a number of different mechanisms for computing consistent query answers. Note that Definition 7 suggests that consistent query answers can be computed by evaluating the given query in every repair of the given database. However, this approach is not practical because there may be exponentially many repairs even in very simple cases.

EXAMPLE 10 *Consider the functional dependency $A \rightarrow B$ over R and the following family of instances of R, each of which has $2n$ tuples (represented as columns):*

A	a_1	a_1	a_2	a_2	$\cdots$	a_n	a_n
B	b_0	b_1	b_0	b_1	$\cdots$	b_0	b_1

Each instance has 2^n repairs.

Query rewriting

Query rewriting is based on the following idea: *Given a query Q and a set of integrity constraints, construct a query Q' such that for every database instance r the set of answers to Q' in r is equal to the set of consistent answers to Q in r.*

Query rewriting was first proposed in [2] in the context of the domain relational calculus. The approach presented there was based on concepts from semantic query optimization [15], in particular the notion of a *residue*.

Residues are associated with literals of the form $P(\bar{x})$ or $\neg P(\bar{x})$ (where $\bar{x}$ is a vector of different variables of appropriate arity). For each literal $P(\bar{x})$ and each constraint containing $\neg P(\bar{x})$ in its clausal form (possibly after variable renaming), a local residue is obtained by removing $\neg P(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint. For each literal $\neg P(\bar{x})$ and each constraint containing $P(\bar{x})$ in its clausal form (possibly after variable renaming), a local residue is obtained by removing $P(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint. Finally, for each literal the global residue is computed as the conjunction of all local residues (possibly after normalizing variables).

EXAMPLE 11 *The functional dependency*

$$(\forall x)(\forall y)(\forall z)(\neg Student(x,y) \lor \neg Student(x,z) \lor y = z)$$

produces for $Student(x,y)$ *the following local and global residue*

$$(\forall z)(\neg Student(x,z) \lor y = z)$$

The rewritten query is obtained in several steps. First, for every literal, an expanded version is constructed as the conjunction of this literal and its global residue. Second, the expansion step is iterated by replacing the literals in the residue by their expanded versions, until no changes occur. Finally, the literals in the query are replaced by their final expanded versions.

EXAMPLE 12 *Under the functional dependency*

$$(\forall x)(\forall y)(\forall z)(\neg Student(x,y) \lor \neg Student(x,z) \lor y = z)$$

the query $Student(x,y)$ *is rewritten into*

$$Student(x,y) \land (\forall z)(\neg Student(x,z) \lor y = z).$$

In this case, the expansion step is iterated only once.

The above approach is applicable to binary integrity constraints and queries that are conjunctions of literals [2]. For more general queries it is incomplete: The rewritten query does not necessarily produce all consistent answers. For some non-binary constraints, the rewriting may fail to terminate.

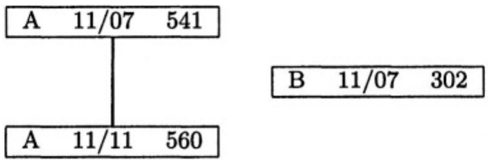


Figure 1. Conflict graph

Clearly, the query-rewriting approach is applicable to SQL queries corresponding to the above class of relational calculus queries.

EXAMPLE 13 *In the database of Example 6, the query*

```
SELECT *
FROM BrownVotes
```

is transformed to:

```
SELECT *
FROM BrownVotes B1
WHERE NOT EXISTS
  SELECT *
  FROM BrownVotes B2
  WHERE B1.County = B2.County
    AND B1.Tally <> B2.Tally.
```

This opens the possibility of using SQL engines for computing consistent query answers. In that way, very large databases can be handled.

Conflict graphs

Although the set of repairs of a database instance may be of exponential size, it can often be compactly represented. For functional dependencies, it is very natural to define the *conflict graph* of the given instance, whose vertices are the tuples in the instance and the edges represent conflicts between tuples. Repairs correspond to maximal independent sets in the conflict graph.

EXAMPLE 14 *The conflict graph of the instance of the relation BrownVotes from Example 6 is represented in Figure 1.*

We show here a nondeterministic algorithm [17, 16] for checking whether *true* is a consistent answer to a ground query Φ in an instance r of a relation *P*. We assume that the sentence Φ is in CNF, i.e. of the form $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \Phi_l$, where each Φ_i is a disjunction of ground literals. Φ is true in every repair of r if and only if each of the clauses Φ_i is true in every repair. So it is enough to provide an algorithm that will check if for a given ground clause *true* is a consistent answer.

It is easier to think that we are checking if *true* is **not** a consistent answer. This means that we are checking whether there exists a repair in which $\neg\Phi_i$ is true for some i. But $\neg\Phi_i$ is of the form

$$P(\bar{t}_1) \wedge P(\bar{t}_2) \wedge \ldots \wedge P(\bar{t}_m) \wedge \neg P(\bar{t}_{m+1}) \wedge \ldots \wedge \neg P(\bar{t}_n),$$

where the $\bar{t}_i$'s are tuples of constants. (We assume that all facts in the set $\{P(\bar{t}_1), \ldots, P(\bar{t}_n)\}$ are mutually distinct.)

