# Towards Security Hardening of Scientific Demand-Driven and Pipelined Distributed Computing Systems

Serguei A. Mokhov

Department of Computer Science and Software Engineering
Faculty of Engineering and Computer Science
Concordia University, Montreal, Quebec, Canada
E-mail: mokhov@cse.concordia.ca

## Abstract

*This work highlights and takes aim at the more critical security aspects required for two different types of distributed systems for scientific computation. It covers two open-source systems written in Java: a demand-driven system – General Intensional Programming System (GIPSY) and a pipelined system – Distributed Modular Audio Recognition Framework (DMARF), which represent the distributed scientific computational engines as case studies with respect to the security aspects in the presence of a malicious threat locally on the nodes and on the network that can drop, inject, alter data, poison data value warehouse(s) as well as incite malicious code injection. More specific goals include data/demand integrity, data/demand origin authentication, confidentiality, high availability, and malicious code detection. We address some of the goals to a degree, some with the Java Data Security Framework (JDSF) as a work-in-progress.*

## 1. Introduction

We introduce the purpose of the research presented in this paper and then the two use-case systems we apply it to. We begin with the problems we are taking aim to address, the intermedia proposed solution details and their limitations.

**Problem Statement.** When rather complex research and scientific distributed systems are developed, sometimes ad-hoc or sporadically, in research institutions, such as universities, or during a variety of course projects, by researchers and students alike who come and go, there are little-to-no rigor and formality usually present in typical software requirements engineering processes when such systems are designed and implemented. This especially applies to the security aspects of such systems when they run over the public Internet connections, as the security requirements are often neglected in favor of the scientific computation functionality, parallelism, and the efficiency thereof or even often just a possibility of distributed computation itself using various distributed technologies. The security risks relate to maliciously induced incorrect computation results and cache (called a value warehouse that is used for dynamic computation to avoid recomputing the same result value) "poisoning". Other issues revolve about availability issues, spread of malware using the discussed systems for transport as attack vectors, and if the systems work on the analysis of confidential data, then the confidentiality issues arise as well. While some of the mentioned threats can be overcome by the underlying middleware technologies and protocols that already have security mechanisms in place (e.g. over SSL or SSH, security options in SNMPv3, proprietary protocols), they often cannot cope with some aspects of the computation, e.g. denial of service or malicious code.

**Proposed Solution.** We overview two concrete scientific research distributed systems, their needs and properties in order to solicit the security requirements for two types of distributed computing engines, draw on their parallels and differences, generalize them, and offer some solutions. The authors believe this analysis would apply to many systems of these two kinds. We review the applicability of the Java Data Security Framework (JDSF), proxy certificates, and other solutions that can be implemented for the security layer of the studied systems. We do realize the overhead of any security mechanisms introduced vs. computation and communication performance that often researchers are looking for, so we define one of the requirements of the security layer being optional and being turned on on-demand (dynamically recongfigurable) when the computing nodes need to communicate to the outside of the local area network potentially through a public channel.

**Organization.** The article is structured in such a way as to get the reader the idea of the two distributed systems used as case studies and their particularities followed by the brief description of the JDSF. Then we review the threat model,

security requirements, and their applicability to the two systems, and then we conclude.

### 1.1. General Intensional Programming System (GIPSY)

**GIPSY Overview.** The General Intensional Programming System (GIPSY) [28, 30, 36, 10, 31, 16, 29, 27] is an open-source platform implemented primarily in Java to investigate properties of the Lucid [1, 2, 3] family of intensional programming languages and beyond. GIPSY being developed and maintained by the GIPSY Research and Development Group at Concordia University, Montreal, Canada. It is a distributed system, designed as a modular collection of frameworks where components related to the development (RIPE, a Run-time Integrated Programming Environment, implemented in `gipsy.RIPE`), compilation (GIPC, the General Intensional Programming Compiler, implemented in `gipsy.GIPC`), and execution (GEE, the General Eduction Engine, implemented in `gipsy.GEE`) of Lucid programs are separated allowing easy extension, addition, and replacement of the components.

