Security Requirements Model for Grid Data Management Systems *

Syed Naqvi[1,2], Philippe Massonet[1], Alvaro Arenas[2]

[1]Centre of Excellence in Information and Communication Technologies (CETIC), Belgium
{syed.naqvi, philippe.massonet}@cetic.be

[2]CCLRC Rutherford Appleton Laboratory, United Kingdom
{s.naqvi, a.e.arenas}@rl.ac.uk

Abstract. In this paper, we present our ongoing work of a policy-driven approach to security requirements of grid data management systems (GDMS). We analyse the security functionalities of existing GDMS to determine their shortcomings that should be addressed in our work. We identify a comprehensive set of security requirements for GDMS followed by the presentation of our proposed Security Requirements Model. Derivation of security policies from security requirements and their consequent refinement is also presented in this paper. Our approach of addressing modelling issues by providing requirements for expressing security related quality of service is the key step to turn storage systems into knowledge representation systems.

Keywords: Grid security, requirements analysis, distributed data management.

1 Introduction

Grids enable access to, and the sharing of, geographically distributed heterogeneous resources such as computation, data and information sources, sensors and instruments, for solving large-scale or complex problems. One of the key Grid applications is the use of grids in emergency response. In this kind of applications, Grids become a critical information infrastructure providing essential information to emergency departments in order to minimise adverse impacts of potential tragedies. For instance, Grids may be useful in preventing floods, which can be achieved by integrating data from various sources - networks of sensors in a river basin, weather prediction centres, historical flood datasets, topography, population and land use data - for processing in sophisticated numerical flood models. The massive data sets that would need to be accessed and processed would require huge network facilities, data storage, and processing power to deliver accurate predictions. This paper focuses on one element of such critical infrastructure: Grid data management systems (GDMS).

** This research work is supported by the European Network of Excellence **CoreGRID** (project reference number 004265). The CoreGRID webpage is located at www.coregrid.net.*

We have carried out a formal analysis of security requirements for semantic grid services to explore how these requirements can be expressed as metadata associated to these services. It also explores issues of negotiation of the QoS parameters in order to reach Service Level Agreements (SLA). This work is being used to gridify the *FileStamp* distributed file system which is currently using the peer-to-peer technology for the exchange of data resources across the distributed sites. In this paper, we present a case study of *FileStamp* to explain security requirements model for GDMS.

This paper is organized in the following manner: an overview of the security functionalities of existing GDMS is given in section 2. *FileStamp* distributed file system is presented in section 3. Section 4 illustrates our proposed security requirements model. Our approach vis-à-vis the related work is discussed in section 5. Finally some conclusions are drawn in section 6 along with the outline of our future directions.

2 Overview of Security Functionalities in GDMS

Grid data management systems [1] offer a common view of storage resources distributed over several administrative domains. The storage resources may be not only disks, but also higher-level abstractions such as files, or even file systems or databases.

In this section, an overview of the security functionalities of various existing GDMS is presented:

2.1 ARMADA

Using the Armada framework [2], grid applications access remote data sets by sending data requests through a graph of distributed application objects. The graph is called an *armada* and the objects are called *ships*.

Armada provides authentication and authorization services through a security manager known as the *harbor master.* Before installing an untrusted ship on a harbor, the harbour master authenticates the client wishing to install the ship and authorizes use of the host resources based on the identity of the client and on the security policies set by the host.

The harbor master uses authentication mechanisms, provided by the host machine, to identify clients that wish to install ships on the harbor. The host provides mechanisms that implement security policies set by the host administrator. The options for implementing authentication include using SSH or using Kerberos authentication service.