The nonderministic algorithm selects an edge E_j in the conflict graph forevery j, such that $m+1 \leq j \leq n$, $\bar{t}_j \in r$, and $\bar{t}_j \in E_j$, and constructs a set of tuples S, such that

$$S = \{\bar{t}_1, \ldots, \bar{t}_m\} \cup \bigcup_{m+1 \leq j \leq n, \bar{t}_j \in r} (E_j - \{\bar{t}_j\})$$

and *there is no edge* $E \in \mathcal{G}_{F,r}$ *such that* $E \subseteq S$. If the construction of S succeeds, then a repair in which $\neg\Phi_i$ is true can be built by adding to S new tuples from r until the set is maximal independent.

The algorithm is also applicable to denial constraints that generalize functional dependencies. In this case the notion of conflict graph is replaced by that of conflict hypergraph. The algorithm can be extended to deal with nonground queries, too [18]. However, it is still not applicable to queries with quantifiers or general universal constraints.

Typically, the number of conflicts in a database is not large, and thus the conflict graph does not require much space and fits in main memory. In such a case, the above approach is practical even for large databases.

Logic programs

Another approach to computing consistent query answers relies on logical specification of repairs [3, 5]. A logic program (with disjunction and classical negation) Π_1 is constructed on the basis of the given integrity constraints and database instance. Repairs correspond to *answer sets* [30] of this program. A query is also represented using a logic program Π_2 with a distinguished query predicate. The query atoms that belong to every answer set of the program $\Pi_1 \cup \Pi_2$ provide consistent query answers. We demonstrate this construction through an example.

EXAMPLE 15 *Consider Example 3 In this case, the logic program* Π_1 *contains the facts*

$$Student('\text{Smith}', '\text{Los Angeles}').$$
$$Student('\text{Smith}', '\text{New York}').$$

as well as the following rules:

$$\neg Student'(x,y) \lor \neg Student'(x,z) \leftarrow Student(x,y), Student(x,z), y \neq z.$$
$$Student'(x,y) \leftarrow Student(x,y), not\ \neg Student'(x,y).$$
$$\neg Student'(x,y) \leftarrow not\ Student(x,y), not\ Student'(x,y).$$

Here Student' refers to the repaired version of Student. The first rule is responsible for repairing integrity violations. The second and third rules guarantee persistence. The first rule should override the remaining ones. This can be accomplished, for example, using logic programs with exceptions [38]. The first rule will then be a higher-priority exception, and the remaining rules – lower-priority defaults. A logic program with exceptions can be converted into a logic program with disjunction and classical negation [38]. Consider now any relational calculus query Q. Such a query can be converted to a stratified logic program in a standard way.

The approach outlined above is very general as it can handle arbitrary relational calculus queries and binary universal constraints. It has been extended to handle inclusion dependencies under the assumption that null values are allowed in repairs (which slightly changes the semantics of consistent query answers). A similar approach was independently proposed by Greco et al. [34, 33]. That approach can handle arbitrary universal constraints. In [8], another encoding of repairs by logic programs was proposed. That encoding uses additional predicate arguments to represent *annotations,* along the lines of [7]. The resulting program is somewhat simpler because it does not require classical negation (but still uses disjunction). In [54], specification of repairs using ordered logic programs is proposed. In that formulation, disjunctions and multiple versions of the same predicate are not necessary.

Answer sets of logic programs with disjunction and classical negation can be computed using any of the number of systems proposed in the logic programming community: DLV [22] or `smodels` [51]. These implementations can handle only relatively small databases because they work by grounding a logic program and use only main memory.

The paper [23] proposes several optimizations that are applicable to logic programming approaches. One is localization of conflict resolution, another - encoding tuple membership in individual repairs using bitvectors, which makes possible efficient computation of consistent query answers using bitwise operators. However, we have seen in Example 10 even in the presence of one functional dependency there may be *exponentially* many repairs [6]. With only 80 tuples involved in conflicts, the number of repairs may exceed 10^{12}! It is clearly impractical to efficiently manipulate bitvectors of that size.

Aggregation

Aggregation is common in data warehousing applications where inconsistencies are likely to occur. However, in the presence of aggregation operators, the notion of consistent query answer needs to be slightly adjusted.

EXAMPLE 16 *Consider Example 6. The aggregation query*

```
SELECT SUM(Tally)
FROM BrownVotes
```

returns a different answer in every repair. Therefore, it has no consistent answer in the sense of Definition 7. However, it is clear that from the information present in the database we can draw the inference that the returned sum of tallies is not completely unknown but rather falls between 843 *and* 862.

The definition below reflects this intuition.

DEFINITION 17 *[4, 6](a) A consistent answer to an aggregation query Q in a database instance r is the minimal closed interval $I = [a, b]$ such that for every repair r' of r, the scalar value $Q(r')$ of the query Q in r' belongs to I.*
(b) The left and right end-points of the interval I are the greatest lower bound *(glb) and* least upper bound *(lub), respectively, answers to Q in r.*

So in Example 16, the consistent answer is the closed interval [843, 862].

In some cases, consistent answers to aggregation queries can be efficiently computed [4, 6].

EXAMPLE 18 *The consistent answer to the aggregation query*

```
SELECT SUM(Tally)
FROM BrownVotes
```

ic computed by the following query (in the syntax of SQL:1999):

```
WITH Partial(County,MinS,MaxS) AS
  (SELECT County,MIN(Tally),MAX(Tally)
   FROM BrownVotes
   GROUP BY County)

SELECT SUM(MinS) ,SUM(MaxS)
FROM Partial;
```

The computational complexity of consistent answers to first-order and aggregation queries is further discussed below.