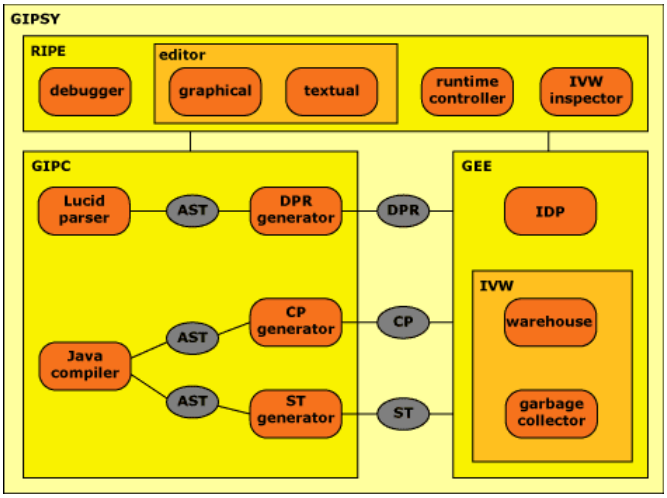


**Figure 1. High-level Structure of the GIPSY**

GIPSY has a collection of compilers under the General Intensional Programming Compiler (GIPC) framework and the eduction execution engine (GEE) among other things (see general architecture in Figure 1. The two modules are the major primary components for compilation and execution of intensional programs, which will require amendments for the changes proposed in this work. In [16] and previous works [4, 15, 14] a notion of JLucid and Objective Lucid languages (with sequential threads and communication procedures) was introduced that led to a more general framework for greater interoperability between intensional and imperative programming languages – GICF. The availability of the Demand Migration System (DMS) [37] makes GIPSY a truly distributed system. With the greater flexibility that the languages, GICF, and DMS brought in there arise the issues of security and demand monitoring. GIPSY is also used as an investigation platform for the distributed cyberforensics evaluation of the Forensic Lucid programs [13, 25, 26]. GEE is the component where the distributed demand-driven evaluation takes place, e.g. relying on the DMS. Thus, GIPSY has some raw implementation of RMI and the one that relies on the DMS for its transport needs.

**Introduction to GICF.** GLU [8, 7], JLucid [4], and later Objective Lucid [15] prompted the development of a General Imperative Compiler Framework (GICF). The framework targets the integration (embedding of) different imperative languages into GIPSY (see [36]) programs for I/O, portability, extensibility, and flexibility reasons. GLU promoted C and Fortran functions within; JLucid/Objective Lucid promoted embedded Java. Since GIPSY targets to unite all intensional paradigms in one research system, it makes an effort to be as general as possible and as compatible as possible and pragmatic at the same time. The GICF is there if we want to be able to run, for example, GLU programs with minimum (if at all) modifications to the code base. GIPSY is extended with a module in this case (a compiler) to support C-functions as it does for Java. GICF is made extensible such that later on the language support for C++, Perl, Python, shell scripts, or whatever can be relatively easily added. Wiht GICF it is also possible to have a multi-segment multi-language (with multiplicity of 3 or more languages) GIPSY program with embedded code. The provided `embed()` call allows the loading of code from a variety of locations via standard protocols, such as HTTP, HTTPS, FTP, and other Internet protocols via the URL parameter.

**Security Problems Associated with GICF.** JLucid, Objective Lucid, and GICF opened up doors for very flexible use of external languages and resources as a part of coarse-grain intensional computation. Unfortunately, there are security considerations to deal with when embedding a potentially vulnerable unsigned code from a possibly untrusted remote location and then propagate it to all the workers participating in computation, which can result in either gaining some unwanted privileges to the attackers, installing backdoors, or performing DDoS through the compromised worker nodes. The gained privileges would be that of the user running a GIPSY service the malicious payload can exploit in remote code execution or backdoor installation; the DDoS in the simplest case can include a worker payload with an infinite loop in its implementation that either opens files, network connections, becomes a DDoS "zombi" for future attacks, or spawns processes until the targeted host is down. While some of these can be prevented at the op-

erating system level or the JVM by limiting the amount of resources (memory, file descriptors, sockets, etc.) service processes are allowed acquire, it is still better to prevent such code from propagation in the first place.

**Demand Migration System.** The Demand Migration System (DMS) is an implementation of the Demand Migration Framework (DMF) introduced by Vassev and extended by Pourteymour in [37, 32]. The initial version of the DMS relies on Jini [9] for transport and storage of the demands and the results in a JavaSpaces [11] repository acting as a data warehouse cache for the most frequently demanded computations and their results (demand store). The DMF is an architecture that is centered around the demand store with transport agents (TAs) that implement a particular protocol (as a proof-of-concept Jini was used, now a JMS [35] TA is under works [32]) to deliver demands between the demand store, workers (that do the actual computation for procedural demands), and generators (that request the computation to be done). Thus, the distributed security aspect here solely relies on the underlying communication protocols (the bottom line is Java RMI [39]), but provides no other measures or a way to harden the system if it runs over a public unsecured network where the trust is not fully established between the communicating agents, generators, and the demand store (that may all reside in any number of hosts in any quantity) and there is nothing preventing alteration of the low-level packets with corrupt/incorrect results or malicious code that either to spread or hog the computation or network otherwise.

## 1.2. Distributed MARF

DMARF [12] is based on the classical MARF (introduced below) whose pipeline stages were made into distributed nodes and later extended to be managed over SNMPv2 [22].