The most common approaches used to protect system resources from untrusted code are hardware protection (e.g., running the untrusted code in a separate Unix process), software fault isolation (SFI) [3], verification of assembly code [4-5], and use of a type-safe language (e.g., Java or Modula3 [6]). Hardware protection requires untrusted code to run in a separate address space from the harbor. While this clearly protects the harbor from the client code, the overhead of communicating through nor-

mal IPC system calls is quite high. Both SFI and verification of assembly code offer promising solutions, but they typically target a limited set of machines, making them non-portable. Type-safe languages provide portability and memory protection for untrusted code: two important features for heterogeneous grid environments.

2.2 GridNFS

GridNFS [7] is a middleware solution that extends distributed file system technology and flexible identity management techniques to meet the needs of grid-based virtual organizations. The foundation for data sharing in GridNFS is NFS version 4 [8], the IETF standard for distributed file systems that is designed for security, extensibility, and high performance.

The challenges of authentication and authorization in GridNFS are met with X.509 credentials, which can bridge NFSv4 and the Globus Security Infrastructure, allowing GSI identity to be used in access control lists on files exported by GridNF servers.

The addition of data servers to the NFSv4 protocol does not require extra security mechanisms. The client uses the security protocol negotiated with a state server for all nodes. Servers communicate over RPCSEC_GSS, the secure RPC mandated for NFSv4. A failed state server can recover its runtime state by retrieving each part of the state from the data servers and the failure of a data server is not critical to system operation.

2.3 GFARM

The Gfarm file system [9] is a parallel file system, provided as a Grid service for peta-scale data-intensive computing on clusters of thousands of nodes. To execute user applications or access Gfarm files on the Grid, a user must be authenticated by the Gfarm system, or the Grid, basically by using the Grid Security Infrastructure [10] for mutual authentication and single sign-on. However, the problem here is that the Gfarm system may require thousands of authentications and authorizations from amongst thousands of parallel user processes, the Gfarm metadata servers, and the Gfarm file system daemons, thus incurring substantial execution overhead. To suppress this overhead, the Gfarm system provides several lightweight authentication methods when full Grid authentication is not required, such as within a trusted cluster.

2.4 GVFS

Grid Virtual File System (GVFS) [11] is a virtualized distributed file system for providing high-performance data access in grid environments and seamless integration with unmodified applications.

GVFS utilizes user level proxies to dynamically map between short-lived user identities allocated by middleware on behalf of a user. The data transfer in GVFS is on demand and transparent to the user. GVFS employs client-side proxy managed disk cache through user-level proxies that can be customized on a per-user or per-

application basis. For instance, cache size and write policy can be optimized according to the knowledge of a Grid application. A more concrete example is enabling file-based disk caching by meta-data handling and application-tailored knowledge to support heterogeneous disk caching. The proxy cache can be deployed in systems which do not have native kernel support for disk caching, e.g. Linux. Because the proxy behaves both as a server (receiving RPC calls) and a client (issuing RPC calls), it is possible to establish a virtual file system by forwarding along a chain of multiple proxies. Thus in addition to the server-side proxy (responsible for authenticating requests and mapping identities), another proxy can be started at the client-side to establish and manage disk caches. Furthermore, a series of proxies, with independent caches of different sizes, can be cascaded between client and server, supporting scalability to a multi-level cache hierarchy.

The GVFS literature does not provide details about the security functionalities. It only mentions the use of Secure Shell - Data access is forwarded by GVFS proxies via SSH tunnels.

3 *FileStamp*: A Distributed File System

In this section, we present a case study of *FileStamp* – an existing distributed file system. *FileStamp* is basically a peer-to-peer file sharing system which is in the process of gridification at present.

The exponential growth in the scale of distributed data management systems and corresponding increase in the amount of data being handled by these systems require efficient management of files by maintaining consistency, ensuring security, fault tolerance and good performance in terms of availability and security. Read only systems such as CFS [12] are much easier to design as the time interval between meta-data updates is expected to be relatively high. This allows the extensive use of caching, since cached data is either seldom invalidated or kept until its expiry. Security in a read-only system is also quite simple to implement. Digitally signing a single root block with the administrator's private key and using one-way has functions allow clients to verify the integrity and authenticity of all file system data. Finally, consistency is hardly a problem as only a single user, the administrator, can modify the file system.