Computational complexity

We summarize here the results about the computational complexity of consistent query answers [2, 6, 16, 17]. We adopt the *data complexity* assumption [1, 37, 55] that measures the complexity of the problem as a function of the number of tuples in a given database instance. The given query and integrity constraints are considered fixed.

The query rewriting approach [2] – when it terminates – provides a direct way to establish PTIME-computability of consistent query answers. If the original query is first-order, so is the transformed version. In this way, we obtain a PTIME procedure for computing CQAs: transform the query and evaluate it in the original database. Note that the transformation of the query is done independently of the database instance and therefore, does not affect the data complexity. For example, in Example 10 the query $R(x, y)$ will be transformed (similarly to the query in Example 12) to another first-order query and evaluated in PTIME, despite the presence of an exponential number of repairs. The logic programming approaches described earlier do not have good asymptotic complexity properties because they are all based on Π_2^p-complete classes of logic programs [20].

The paper [16]([17] is an earlier version containing only some of the results) identifies several new tractable classes of queries and constraints. This paper contains the conflict-graph-based algorithm presented earlier, which can easily be shown to work in PTIME. Another tractable class consists of conjunctive queries where all the conjuncts do not share variables and integrity constraints that are functional dependencies, with at most one dependency per relation. The paper [16] also shows that relaxing any of those restrictions leads to co-NP-completeness. Recently, [27] has identified a class of conjunctive queries with relation arity at most 2, for which data complexity of computing consistent query answers is in PTIME, although the answers cannot be computed by query rewriting.

The paper [6] contains a complete classification of tractable and intractable cases of the problem of computing consistent query answers (in the sense of Definition 17) to aggregation queries w.r.t. a set of FDs F. Its results can be summarized as follows:

- For all SQL2 aggregate operators except `COUNT(A)`, the problem is in PTIME iff the set of FDs F contains at most one nontrivial FD ($|F| \leq 1$).

- For `COUNT(A)`, the problem is NP-complete, even for one nontrivial FD.

- If F is in BCNF and $|F| \leq 2$, then the lub-answer to COUNT (*) queries can be computed in PTIME.

Relaxing any of the above restrictions leads to NP-completeness.

Alternative repair semantics

The definition of repair (Definition 5), while intuitive, is not the only possibly one. It postulates that repairs be obtained by deleting and inserting facts, with deletion and insertion treated symmetrically. If denial constraints are the only constraints present, then only deletions can lead to repairs. However, in the presence of inclusion or tuple-generating dependencies, insertions can also be used to bring about the satisfaction of the constraints.

Several options have been explored in the literature in this context. First, one can ignore insertions and work with deletions only [16]. That is valid if we can assume that the information in the database is not necessarily correct (thus there may be inconsistencies) but it is complete. In fact, referential integrity actions in the SQL:1999 standard are limited to deletions. Second, one can allow deletions only for repairing denial constraints (there is no other way to do it) and use insertions to fix all the other constraints. This is the approach adopted in [14]. It seems suitable to data integration applications where the data is necessarily incomplete. Finally, in some cases it is natural to consider updates of individual attributes, instead of inserting/deleting whole tuples [56].

EXAMPLE 19 *Consider the relation* Emp *with attributes* Name, Salary, *and* Manager, *where* Name *is the primary key. The constraint that* no employee can have a salary greater than that of her manager *is a denial constraint:*

$$\forall n, s, m, s', m'.\ [\neg Emp(n, s, m) \vee \neg Emp(m, s', m') \vee s \leq s'].$$

Consider the following instance of Emp *that violates the constraint:*

Name	Salary	Manager
Jones	*120K*	*Black*
Black	*100K*	*Black*

Under Definition 5, this instance has two repairs: one obtained by deleting the first tuple and the other – by deleting the second tuple. It might be more natural to consider the repairs obtained by adjusting the individual salary values in such a way that the constraint is satisfied.

The first approach leads to finitely many repairs, which guarantees decidability for arbitrary queries and constraints. Some tractable cases are identified in [16]. In the second approach, there may be infinitely many repairs (in the presence of inclusion dependencies). Some decidable cases and more detailed complexity analysis are presented in [14]. In the third approach, Wijsen proposes to represent all repairs of an instance by a single *trustable tableau*. From this tableau, answers to conjunctive queries can be efficiently obtained. It is not clear, however, what is the computational complexity of constructing the tableau, or even whether the tableau is always of polynomial size.

The minimality criterion constitutes another dimension of the repair definition. The approaches discussed so far minimize the *set* of changes but it is also possible to minimize the *cardinality* of this set. The latter approach has been pursued in the context of belief revision [19]. The paper [5] contains some discussion of how the modified definition of repair can be implemented using a logic-program-based approach.

Future directions

Conflict resolution

The notion of repair can be useful not only in the context of consistent query answering. In some cases, it is necessary to compute a single repair, removing all the integrity violations. For denial constraints, this task can be accomplished in PTIME. However, if inclusion dependencies are added, the task may become co-NP-complete [16].

In general, one would like to use some form of *priority* or *preference* to influence repairing. For example, some piece of information may be more reliable or more up-to-date than another. The issue of computing repairs and consistent query answers in the presence of priorities or preferences is largely open. The work on prioritized logic programming [12, 50] may be relevant in this context.