**Classical MARF.** The Modular Audio Recognition Framework (MARF) [17, 19, 18, 20] is another open-source research platform and a collection of pattern recognition, signal processing, and natural language processing (NLP) algorithms written in Java and arranged into a modular and extensible framework facilitating addition of new algorithms for use and experiments by scientists. MARF can run distributively [12] over the network, run stand-alone, or may just act as a library in applications. MARF has a number of algorithms implemented for various pattern recognition and some signal processing tasks. The backbone of MARF consists of pipeline stages that communicate with each other to get the data they need in a chained manner. In general, MARF's pipeline of algorithm implementations is presented in Figure 2, where the implemented algorithms

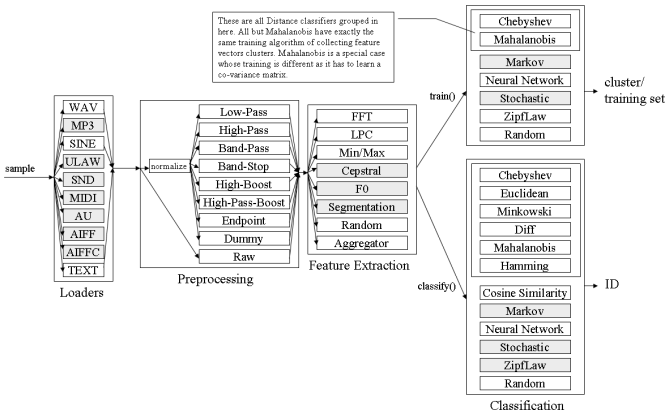


**Figure 2. MARF's Pattern Recognition Pipeline**

are in white boxes, and the stubs or in-progress algorithms are in gray. The pipeline consists of four basic stages: sample loading, preprocessing, feature extraction, and training/classification. There are a number of applications that test MARF's functionality and serve as examples of how to use MARF's modules. One of the most prominent applications `SpeakerIdentApp` – Text-Independent Speaker Identification (who, gender, accent, spoken language, etc.).

**Distributed Version and SNMP.** The classical MARF presented in the previous section was first extended [12] to allow the stages of the pipeline to run as distributed nodes as well as a front-end, as roughly shown in Figure 3. The basic stages and the front-end were implemented without backup recovery or hot-swappable capabilities at this point; just communication over Java RMI [39], CORBA [33], and XML-RPC WebServices [34]. Later, the DMARF was further extended to allow management of its nodes with SNMP [22] by implementing the proxy SNMPv2 [5] agents and translating some of the management information to DMARF's "native" operations. For the DMARF, the research and development group received its own SMI number [6]. There is also an undergoing project on the intensional scripting language, MARFL [21] that allows to script MARF tasks and applications and allows them to be run distributively either using MARF's own infrastructure or over the GIPSY. The latter fact presents the new hazards similar to those described in GIPSY. Thus, the security of this distributed system as-is relies again on the underlying protocols of Java RMI, CORBA, XML-RPC, and SNMP. The latter does provide some extensive security features for the management information in the v3 of the protocol, but as-is nothing assures *data* integrity and origin authentication. Unlike GIPSY, DMARF doesn't deal with explicit code execution, so there is no malicious code injection problem in this case.

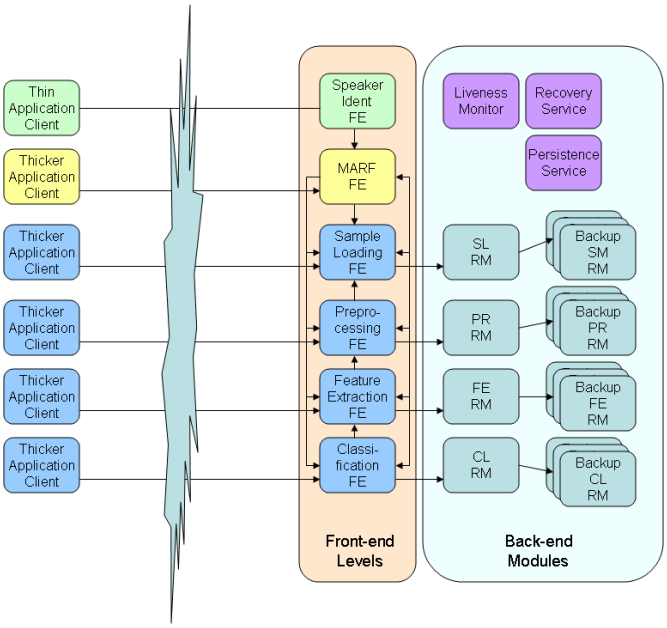


**Figure 3. The Distributed MARF Pipeline**

### 1.3. Comparison of GIPSY and Distributed MARF

Here we summarize the similarities and differences between the two systems in order to extract common and specific secruity-related requirements:

**Demand-driven vs. pipelined.** The GIPSY employs a demand-driven eductive execution model: its General Eduction Engine (GEE) evaluates intensional (Lucid dialects) or imperative expressions (e.g. Java methods, C++ functions, etc.) in presence of which a procedural demand is generated, delivered to a networked demand store (we call it a data value warehouse or cache), where it can be picked up from by an observing worker on some other node for evaluation that procedure as an atomic sequential thread. Once worker completes the evaluation, the result of the computation is stored back into the warehouse for the generator to pickup and return back to the executing program. The described type of distributed asynchronous communication is managed by the earlier described DMS [37, 38]. DMARF due its nature of application, implements a pipelined or chained way of connecting some of its distributed nodes following the pattern recognition pipeline stages, thereby making it look synchronous. While there is a synchrony within the pipeline, it is formed for the current subject/sample evaluation, and the pipeline connection path may be different for each new subject or sample. DMARF by itself does not exhibit the demand-driven model of computation (though can be adjusted to use one as such) and allows applications connecting directly to some of the nodes avoiding the pipeline if they just require the services of those particular nodes.