Multi-writer file systems face a number of operational issues not found in the read only systems. These issues include maintaining consistency between replicas, enforcing access control, guaranteeing that update requests are authenticated and correctly processed, and dealing with conflicting updates.

FileStamp is a distributed file system. It is developed to find a solution to the problems encountered in multi-writer file systems. It is a highly scalable, completely decentralized multi-writer peer-to-peer file system. The current version of the *FileStamp* is based on Pastis [13] architecture. It aims at making use of the aggregate storage capacity of hundreds of thousands of PCs connected to the Internet by means of a completely decentralized network. Replication allows persistent storage in spite

of a highly transient node population, while cryptographic techniques ensure the authenticity and integrity of file system data.

Routing and data storage are handled by the Pastry [14] routing protocol and the PAST [15] distributed hash table (DHT). The good locality properties of Pastry/PAST allow Pastis to minimize network access latencies, thus achieving a good level of performance when using a relaxed consistency model. In Pastis, for a file system update to be valid, the user must provide a certificate signed by the file owner which proves that he has write access to that file.

Pastis security features require considerably enhancement for the successful gridification of the *FileStamp*. These include the use of standard credentials for authentication (such as X.509 certificate); authorization scheme for policy the management and enforcement (such as CAS: Community Authorization Service [16]); encrypted movement of data between remote sites; and some dependable fault tolerance mechanism. These requirements are elaborated in [17]. However, to understand the proposed fault-tolerance mechanism, consider a grid storage system shown in figure 1a. Data elements A and B are distributed over several resources. A_1 and A_2 are the subparts of A; B_1 and B_2 are the subparts of B. Figure 1b depicts a failure situation where a node is broken down. The resource broker will start searching resources that match the storage requirements and preferences of the stakeholders of the data elements. This is a critical phase as security negotiations and the matching of security parameters have to be resolved besides seeking the storage capacities and other performance parameters. The security assurances should be met before moving the data-set to a new node. Figure 1c shows that the resource broker didn't find a node that can host both A_1 and B_2 simultaneously (as was the case before the failure occurred) and hence it found two different nodes – one for A_1 and the other for B_2 – to maintain the same security level of these elements.

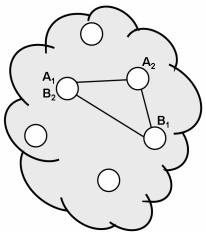

Fig 1a:
Distributed data elements

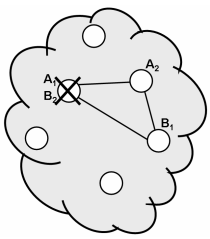

Fig 1b:
1 storage site is broken down

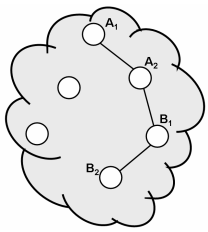

Fig 1c:
Redistribution of data-set

4 Security Requirements Model

This section presents a concise security requirements model of distributed file systems. This model is built by using KAOS [18] requirements engineering tool Objec-

tiver [19]. This model is although not comprehensive yet it is used to illustrate the various components of the security requirements model.

4.1 Problem Statement

We consider a simple problem statement so that more attention could be given to elaborate the various components of the security requirements model rather than indulging into the complexities of the model itself.

The problem addressed in this section is to assure fault tolerant and secure management of a distributed file system (*FileStamp*). Fault tolerance is attained by keeping an adequate number of replicas at different nodes; whereas the secure management is based on the encrypted transfer of files between the nodes. The various parameters involved in attaining in these two broad requirements are illustrated in this section.