Data integration and exchange

Assume that we have a collection of (materialized) data sources and a global, virtual database that integrates data from the sources. According to the *local-as-view* approach [42, 46, 52], we can look at the data sources as *views* over the global schema. Now, given a query to the global database, one can generate a *query plan* that extracts the information from the sources [31, 42, 43, 44]. In the global-as-view approach [41], the global database is defined as a view over the data sources.

Sometimes one assumes that certain integrity constraints hold in the global system, and those integrity constraints are used in generating the query plan; actually, there are situations where without integrity constraints, no query plan can be generated [21, 32, 35]. The problem is that we can rarely be sure that such global integrity constraints hold. Even in the presence of consistent data sources, a global database that integrates them may become inconsistent. The global integrity constraints are not maintained and could easily be violated. In consequence, data integration is a natural scenario to apply the methodologies presented before. What we have to do is to retrieve consistent information from the global database.

Several new interesting issues appear, among them are: (a) What is a consistent answer in this context? (b) If we are going to base this notion on a notion of repair, what is a repair? Notice that we do not have global *instances* to repair, (c) How can the consistent answers be retrieved from the global systems? What kind of query plans do we need? These and other issues are addressed in [10–11] for the local-as-view approach and in [40] for the global-as-view approach.

Recently, a new kind of scenario for data integration, called *data exchange* [47, 24], has been identified. In this scenario, one considers source and target databases, and the mappings between them are described using source-to-target dependencies that generalize inclusion dependencies. The distinctive feature of this scenario is that an instance of the target database is materialized, using an instance of the source database and the source-to-target dependencies. However, the issue of what to do if, for the given contents of the data sources, there is no target database satisfying the target constraints has not been addressed so far. Such a scenario would clearly require that the construction of the target database be augmented with some form of repairing the inconsistencies.

Data cleaning

The task of data cleaning [36, 48, 28] is to remove errors and inconsistencies in the data submitted to a data warehouse. Clearly, repairing integrity violations is a part of this task. However, there is much more to data cleaning. One has to resolve schematic discrepancies, combine different records describing the same entity, normalize the data etc. Domain knowledge and heuristics play a significant role in data cleaning. Declarative, high-level descriptions of the data cleaning process have been proposed [28]. However, the semantics of the databases resulting from the cleaning process has not been formally characterized so far in the literature.

Other applications

In spatial or spatiotemporal databases inconsistencies arise quite often. For example, there may be inconsistent readings of an object's location or the extent of a forest fire. Spatial or spatiotemporal objects correspond to infinite sets of points. However, those sets are usually finitely representable, e.g., using constraint databases [39]. Fortunately, the notion of repair is applicable not only to finite databases but also to infinite ones. Queries to constraint databases can be rewritten in the same way as those to relational databases, opening the possibility of computing consistent query answers in spatial or spatiotemporal databases.

Relational integrity constraints have been adapted to XML databases [25–26]. However, as XML updates are more complex than relational ones, it is not quite clear how to define repairs and consistent query answers in that context.

Conclusions

We have presented a survey of recent work on consistent query answering. This work has identified relevant expressiveness vs. tractability tradeoffs and proposed a variety of computational approaches. Many semantic and computational issues in this area remain to be explored. New application scenarios promise also to bring a variety of new problems.

A more detailed survey of consistent query answering that contains also an extensive discussion of related work is [9].

Acknowledgments

Thanks go to all those who have collaborated with me in the area of consistent query answering, in particular Marcelo Arenas, Leopoldo Bertossi and Jerzy Marcinkowski. The support of NSF under the grant IIS-0119186 is gratefully acknowledged.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Specifying and Querying Database Repairs Using Logic Programs with Exceptions. In *International Conference on Flexible Query Answering Systems (FQAS),* pages 27–41. Springer-Verlag, 2000.

[4] M. Arenas, L. Bertossi, and J. Chomicki. Scalar Aggregation in FD-Inconsistent Databases. In *International Conference on Database Theory (ICDT),* pages 39–53. Springer-Verlag, LNCS 1973, 2001.

[5] M. Arenas, L. Bertossi, and J. Chomicki. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming,* 3(4–5):393–424, 2003.

[6] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science,* 296(3):405–434, 2003.

[7] M. Arenas, L. Bertossi, and M. Kifer. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *International Conference on Computational Logic,* pages 926–941. Springer-Verlag, LNCS 1861, 2000.

[8] P. Barcelo and L. Bertossi. Logic Programs for Querying Inconsistent Databases. In *International Symposium on Practical Aspects of Declarative Languages (PADL),* pages 208–222. Springer-Verlag, LNCS 2562, 2003.

[9] L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases.* Springer-Verlag, 2003.

[10] L. Bertossi, J. Chomicki, A. Cortes, and C. Gutierrez. Consistent Answers from Integrated Data Sources. In *International Conference on Flexible Query Answering Systems (FQAS),* Copenhagen, Denmark, October 2002. Springer-Verlag.

[11] L. Bravo and L. Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *International Joint Conference on Artificial Intelligence (IJCAI),* 2003. To appear.

[12] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence,* 109(1-2):297–356, 1999.

[13] F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems.* Chapman &Hall, 1997.

[14] A. Cali, D. Lembo, and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *ACM Symposium on Principles of Database Systems (PODS),* pages 260–271, 2003.