**Java.** Both systems are written in the Java language and share some common utility base. They also overlap in the use of the available Java communication technologies, such as the RMI. DMARF has Web Services and CORBA implementation, while GIPSY has Jini and JMS.

**Author.** The author of this paper actively participates in the design and development of both systems along with his colleagues and as such has an in-depth knowledge of the two systems to search for a viable common add-on solution of the security layer and defining its properties.

### 1.4. Java Data Security Framework

JDSF [23, 24] is one of the proposed frameworks to allow security researches working with several types of data storage or databases in Java to evaluate different security algorithms and methodologies in a consistent environment. The JDSF design aims at the following aspects of data security: confidentiality (data is private), integrity (data is not altered in transit), origin authentication (data coming from a trusted source), and SQL randomization (for relational databases only). An abstraction of the common essential cryptographic primitives are also provided. The abstractions expose a common API proxied to the common Java open-source implementations of the cryptographic algorithms for encryption, hashing, digital signatures, etc. The higher-level JDSF design summary is illustrated in several UML specification diagrams: Figure 4 illustrates main subpackages of JDSF and its configuration, Figure 5 illustrates the design of security-enhanced storage, Figure 6 illustrates the design of the authentication subframework, Figure 7 illustrates the design of the privacy subframework, Figure 8 illustrates the design of the integrity subframework, and Figure 9 illustrates the design of the abstraction API over concrete cryptographic primitives. The JDSF is naturally convenient to use in the scope of the research in the article as it is, like the GIPSY and DMARF, is implemented in Java, open-source, and evolved in some relationship with MARF and is within its CVS repository. The main disadvantage of JDSF as it focuses on the data storage only and can't address all of our concerns here, in particular it can't deal with DDoS, and at present has no provision for malicious code detection, or any type of network protocol security other than secure serialization to a networked storage.

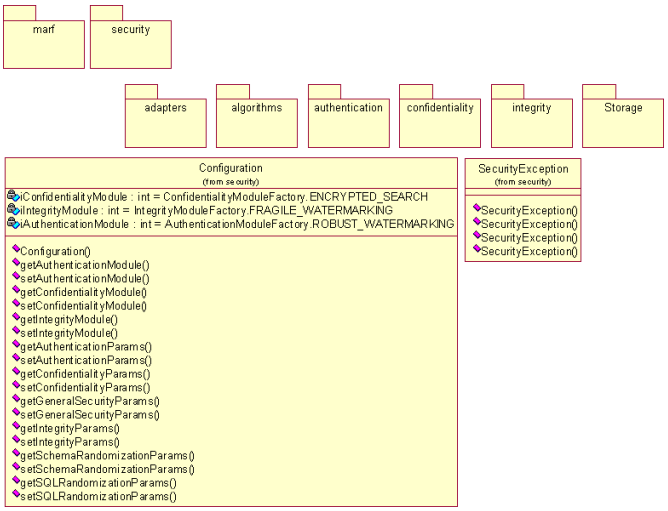


**Figure 4.** `marf.security` **Package.**

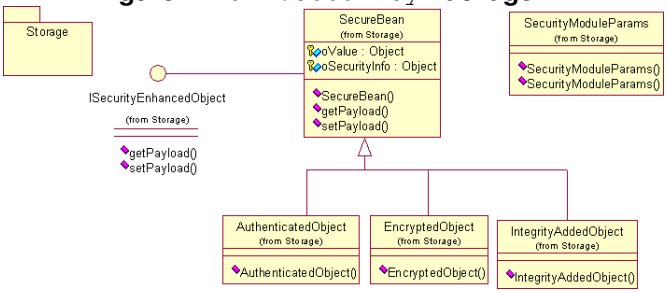


**Figure 5.** `marf.security.Storage` **Package and Classes.**

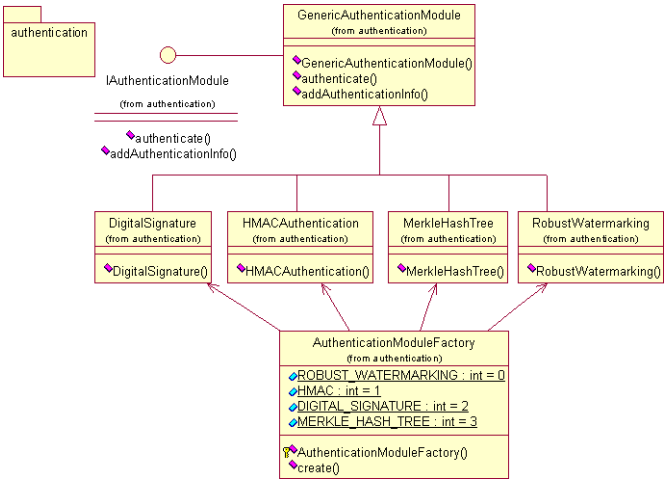


**Figure 6.** `marf.security.authentication` **Package and Classes.**

## 2. Threat Model, Security Requirements, and Proposed Solutions

The threat model has to do with the management and computation information's confidentiality, integrity, avail-