4.2 Goal Model

Figure 2 depicts the overall goal model of the security requirements of a distributed file system. It illustrates that the main goal of the system is to assure that the files are always secure and available. This overall goal is refined with the sub-goals of availability and security. These sub-goals are further refined to describe the set of auxiliary

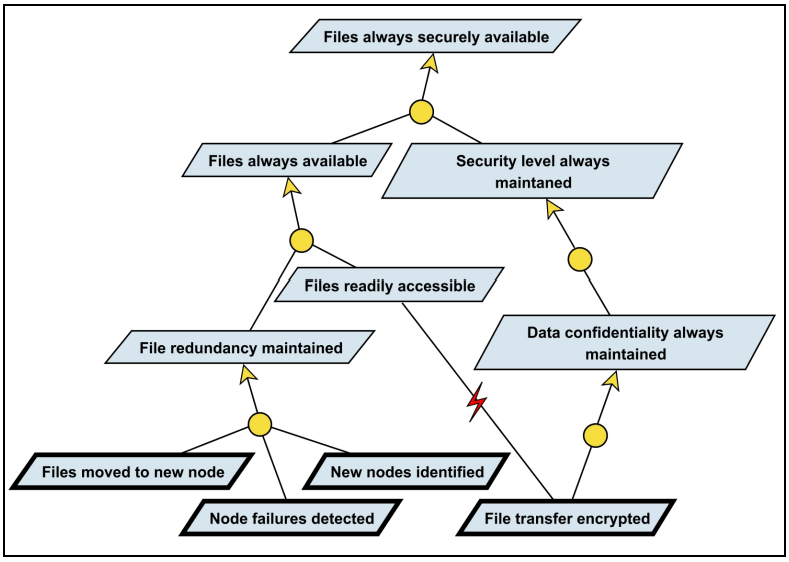

Figure 2: Goal Model

sub-goals needed to elaborate the upper level goals. Finally a set of requirements is associated with each refined sub-goal to demonstrate the prerequisite of attainment of

these goals. In figure 1, the goals and sub-goals are represented by thin-lined parallelograms whereas the requirements of the refined goals are represented by the thick-lined parallelograms. A goal model also includes the constraints of attaining certain goals. For example, in figure 1, the goal *files readily accessible* is constrained by the data confidentiality requirement of *encrypted file transfer*.

4.3 Responsibility Model

By definition, the responsibility model is derived from the goal model. A responsibility model contains all the responsibility diagrams. A responsibility diagram describes for each agent, the requirements and expectations that it is responsible for, or that have been assigned to it.

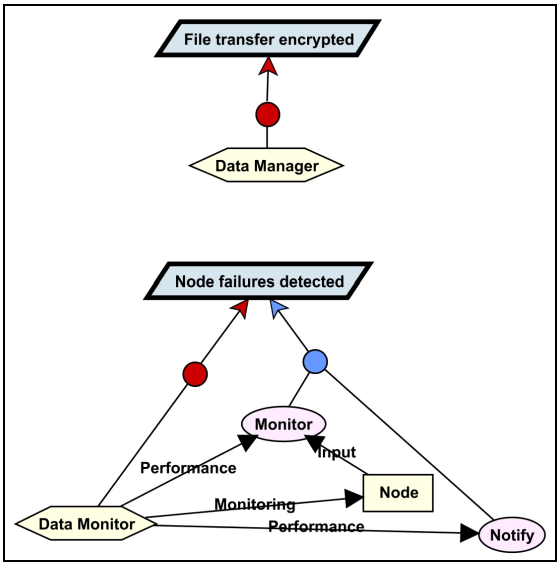

Figure 3: Responsibility Model

Figure 3 contains the responsibility diagrams of the problem statement considered in this section. It assigns the responsibility of the requirement *encrypted file transfer* to the *data manager*. Likewise, the responsibility of the requirement *node failures detected* is assigned to the *data monitor* that monitors the object *node* and employs the *monitor* and *notify* operations to keep an eye on the performance.