[15] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems,* 15(2):162–207, 1990.

[16] J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. Technical Report cs.DB/0212004, arXiv.org e-Print archive, December 2002. Under journal submission.

[17] J. Chomicki and J. Marcinkowski. On the Computational Complexity of Consistent Query Answers. Technical Report arXiv:cs.DB/0204010, arXiv.org e-Print archive, April 2002.

[18] J. Chomicki, J. Marcinkowski, and S. Staworko. Computing Consistent Query Answers Using Conflict Hypergraphs. Submitted, September 2003.

[19] M. Dalal. Investigations into a Theory of Knowledge Base Revision. In *National Conference on Artificial Intelligence,* St.Paul, Minnesota, August 1988.

[20] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys,* 33(3):374–425, 2001.

[21] O.M. Duschka, M.R. Genesereth, and A.Y. Levy. Recursive Query Plans for Data Integration. *Journal of Logic Programming,* 43(1):49–73, 2000.

[22] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving in DLV. In J. Minker, editor, *Logic-Based Artificial Intelligence,* pages 79–103. Kluwer, 2000.

[23] T. Eiter, M. Fink, G. Greco, and D. Lembo. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *International Conference on Logic Programming (ICLP),* 2003.

[24] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *International Conference on Database Theory (ICDT),* pages 207–224. Springer-Verlag, LNCS 2572, 2003.

[25] W. Fan, G. Kuper, and J. Simeon. A Unified Constraint Model for XML. *Computer Networks,* 39(5):489–505, 2002.

[26] W. Fan and J. Simeon. Integrity Constraints for XML. *Journal of Computer and System Sciences,* 66(1):254–201, 2003.

[27] A. Fuxman and R. Miller. Towards Inconsistency Management in Data Integration Systems. In *IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03),* 2003.

[28] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C-A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *International Conference on Very Large Data Bases (VLDB),* pages 371–380, 2001.

[29] P. Gärdenfors and H. Rott. Belief Revision. In D. M. Gabbay, J. Hogger, C, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming,* volume 4, pages 35–132. Oxford University Press, 1995.

[30] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing,* 9(3/4):365–386, 1991.

[31] G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *International Conference on Database Theory (ICDT),* pages 332–347. Springer-Verlag, LNCS 1540, 1999.

[32] J. Grant and J. Minker. A Logic-Based Approach to Data Integration. *Theory and Practice of Logic Programming,* 2(3):323–368, 2002.

[33] G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *International Conference on Logic Programming (ICLP),* pages 348–364. Springer-Verlag, LNCS 2237, 2001.

[34] S. Greco and E. Zumpano. Querying Inconsistent Databases. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR),* pages 308–325. Springer-Verlag, LNCS 1955, 2000.

[35] J. Gryz. Query Rewriting using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems,* 24(7):597–612, 1999.

[36] M. Hernandez and S. Stolfo. The Merge/Purge Problem for Large Databases. In *ACM SIGMOD International Conference on Management of Data,* pages 127–138, 1995.

[37] P. C. Kanellakis. Elements of Relational Database Theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science,* volume B, chapter 17, pages 1073–1158. Elsevier/MIT Press, 1990.

[38] R. Kowalski and F. Sadri. Logic Programs with Exceptions. *New Generation Computing,* 9(3/4):387–400, 1991.

[39] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases.* Springer-Verlag, 2000.

[40] D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. In *9th International Workshop on Knowledge Representation meets Databases (KRDB'02),* Toulouse, France, 2002.

[41] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM Symposium on Principles of Database Systems (PODS),* 2002. Invited talk.

[42] A. Y. Levy. Combining Artificial Intelligence and Databases for Data Integration. In *Artificial Intelligence Today,* pages 249–268. Springer-Verlag, LNCS 1600, 1999.

[43] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-Answering Algorithms for Information Agents. In *National Conference on Artificial Intelligence,* pages 40–47, 1996.

[44] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *International Conference on Very Large Data Bases (VLDB),* pages 251–262, 1996.

[45] J. Minker. On Indefinite Databases and the Closed World Assumption. In *International Conference on Automated Deduction (CADE),* pages 292–308. Springer-Verlag, LNCS 138, 1982.

[46] A. Motro. Multiplex: A Formal Model for Multidatabases and Its Implementation. In *International Workshop on Next Generation Information Technology and Systems (NGITS),* pages 138–158. Springer-Verlag, LNCS 1649, 1999.

[47] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *International Conference on Very Large Data Bases (VLDB),* pages 598–609, 2002.

[48] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin,* 23(4):3–13, 2000.

[49] R. Reiter. On Closed World Databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases,* pages 55–76. Plenum Press, 1978.

[50] C. Sakama and K. Inoue. Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence,* 123:185–222, 2000.

[51] P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence,* 138(1-2):181–234, June 2002.

[52] J. D. Ullman. Information Integration Using Logical Views. In *International Conference on Database Theory (ICDT),* pages 19–40. Springer-Verlag, LNCS 1186, 1997.

[53] R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems,* chapter 10. Kluwer Academic Publishers, Boston, 1998.

[54] D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *European Conference on Logics for Artificial Intelligence (JELIA),* pages 432–443. Springer-Verlag, LNAI 2424, 2002.

[55] M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC),* pages 137–146, 1982.