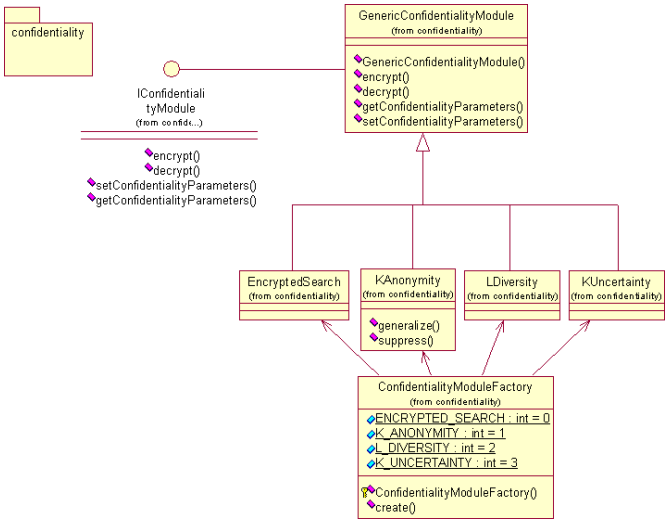


**Figure 7.** `marf.security.confidentiality` **Package and Classes.**

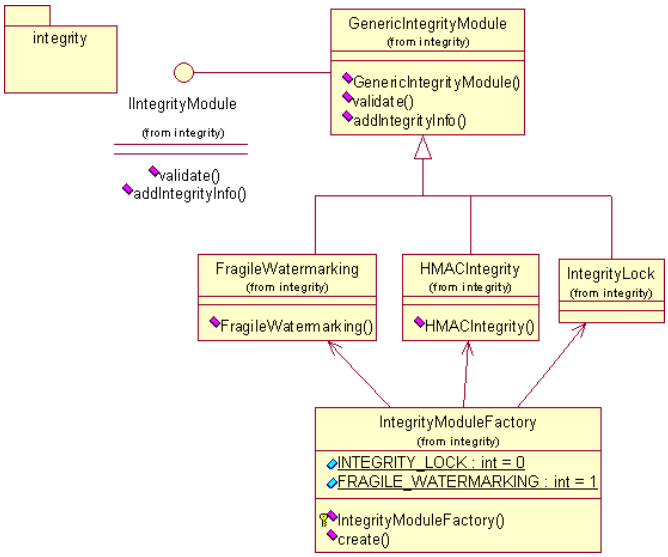


**Figure 8.** `marf.security.integrity` **Package and Classes.**

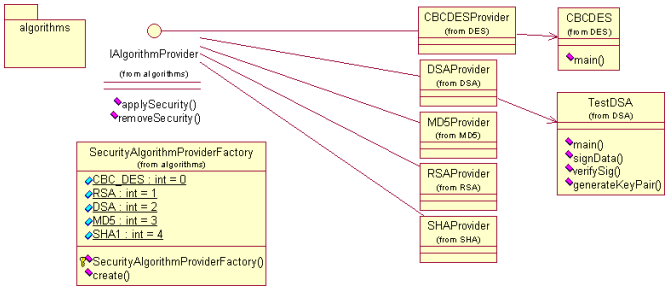


**Figure 9.** `marf.security.algorithms` **Package and Classes.**

ability, and authentication based on some of the problems outlined earlier in the description of the two systems. The aspects of high-availability, malicious code detection and certified compilation among others are treated as subtopics of the one of the primary four aspects as appropriate. Authorization and access control are not covered in this work because they are handled by the host operating system software and system tools outside of the scope of the studied systems. Importance and relevance of every computing aspect in each studied systems is highlighted.

### 2.1. Confidentiality

In general, the confidentiality aspect is of lesser concern in both systems as they care more about correctness and accuracy of the computation and stored results and their availability. There may, however, be a valid need to conceal the data in the system performing the computation for some applications that have the requirement (e.g. identities of speakers in a speaker identification application or details of a distributed cyberforensic investigation [13, 26]); thus, the confidentiality aspect is more pertinent to the application side that may run on top of the systems. In GIPSY, the confidentiality requirement is generally less applicable than in DMARF except the aforementioned cyberforensic case. In DMARF, depending on the application front-end, confidentiality, and more specifically privacy may be more desired when some subject's identity data is not required to produce results such as accuracy or performance statistics. An example of such an application is one of the statistics gathered by the `SpeakerIdentApp` in order to measure the accuracy of a given combination and configuration of algorithms used in the pipeline. Application of the JDSF's confidentiality subframework would resolve the confidentiality concerns in the two systems. Its integration into the security layer would be minimal because all the systems share the common implementing language, Java. JDSF will be "injected" on the computing and generating nodes and will be invoked before the data leaves the system or when it arrives the system. The configuration assumes the key set up and exchange has already happened. The "injection" of the JDSF can be source code-based or proxy-based. The source code-based solution will enable better integration of the system with the security layer, more security as the the information leakage between the inner layers and the security layers minimized or entirely eliminated. However, it is a more tedious and work-intensive integration that also introduces a dependency on JDSF. Proxy-based integration implies certified proxies act like transport agents and implement at the same time agent's API in both GIPSY or DMARF, encrypt/anonymize (or do the inverse on the receiving end) data, and pass it on. The improves flexibility, key management through SNMPv3, removes dependency on JDSF, but introduces the problems of getting data securely between the proxies and the actual agents as well as maintenance of and dependence on the availability of the proxy services. While the confidentiality of data in transit between the agent and the proxy is relatively easily achievable via SSL, the protocol overhead may be more prohibitive, than implementing the security layer in the code base by redirecting the storage method calls through JDSF. Confidentiality of the management and control information can be achieved by the same means or implementing the SNMPv3 proxies. GIPSY does not presently have a facility to work with SQL-based databases; DMARF has an export API to produce an SQL output. With the untrusted SQL, the confidentiality of data can be compromised through SQL injections attacks; that can be prevented by SQL randomization, also designed in JDSF.