4.4 Object Model

The object model is used to define and document the concepts of the application domain that are relevant with respect to the known requirements and to provide static

constraints on the operational systems that will satisfy the requirements. The object model consists of objects pertaining to the stakeholders' domain and objects introduced to express requirements or constraints on the operational system. There are three types of entities that can be found in the object model: entities (independent passive objects); agents (independent active objects); and associations (dependent passive objects).

In figure 3, *node* is an object that is used as an input to the *monitor* operation. The *monitor* operation satisfies the requirement of node failure detection.

The object model is compliant with UML class diagrams as the entities correspond to UML classes; associations correspond to UML binary association links or n-array association classes. Inheritance is available to all types of objects including associations. Objects can be qualified with attributes.

4.5 Operation Model

The operation model describes all the behaviours that *agents* need to fulfil their requirements. Behaviours are expressed in terms of operations performed by agents.

Figure 4 shows the operation model of the problem statement considered in this section. The file transfer requirement (with or without encryption) requires an operation *move files*. Likewise the requirement of identifying new nodes requires an operation of *find available nodes*. Another example is the use of *monitor* and *notify* operations for the requirement of the detection of node failures.

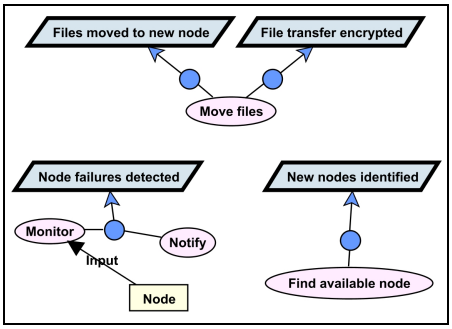

Figure 4: Operation Model

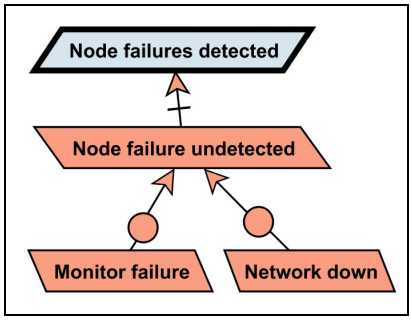

Figure 5: Obstacles Model

3.6 Dealing with Obstacles

Obstacles are the situations where a goal, a requirement or an expectation is violated. In such situation, the obstacle is said to *obstruct* the goal, requirement or expectation. Dealing with obstacles allows analysts to identify and address exceptional circumstances.

Figure 5 depicts the obstacles model of the problem statement considered in this

section. It shows that undetected node failures are the obstacles for the requirement of node failure detection.

5 Discussions

We have considered the *FileStamp* as a case study to interrelate security requirements and security policies for grid data management system. It is a part of our ongoing work of a policy-driven approach to security requirements. The implementation of this approach in the real systems requires formal derivation of security policies from the requirements model we have presented in the preceding section. These crude policies need to undergo refinement process so that operational policies can be obtained. These operational policies can be directly implemented to a system – GDMS in the context of our current work.

5.1 Derivation of Security Policies from Security Requirements

Security policies define the types of security measures that are used and what scope those measures have but not how those measures are designed or implemented. System security policies are derived from security requirements that specify the risks and threats that must be countered. These policies are system-specific and reflect the threat environment and the security problems assumed by system designers.

We need to derive implementable policy from the high level requirements model. This policy is then refined into operational policy. At the operational stage, it is ready to be implemented in the real systems.

5.2 Refinement of High-level Policies into Operational Policies

Policy refinement is the process of transforming a high-level, abstract policy specification into a low-level, concrete one. The objectives of a policy refinement process are identified in [20]. They are:

- Determination of the resources that are needed to satisfy the requirements of the policy.
- Translation of the high-level policies into operational policies that the system can enforce.
- Verification that the lower level policies actually meet the requirements specified by the high-level policy.