[56] J. Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *International Conference on Database Theory (ICDT),* pages 378–393. Springer-Verlag, LNCS 2572, 2003.

[57] M. Winslett. Reasoning about Action using a Possible Models Approach. In *National Conference on Artificial Intelligence,* 1988.

ROLE OF CERTIFICATION IN MEETING ORGANISATION SECURITY REQUIREMENTS

William List CA FBCS
Wm. List & Co., W.list@ntlworld.com

Abstract: Security in systems is now a top priority. Management in organisations wish to be assured that their systems are reliable and that the information provided to stakeholders is secure and correct. This paper explores briefly the two main ISO standards for security - the Common Criteria and the 7799 family. It identifies current limitations in the standards and suggests area where the standards could be developed to assist everyone in meeting the future security needs

Key words: Common Criteria, ISO 17799, Information Security, Improvements, Internal Control, evaluation, certification

1. INTRODUCTION

Management wishes to have confidence that the systems they have put in place are functioning as expected and are able to continue to operate in adverse conditions. This is essentially the requirement for an Internal Control system mandated in the OECD guidance on Corporate Governance.

Part of these procedures are automated and there are two main ISO standards Common Criteria (ISO/IEC 15084) and ISO/IEC17799 (BS7799 part 2:2002) which are the basis of evaluation or certification. In addition, and not discussed here, are standards from ISACA (CobiT), IIA, and various other organisations.

The objective of this paper is to raise some issues about certification and evaluation for consideration by those involved as developers, certifiers or evaluators and the people who seek to rely on the certificates issued.

2. BACK GROUND

The paper briefly addresses the Common Criteria and 7799 family. The paper also considers the impact of risk analysis and the problems of misunderstandings.

2.1 The Common Criteria (CC)

This standard comprises:

- A set of components of security functionality.
- A set of rules for the creation of 'Targets of Evaluation (TOE) and Security Targets (ST)
- A methodology for evaluation of the ST or TOE

A TOis the specification of security functionality as prepared by the organisation submitting a product for evaluation. A ST is the specification of the requirements for a specific implementation.

In addition there are documents called Protection Profiles (PP) which set out generically the totality of the security requirements.

The common criteria is good for evaluation of hardware and/or software products (or groups of products). It does not cover the people element of security except in so far as the people procedures are asserted in the TOE or ST.

The limitations of the Common Criteria are:

- Functionality for batch processing (and essential element in many business processes) is missing.
- Requirement to build in mechanisms to report the contents of files (e.g. access tables) is not specified.
- Product manufacturers select those security functions for evaluation. There may be other security functions in the product, which are not evaluated.
- The evaluation methodology is based on the assumption that the development documentation available for evaluation is not for evolutionary product.

- It is suitable for a system only if the system is a collection of products excluding the people – its effectiveness decreases the more people intervention is required for the system to function.
- It covers security functionality only. If the rest of the product is poor or malfunctions this may cause the results of processing to be wrong (but securely wrong!).

2.2 The 7799 family

The 7799 family comprises two standards:

- ISO/IEC 17799 – at present the 2000 version – which is a list of some 125 controls. Organisations select appropriate controls from the list to meet their identified information security needs.
- BS7799 Part 2 – at present the 2002 version – which is the specification for an information security management system (ISMS). Organisations follow the standard to create a system for the ongoing management of information security based on risk analysis.

The 7799 family cover all aspects of Information security; hardware, software and people. Fundamental to the use of the Family is the decision on the scope of the security implementation. The scope could be the entire organisation or some part of it.

Limitations of the 7799 family

- The list of controls in 17799 is biased to the security of the IT infrastructure in that the controls identified for business processes are more concerned with development principles than detailed controls extant in line user departments or specific processes.
- The framework for an ISMS as set out in BS7799 Part 2 is applicable to all situations but if applied within an IT scope may not fully address the business process requirements. This is often true if the people involved have limited experience of the business processes.

2.3 Role of Risk analysis

Risk analysis is fundamental to any decisions on the extent to which security measure are required in a product or system. On the basis of the identified threats and vulnerabilities appropriate controls are implemented to cover the risks within the scope of the analysis.

There are always residual risks that are deemed acceptable and there are risks omitted from the analysis and no process is 100%. The result is that certain impacts will occur and there requires to be processes to find the impacts and take appropriate action.

Where the scope of a certification or evaluation is anything less than the whole organisation the risk analysis will be limited and may therefore place undue weight on certain risks within scope when looked at from the viewpoint of the organisation as a whole.

The question is 'to what extent is the quality of the risk analysis a subject for certification or evaluation?'

In the 7799 schemes provided the risk analysis is accepted by the appropriate official it is unlikely that the auditors will challenge it.

In Common Criteria evaluations the analysis may be challenged but only in an abstract context of a product not the live environment of which the evaluators have no knowledge.

2.4 A material problem of misunderstanding?

There are a number of terms in the standards and common usage, which do not necessarily have a consistent meaning. Within individual security (or other) communities the terms are broadly agreed; but outside those communities other definitions exist. The possibility of misunderstanding between individuals and organisations implementing 'information security' and/or certifying or evaluating systems or products are quite substantial. This potential misunderstanding may materially limit the value or comprehension of a certificate or evaluation report to organisations wishing to rely on them.