### 2.2. Integrity

Integrity is a paramount aspect for both systems as in no event they should produce incorrect results due to either mal-intently malformed packets or data structures that come from the network, be it demands in GIPSY or samples and results in DMARF. This is a stringent requirement as of to produce correct and accurate results of the computation; otherwise, any results from the systems at large will not be trustworthy and publishable in scientific research venues. The issue aggravates, when the incorrect results persist and accumulate, such as in GIPSY's warehouse or persistent cache. Some of the integrity is assured by the Java Virtual Machine upon serialization and class loading with the byte-code verifier, but it is not sufficient with the communicating distributed systems. Integrity of the data in transit is achieved by the JDSF's integrity subframework for the similar reasons and mechanisms as in the confidentiality aspect described in the earlier section. Integrity of the SQL-based data can also be maintained by randomizing the SQL statements. Both, JDSF source code or certification proxy approaches are applicable in the integrity case. In fact, the proxy certificates (e.g. SSL) are better suited in the integrity aspect than in confidentiality requirement solution as it is easier to deploy and maintain the separation as no confidentiality protocol is required when communicating the data between the agent and the proxy. The integrity of a GIPSY node can be abused by the presence of the injected malicious code through the `embed()` call or intentionally written malware GIPSY program code with the attacker hoping the code to propagate through the GIPSY network of workers "infecting" them. There is a number of things that the attacker can do. The simplest is the DoS of the GIPSY network itself by supplying a procedural demand that contains an infinite loop and the workers at present are not designed to timeout and the timeout interval is impossible to uni-

versally define. The more serious incursions may include DDoS against other hosts, worms, and spread of any known malware, that may not be easily detectable by the host OS tools and services (e.g. antiviruses or IDSes) if the arriving and leaving demand payload is encrypted. These are difficult issues to address and at present JDSF is nowhere close to have means to provide some solutions to these. The solutions here would involve static and dynamic procedural code analysis, model checking, and certified compilation. The latter would sign the code at the trusted generator to at least disallow unsigned embedded code execution; it will not prevent, however, intentionally crafted GIPSY program, as it will leave certified from the attackers computer. This is where the additional model checking will have to take place, and internal intrusion detection engine has to be made. In the short run, we can apply a restriction and filtering of non-local `embed()` calls and implementation of the proxied `embed()` with certification support and timeouts.

### 2.3. Authentication

The data origin authentication aspect is crucial in both systems as well. It relates to the integrity described above and correctness of a scientific computation as if the data coming from an untrusted source may seem not to have violated integrity checks, but may still be intentionally incorrect thereby poisoning cached results in the data store in either system and accumulating error over time. Authentication also has to do with code signing in the procedural demands that deliver code payload to assure it comes from a trusted source and either signed by one of our node's compilers or external authority. This in part relates to the follow up section on availability. The JDSF's authentication subframework can be used here in part similarly in the way it is done for the confidentiality and and the integrity aspects described above. The authentication can be achieved similarly to the DNSsec methodology and can be hierarchical. We do not consider the authentication of the users here to prove their identities to the systems as there are no users and user management in the studied systems.

### 2.4. Availability

Generally, any reliable distributed system has to provide some redundancy in order to achieve constant high-availability of its services. In the studied systems the availability can be disrupted by the regular network problems connecting the participating nodes and the malicious code compromising the integrity of the nodes (we do not consider human administrators killing processes at the computing nodes related to our system as we can't deal with it easily). As described slightly earlier, malicious code detection for GIPSY is paramount to ensure availability of all the GIPSY services as avoid becoming a DDoS source to itself and others as well as a malware spreading mechanism. Availability has also to do with non-mal-intended code having infinite loops that have to be either aborted or disallowed as far as static code analysis allows. Availability is generally very difficult to guarantee in a distributed environment, and JDSF has no provisions for that as well. The ways to deal with the malicious code are also described earlier in the previous sections, and will require much further research and elaboration in the future work.