The first of these objectives involves mapping abstract entities defined as part of a high-level policy to concrete objects/devices that make up the underlying system. The second specifies the need to ensure that any policies derived by the refinement process be in terms of operations that are supported by the underlying system. The final objec-

tive requires that there be a process for incrementally decomposing abstract requirements into successively more concrete ones, ensuring that at each stage the decomposition is correct and consistent.

For the refinement of higher level policies into operational policies, we need a formal representation for objects, their behaviour and organisation; a technique for refining high-level goals into more concrete ones; and finally a means of inferring the combination of operations that will achieve the concrete goals. We intend to use the formalism presented in [21] to model the behaviour and organisation of the objects, together with the goal elaboration technique presented in [22] to refine high-level goals into concrete ones. However, the refined goals cannot be directly used in policies without first identifying the operations that will achieve them.

5.3 Related Work

Some techniques have been defined with the objective of taking security into account at requirement engineering. A main inspiration in our work is [23], which further extend KAOS for the specification and analysis of security requirements. The extended framework addresses malicious obstacles (called anti-goals) set up by attackers to threaten security goals. Threat trees are built systematically through anti-goal refinement until leaf nodes are derived that are either software vulnerabilities observable by the attacker or anti-requirements implementable by this attacker. Then, new security requirements are obtained as countermeasures. Massacci proposes in [24] extensions to the Tropos methodology –an agent-oriented software engineering methodology for modelling and analysis trust and security requirements. Central to their work is that is the assumption that in modelling security and trust it is necessary to distinguish between the actors that manipulate resources, accomplish goals or execute tasks, and actors that own the resource or the goals. They first develop a trust model, determining the trust relationship between actors, and then a functional model, where it is analysed the actual delegations against the trust model, checking whether an actor that offer a service is authorised to have it.

A first attempt to derive policies from high-level goals is presented by Bandara et al in [25]. Their main objective is to refine high-level policies -represented as a goal- into low-level operations that will allow a given system to achieve the desired goal. Their approach combines the KAOS requirement-engineering methodology, the Event Calculus, and abductive reasoning techniques. We have been inspired by their work, but taking an alternative approach. We use security requirements to derive the high-level policies, which can then be further refined using their approach. Close to Bandera's work is the work of Rubio-Loyola [26], which refines policies by applying requirement engineering and model checking techniques. His approach allows one to find system executions aimed at fulfilling low-level goals that logically entail high-level administrative guidelines. From system executions, policy information is abstracted and eventually encoded into a set of refined policies specified in Ponder. Above approaches have been applied to the networking management domain. Our interest is in applying them to the Grid area.

6 Conclusions

In this paper we have presented our work on modelling of security requirements of grid data management systems (GDMS). This work addresses issues related to storage management policies by modelling security requirements at the application level, and the requirements on mechanisms for using storage semantic web services. We have illustrated our proposed model with the help of a case study of the gridification of an existing distributed file system – *FileStamp*.

Our approach is a pioneer work towards the gridification of a grid data management system as most of the gridification efforts are limited to the fabric layer and the applications layer. Our long term objective is to transform storage systems into knowledge representation systems with suitable security features.

Our immediate future directions are the further elaboration and refinement of our model in the KAOS. Then we shall work on the derivation of security policy from the comprehensive and refined requirements model and eventually the refinement of the high level policy into operational policy for its implementation on a real grid data management system. We shall employ the Service Level Security Agreements to settle the conflicts.