To enumerate three particular areas as examples of the possible misunderstandings I cite systems, users and audit:

2.4.1 Systems

A system is what the writer (speaker) believes it to be at the time. It could be any or all of the following:

- The hardware
- The operating system - or other infrastructure software e.g. a DBMS
- The business process applications
- Including the IT people
- Including the line user people up to the Board level

The problem is whether the reader (or listener) understands the same system as the writer. This is a complication when people are discussing system certification or evaluation - what do they mean?

2.4.2 Users

In any particular situation it is reasonably clear who the users are. In general however there are at least three possible interpretations:

- Persons who are in the organisation where IT functionality is used in delivering the business objectives;
- Persons who are in the IT department;
- Customers if you are a vendor of hardware or software.

2.4.3 Audit

Audit in many security standards is considered as writing a log (after determining what content should be written to the log).

Audit is considered as the process of creating a certificate or evaluation (sometimes also called evaluation).

Audit may be an internal process whereby independent people check on the work done by other people or automated processes.

Audit is also the term for the performance of financial and other external examinations under a variety of company legislation.

3. ARE EVALUATIONS AND CERTIFICATIONS OF ANY VALUE?

The current evaluation of products and certification of security in systems are valuable. They provide a degree of assurance that specified objectives have been achieved. They are not perfect and therefore absolute trust in the results is unwise.

3.1 CC Evaluations

The evaluations performed under the CC are appropriate for hardware and software in any combination. They are best when the product requires no external activity to perform correctly and their value diminishes as the amount of external activity increases. This is because the CC does not address people issues but only the functionality of hardware or software.

They suffer from the limitations set out in 2.1 above

3.2 7799 certifications

The usefulness of the certification depends on the scope of the ISMS. The wider the ISMS the more useful the certification is. A 7799 certificate indicates that the organisation has addressed information security in a systematic manner. It does not indicate if specific procedures are in place (which third parties may require) nor necessarily does it indicate effective security is in place.

3.3 Non security functionality

CC Evaluations and 7799 Certification are limited to security functionality. They do not intend to address other functions that the systems or products perform. Clearly if the non security functionality in a system fails to perform as expected then the result will be some form of error in the information in the system or reported from the system. Therefore the integrity, in the widest sense, of the information is compromised.

It is possible to set the scope of a 7799 certification to include the necessary controls to address malfunctioning of the non security functionality but it is not possible for CC evaluations.

4. WHAT DOES THE BOARD WANT?

The board expects:

- Good information to work on;
- That the system(s) are available when required;
- No loss of confidentiality for organisation's secrets;
- Reasonable confidentiality for legal reasons;
- Reasonable compliance with laws;
- No hindrance to meting business objectives.

These taken together will meet the requirement for the Board to effectively manage the organisation as set out in the OECD Corporate Governance guidelines.

At present the certification and evaluation processes are meeting parts of these objectives but not all of them.

5. WHAT COULD BE DONE TO IMPROVE THE SITUATION?

I would like to suggest that the development work at a standards level in regard to the CC and 7799 address the following issues to expand the current standards so as to more closely address the Board's requirements.

5.1 Within the CC

To develop the processes for the development components of the evaluation process to:

- Accommodate the development process for commercial (particularly business process) products;
- Permit a 7799 certification of the development process to be considered particularly for smaller developers.

To enhance the commercial value of the products evaluated by including components, which address commercial security requirements over and above those presently included. Consider evaluation of products being used in the automation of the business process. Possibly also to consider developing mechanisms whereby processes embedded in business process applications could be confirmed as being compliant with applicable laws and regulations.

5.2 Within the 7799 family

Ensure that the linkage between Information Security and the other parts of the Internal Control system is made explicit in the guidance provided. In general this could be achieved by developing guidance:

- To explain how IT risk fits in with the overall business risk;
- How business process security can be effected particularly in the area of the 'correctness' of the results presented to management and stakeholders.

To explore possible metrics which would enable people to evaluate better the quality of the ISMS.

6. CONCLUSION

In the future more and more of the business and home process will be automated. It is vital that these processes are secure and deliver correct information to the users of the systems.

Our present set of standards grew out of the needs of the large mainframe systems and need to be changed to reflect the reality of the systems of the future. In the security world we need a debate about what changes are required so that those charged with delivering security in the future have the tools they really need to do a first class job.

This paper has set out briefly the current major ISO standards and has made some suggestions as how they might be changed for the future.

Grand Challenges in Data Integrity and Quality: Panel Discussion

Bhavani Thuraisingham[1]
The MITRE Corporation, USA
** On Leave at the National Science Foundation, Arlington VA 22230, USA*

Abstract: This paper provides a summary of the panel discussions on data quality at the IFIP Integrity and Internal Control in Information Systems Conference held in Lausanne, Switzerland, November 2003.

Key words: Data Quality, Data Integrity, Data Security, Quality of Service, Semantic Data Quality, Data Quality Algebra

1. INTRODUCTION

Data integrity and quality have received a lot of attention recently. Data integrity and quality are about maintaining the correctness, accuracy and quality of the data. Especially in a heterogeneous and web-based environment, data may emanate from all kinds of sources. Therefore it is difficult to trust the accuracy and quality of the data. Data is however a critical resource in almost all organizations. It is critical that appropriate techniques are developed to ensure data integrity and quality so that data users can carry out their functions effectively.