## 3. Conclusion

Security aspects, such as data origin authentication, data integrity, malicious code detection are important aspects even for scientific research and experimental distributed systems that are designed to run potentially over the uncontrolled public networks, such as the Internet to reduce the risks of cache poisoning of the data warehouses and stored data or incorrect results as well as DDoS. Thus, it is imperative the systems should allow for the security mechanisms to be easily added; it is also imperative that the mandatory security layer is not required, as to for performance reasons of a large scale scientific computation on a local controlled network or cluster it is an unnecessary overhead. The proposed solution of JDSF does cover a wide array of these aspects, but cannot be a all-in-one solution, and the availability aspect along the lines with the malicious code detection is the most difficult to tackle; thus, suggesting to look at other similar frameworks and enhance the secure coding practices engineered back into the studied systems.

**Future Work.** The future work will focus on the completion of the implementation of the security layer framework for both systems to make it possible to turn on the security layer with configuration parameters in both GIPSY and DMARF. The author of this paper is involved with the both projects as well as JDSF and is able to contribute the distributed security framework layer for both systems. A part of the related research, is the measuring of the overhead induced by the security layer as well as its ease and generality of adaptation to existing systems that had no security goals in mind when they were designed. A particular care will be given to the design and implementation of efficient model checkers and certified compilation aspects for the studied systems.

**Acknowledgments.** We would like to acknowledge the people presently and currently involved with the GIPSY and MARF projects, as well as JDSF. Namely, Joey Paquet, Emil Vassev, Amir Pourteymour, Xin Tong, Stephen Sinclair, Ian Clement, Dimitrious Nicolacopoulos, Linguy

Wang, Mourad Debbabi, Chadi Assi, Lee Wei Huynh, Jian Li, Farid Rassai. This research work was funded by the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada.

## References


[1] E. A. Ashcroft and W. W. Wadge. Lucid - A Formal System for Writing and Proving Programs. volume 5. SIAM J. Comput. no. 3, 1976.

[2] E. A. Ashcroft and W. W. Wadge. Erratum: Lucid - A Formal System for Writing and Proving Programs. volume 6(1):200. SIAM J. Comput., 1977.

[3] E. A. Ashcroft and W. W. Wadge. Lucid, a nonprocedural language with iteration. *Communication of the ACM*, 20(7):519–526, July 1977.

[4] P. Grogono, S. Mokhov, and J. Paquet. Towards JLucid, Lucid with embedded Java functions in the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 15–21. CSREA Press, June 2005.

[5] D. Harrington, R. Presuhn, and B. Wijnen. *RFC 2571: An Architecture for Describing SNMP Management Frameworks*. www.ietf.org, Apr. 1999. `http://www.ietf.org/rfc/rfc2571.txt`, viewed in January 2008.

[6] I. A. N. A. (IANA). *PRIVATE ENTERPRISE NUMBERS: SMI Network Management Private Enterprise Codes*. iana.org, Mar. 2007. `http://www.iana.org/assignments/enterprise-numbers`.

[7] R. Jagannathan and C. Dodd. GLU programmer's guide. Technical report, SRI International, Menlo Park, California, 1996.

[8] R. Jagannathan, C. Dodd, and I. Agi. GLU: A high-level system for granular data-parallel programming. In *Concurrency: Practice and Experience*, volume 1, pages 63–83, 1997.

[9] Jini Community. *Jini Network Technology*. Sun Microsystems, Inc., Sept. 2007. `http://java.sun.com/developer/products/jini/index.jsp`.

[10] B. Lu, P. Grogono, and J. Paquet. Distributed execution of multidimensional programming languages. In *Proceedings 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, volume 1, pages 284–289. International Association of Science and Technology for Development, Nov. 2003.

[11] Q. H. Mamoud. *Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms*. Sun Microsystems, Inc., July 2005. `http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/`.

[12] S. Mokhov. On design and implementation of distributed modular audio recognition framework: Requirements and specification design document. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Aug. 2006. Project Report. A copy is found: `http://marf.sf.net`, last viewed April 2008.

[13] S. Mokhov. Intensional Forensics – the Use of Intensional Logic in Cyberforensics. Technical report, Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, Jan. 2007. ENGR6991 Technical Report.

[14] S. Mokhov and J. Paquet. General imperative compiler framework within the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 36–42. CSREA Press, June 2005.

[15] S. Mokhov and J. Paquet. Objective Lucid – first step in object-oriented intensional programming in the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 22–28. CSREA Press, June 2005.

[16] S. A. Mokhov. Towards hybrid intensional programming with JLucid, Objective Lucid, and General Imperative Compiler Framework in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Oct. 2005. ISBN 0494102934.

[17] S. A. Mokhov. Introducing MARF: a modular audio recognition framework and its applications for scientific and software engineering research. In *Proceedings of the IEEE Engineering/Computing and Systems Research E-Conference (SCSS07/CISSE 2007)*, University of Bridgeport, U.S.A., Dec. 2007. Springer. To appear, `http://cisse2007.org`.