References

1. Grid File System Working Group (GFS-WG) Information Document, *A Survey of the Major Grid File Systems*, Global Grid Forum (GGF) Eleventh Meeting (GGF11), Honolulu, Hawaii, USA, June 6-10, 2004
2. Oldfield, R., Kotz, D., Armada: A Parallel File System for Computational Grids, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid 2001 (CCGRID2001), 15-18 May 2001, pp 194-201
3. Wahbe R., Lucco S., Anderson T. E., and Graham S. L., *Efficient Software-based Fault Isolation*, Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, Ashville, NC, 1993. ACM Press. pp 203-216,
4. Morrisett G., Walker D., Crary K., and Glew N., *From System F to Typed Assembly Language*, Proceedings of the Twenty-Fifth ACM Symposium on Principles of Programming Languages, San Diego, CA, Jan. 1998
5. G Necula., *Proof-Crrying Code*, Proceedings of the Twenty-Fourth ACM Symposium on Principles of Programming Languages, pages 106-119, Paris, France, 1997.
6. Nelson G., *System Programming in Modula-3.* Prentice Hall, 1991
7. Honeyman P., Adamson W. A., McKee S., *GridNFS: Global Storage for Global Collaborations*, CITI Technical Report 05-3, 17 May 2005
8. S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, *Network File System (NFS) version 4 Protocol*, *RFC 3530*, 2003.
9. Tatebe O., Soda N., Morita Y., Matsuoka S., Sekiguchi S., *Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing*, Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04), Interlaken, Switzerland, September 2004.

10. Foster I., Kesselman C., Tsudik G., Tuecke S., *A Security Architecture for Computational Grids*, ACM Conference Proceedings 1998, ISBN 1-58113-007-4, pp 83-92
11. The Grid Virtual File System (GCFS) Project of University of Florida – http://www.acis.ufl.edu/~ming/gvfs/
12. Dabek F., Kaashoek M., Karger D., Morris R., and Stoica I., *Wide-Area Cooperative Storage with CFS*, In the proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), chateau Lake Louise, Banff, Canada, October 2001
13. INRIA Project PASTIS http://regal.lip6.fr/projects/pastis/pastis_fr.html
14. Rowstron A. and Druschel P., *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*, Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001, pp 329-350
15. Druschel P., and Rowstron A., *Past: Persistent and Anonymous Storage in a Peer-to-Peer Networking Environment*, Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)? 2001), pp. 65-70
16. Foster I., Kesselman C., Pearlman L., Tuecke S., and Welch V., *The Community Authorization Service: Status and Future*, In Proceedings of Computing in High Energy Physics 03 (CHEP '03), 2003
17. Naqvi S., Massonet P., Arenas A., Security Requirements Analysis for FileStamp Distributed File System, CoreGRID Technical Report # TR-0038, 2006
18. KAOS Project: www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html
19. Objectiver: The Requirements Engineering Tool – www.objectiver.com
20. Moffett J. and Sloman M., *Policy Hierarchies for Distributed Systems Management*, IEEE JSAC, vol. 11, pp. 1404-14, 1993.
21. Bandara A., Lupu E., and Russo A., *Using Event Calculus to Formalise Policy Specification and Analysis*, Proceedings of the 4th IEEE Workshop on Policies for Networks and Distributed Systems (Policy 2003), Lake Como, Italy, 2003.
22. Darimont R. and Lamsweerde A., *Formal Refinement Patterns for Goal-Driven Requirements Elaboration*, Proceedings of the *4th ACM Symposium on the Foundations of Software Engineering (FSE4)*, pp. 179-190, 1996.
23. Lamsweerde A., *Elaborating Security Requirements by Construction of Intentional Anti-Models*, Proceedings if ICSE 04, 26th International Conference on Software Engineering, ACM-IEEE, 148-157, 2004.
24. Giorgini P., Massacci F., Mylopoulos F., Zannone N., *Requirements Engineering Meets Trust Management: Model, Methodology and Reasoning*, In Proceedings of the Second International Conference on Trust Management. Lecture Notes in Computer Science, vol. 2995, 2004.
25. Bandara A., Lupu E., Moffett J., Russo A., *A Goal Based Approach to Policy Refinement*, Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004
26. Rubio-Loyola J., Serrat J., Charalambides M., Flegkas P., Pavlou G., Lafuente A., *Using Linear Temporal Model Checking for Goal-oriented Policy Refinement Frameworks*. Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005