[1] DISCLAIMER: The views and conclusions contained in this paper are those of the author and the author's understanding of the discussions at the conference. They do not reflect the policies and procedures of the National Science Foundation or of the MITRE Corporation.

While research in data integrity and data quality has progressed a great deal, there is still a lot to do. Although we now have data cleansing and data quality tools, we need a strong research program to advance the field to ensure the quality of the data in a web-based world. Because of the importance of the area we organized a panel at the IFIP Integrity and Internal Control in Information Systems Conference in Lausanne, Switzerland held in November 2003. The panelists were the following researchers (in alphabetical order):

Karl Aberer, EPFL, Lausanne, Switzerland
Yves Deswarte, LAAS-CNRS, Toulouse, France
Sushil Jajodia, George Mason University, USA
Leszek Lilien, Purdue University, USA
Indrakshi Ray, Colorado State University, USA
Monica Scannapieco, University of Rome, Italy

The panelists discussed many challenges. In section 2 we summarize the challenges. Some directions are given in Section 3.

2. PANEL DISCUSSION

The challenges discussed by the panelists are the following:
- Integrating security and integrity
- Quality of Service
- Secure high integrity transactions
- Semantic data quality
- Algebra for data quality
- Trust management and data quality

Below we will elaborate on the challenges for each of the topics listed above:

Integrating security and integrity: There have been many discussions over the years on integrating security and integrity. From a security point of view integrity is about unauthorized modifications of the data. Can we develop solutions to detect and prevent unauthorized modifications as well as malicious corruptions of data? Can we use data mining techniques to detect and possibly prevent faults? How can we develop software that is self-healing? How can we recover from the faults in a timely manner?

Quality of Service: Closely related to the first topic is developing quality of service primitives. We need flexible security and integrity policies. For example, how can we effectively integrate security, integrity, fault tolerance and real-time processing to achieve dependable computing? How can we make tradeoffs say between security, integrity, fault tolerance and real-time computing? For example, in some situations we may need 100% security while in some other cases it is critical that the transactions meet the timing constraints and therefore we may be willing to sacrifice some level of security. How can we specify the flexible policies and subsequently implement the policies in operational systems? How can we ensure that there is end-to-end security and survivability and ensure that systems that are composed are secure and survivable?

Secure High Integrity Transactions: There has been a lot of research on secure transactions and especially on multilevel secure transactions. The idea is for high-level transactions not to interfere with low level transactions and send covert messages. The challenge is to ensure that the integrity constraints and data quality constraints are satisfied by the transactions. There has been work on integrity constraint processing during transaction processing. However integrating security, integrity, data quality and transaction processing remains a challenge.

Algebra for Data Quality: Various algebras have been proposed for data manipulation including relational algebra. Can we extend these algebras to include data quality? One simple way to accomplish this is to have a data quality attribute associated with each attribute in a relation. Then we need to develop an algebra so that data quality values can be computed for the results. So the challenge is how can we develop algebra for manipulating data quality values as well as handle integrity constraints?

Semantic data quality: Database community has worked on semantics for a number of years. They have developed solutions to semantic heterogeneity, a problem that occurs when integrating heterogonous databases. The community is now moving in the direction of information integration where information from diverse disparate data sources has to be integrated on the web. Ontologies have been specified to aid information integration. The goal is to move toward the semantic web. Data quality plays a crucial role. In order to ensure that the data is accurate and consistent for the semantic web, we need to develop data quality techniques for the semantic web. We need to understand the nature of the data and consider the semantics when assigning data quality values. Essentially data quality has to be considered when discussing the semantics of the data. Ontologies have to specify data quality, data integrity as well as data security features. In summary, ensuring data quality for secure information integration on the web remains a challenge.

Trust management: Security researchers are working on trust management issues. The challenge is how do you trust the data and information provided by the publisher? The publisher may have received the data from another source and that source may have received the data form yet another source? One's trust of the quality of the data depends on how much he or she trusts the sender. The sender in turn trusts the quality of data depending on who sends it to him or her. Can we develop a theory of trust? How can we carry out trust negotiations with respect to data quality? That is, can A and B negotiate with each other and develop trust mechanisms so that they can effetely share data and collaborate on tasks? How can we ensure trust and integrity in a peer-to-peer environment? There are many challenges in trust management that need to be addressed for data quality.

3. DIRECTIONS

The panel provided some good insights into directions and challenges in data quality and data integrity. While the panelists were from academia, there was also industry participation at the conference and we got some useful information on data quality for various industries. For many applications in the commercial world, including defense, intelligence, banking, financial, and medical areas, data quality is critical. How can we act on the data for critical applications in medicine, finance and defense if we are not certain of the quality? There was a general consensus that while everyone agrees that data quality and integrity and critical, the areas have not received a lot of attention in the research world.

The challenge is for the various research communities to actively pursue research and convince the research funding organizations that there is a clear need to start research in this area. It was noted that there are now many research funding initiatives in both data management and information security. The question is, for now at least should we focus on data quality and data integrity as part of the research programs in data management and information security? This way the area will get some focus. Then as we make more progress and uncover more challenges, we could devote more resources in this area.

The group also felt that the panel has raised many interesting ideas and discussed some key challenges. We should incorporate some of the ideas presented in the call for papers for next year's conference so that we can keep the research going in this very important field.

INDEX OF CONTRIBUTORS

INDEX OF KEYWORDS