[18] S. A. Mokhov. Choosing best algorithm combinations for speech processing tasks in machine learning using MARF. In S. Bergler, editor, *Proceedings of the 21st Canadian AI'08*, pages 216–221, Windsor, Ontario, Canada, May 2008. Springer-Verlag, Berlin Heidelberg. LNAI 5032.

[19] S. A. Mokhov. Experimental results and statistics in the implementation of the modular audio recognition framework's API for text-independent speaker identification. In C. D. Zinn, H.-W. Chu, M. Savoie, J. Ferrer, and A. Munitic, editors, *Proceedings of the 6th International Conference on Computing, Communications and Control Technologies (CCCT'08)*, volume II, pages 267–272, Orlando, Florida, USA, June 2008. IIIS.

[20] S. A. Mokhov. Study of best algorithm combinations for speech processing tasks in machine learning using median vs. mean clusters in MARF. In B. C. Desai, editor, *Proceedings of C3S2E'08*, pages 29–43, Montreal, Quebec, Canada, May 2008. ACM and BytePress. ISBN 978-1-60558-101-9.

[21] S. A. Mokhov. Towards syntax and semantics of hierarchical contexts in multimedia processing applications using MARFL. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 1288–1294, Turku, Finland, July 2008. IEEE Computer Society.

[22] S. A. Mokhov, L. W. Huynh, and J. Li. Managing distributed MARF's nodes with SNMP. In *Proceedings of PDPTA'2008*, Las Vegas, USA, Aug. 2008. CSREA Press. To appear.

[23] S. A. Mokhov, L. W. Huynh, J. Li, and F. Rassai. A Java Data Security Framework (JDSF) for MARF and HSQLDB. Concordia Institute for Information Systems Engineering,

Concordia University, Montreal, Canada, Apr. 2007. Project Report. Hosted at http://marf.sf.net, last viewed April 2008.

[24] S. A. Mokhov, L. W. Huynh, J. Li, and F. Rassai. A privacy framework within the java data security framework (JDSF): Design refinement, implementation, and statistics. In N. Callaos, W. Lesso, C. D. Zinn, J. Baralt, J. Boukachour, C. White, T. Marwala, and F. V. Nelwamondo, editors, *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics (WM-SCI'08)*, volume V, pages 131–136, Orlando, Florida, USA, June 2008. IIIS.

[25] S. A. Mokhov and J. Paquet. Formally specifying and proving operational aspects of Forensic Lucid in Isabelle. Technical Report 2008-1-Ait Mohamed, Department of Electrical and Computer Engineering, Concordia University, Aug. 2008. In Theorem Proving in Higher Order Logics (TPHOLs2008): Emerging Trends Proceedings.

[26] S. A. Mokhov, J. Paquet, and M. Debbabi. Formally specifying operational semantics and language constructs of Forensic Lucid. In *Proceedings of IMF'08*, Mannheim, Germany, Sept. 2008. To appear.

[27] S. A. Mokhov, J. Paquet, and X. Tong. Hybrid intensional-imperative type system for intensional logic support in GIPSY. Unpublished, 2008.

[28] J. Paquet. *Scientific Intensional Programming*. PhD thesis, Department of Computer Science, Laval University, Sainte-Foy, Canada, 1999.

[29] J. Paquet. A multi-tier architecture for the distributed eductive execution of hybrid intensional programs. Submitted for publication at SAC'09, 2008.

[30] J. Paquet and P. Kropf. The GIPSY Architecture. In *Proceedings of Distributed Computing on the Web, Quebec City, Canada*, 2000.

[31] J. Paquet and A. H. Wu. GIPSY – A Platform for the Investigation on Intensional Programming Languages. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 8–14. CSREA Press, June 2005.

[32] A. H. Pourteymour, E. Vassev, and J. Paquet. Experimental Investigations in GIPSY Demand Migration Systems. Unpublished, 2007.

[33] Sun Microsystems. *Java IDL*. Sun Microsystems, Inc., 2004. http://java.sun.com/j2se/1.5.0/docs/guide/idl/index.html.

[34] Sun Microsystems. *The Java Web Services Tutorial (For Java Web Services Developer's Pack, v2.0)*. Sun Microsystems, Inc., Feb. 2006. http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html.

[35] Sun Microsystems. *Java Message Service (JMS)*. Sun Microsystems, Inc., Sept. 2007. http://java.sun.com/products/jms/.

[36] The GIPSY Research and Development Group. The General Intensional Programming System (GIPSY) project. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2002-2008. http://newton.cs.concordia.ca/~gipsy/, last viewed April 2008.

[37] E. Vassev and J. Paquet. A Generic Framework for Migrating Demands in the GIPSY's Demand-Driven Execution Engine. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 29–35. CSREA Press, June 2005.

[38] E. I. Vassev. General Architecture for Demand Migration in the GIPSY Demand-Driven Execution Engine. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, June 2005. ISBN 0494102969.

[39] A. Wollrath and J. Waldo. *Java RMI Tutorial*. Sun Microsystems, Inc., 1995-2005. http://java.sun.com/docs/books/tutorial/rmi/index.html.