Security Requirements Elaborations for Grid Data Management Systems

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

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

[2]STFC Rutherford Appleton Laboratory, United Kingdom
a.e.arenas@rl.ac.uk

Abstract

In this paper, we present a goal-oriented approach to design policies for managing security requirements of critical information infrastructures (CII). The approach, adapted from a standard and widely accepted requirements engineering methodology, is applied to the security analysis of a specific CII: Grid Data Management Systems (GDMS). Based on domain ontologies, combining concepts borrowed both from Virtual Organisation and Secure UML, a comprehensive set of GDMS security requirements is elicited and structured first semi-formally then formally. The benefits of the early formalisation are illustrated in term of completeness and consistency. A specific security analysis, using anti-goals, is also performed on the resulting formal security model to address the presence of malicious agents. Derivation of security policy from the set of security requirements is performed, first based on an abstract operationalization, still in the problem domain, and then it is translated into more concrete policy templates using standard policy constructs. Perspectives of the policy refinement and translation on existing policy languages and frameworks are described for its low-level implementation.

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

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 [Foster, 2001]. 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 [Hluchy, 2006]. 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). We have carried out a formal analysis of security requirements for semantic grid services to explore how these requirements can be used to derive security policy.

This paper takes a requirements engineering approach for the elaboration of such security requirements. Requirements Engineering (RE) is concerned with the identification of goals to be achieved by the envisioned system, the refinement of such goals and their operationalization into specifications of services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software. Goal-oriented RE refers to the use of goals for requirements elicitation, elaboration, organization, specification, analysis negotiation, documentation and evolution [van Lamsweerde, 2000]. More specifically the KAOS goal-oriented approach [Dardenne, 1993] is considered and applied to the security analysis of GDMS. KAOS supports a number of key features for the development of high quality security model such as: a security specific analysis (using anti-goals, modelling the attacker rationales), the ability to formalise critical parts with tool support for deep consistency and completeness checks [Ponsard 2007], and a dynamic framework for runtime requirements enforcement [Feather, 1998]. This paper demonstrates the benefits of such an early approach to close the gap between declarative security requirements and operational policies. It is achieved through mapping the requirements model onto high-level abstract policy by developing policy templates. The whole approach is illustrated

with the help of a complex data management system [Naqvi, 2006]. The focal point of this case study is ' availability' goals.

This paper is organized in the following manner: section 2 presents statement of the problem that is being used as a reference throughout this paper to elaborate the proposed approach. Use of Goal-oriented requirements engineering techniques for the specification of GDMS security requirements is elaborated in section 3. There is strong relation between security requirements and security policies. This relation is analysed in Section 4 by describing the derivation of security policy from the requirements model. We compare our approach with other known techniques in section 5. The paper is finally concluded with a concise account of future perspectives of this work.

2 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. 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 these two broad requirements are illustrated in this section.

2.1 GDMS as Critical Information Infrastructure (CII)

Grid computing is comparatively a new paradigm and therefore gauging its true potential still requires due exploration of the domain and its applications. Grid itself is already seen as a critical infrastructure for knowledge based economy where the digital data is used as raw materials [Steward 2005]. Applying Grid technologies for managing large chunks of data is the logical evolution of the conventional computing data management systems. Interestingly these nascent Grid-based data management systems possess the intrinsic characteristic of critical information infrastructures (CII). Critical information infrastructures (CII) are communications or information service whose availability, reliability and resilience are essential to the functioning of a modern economy, security, and other essential social values. Markets depend on them, as much as governments, to function properly [Cukier, 2005].

2.2 GDMS Security Requirements

Grid data management systems [Jagatheesan, 2003] offer a common view of storage resources distributed over several administrative domains. The storage resources may not only be disks, but also higher-level abstractions such as files, or even file systems or databases. The scale of the stored data and its distribution over different administrative domains across geopolitical frontiers exacerbate the overall security state of the entire systems as the presence of weakest link in the security chain may simply remains invisible due to the complexity and large size of the GDMS. It is impossible to extensively identify the security requirements of such systems by making simple observations. This situation obliges the security designers to rely on formal analysis techniques [Heitmeyer, 2001] to precisely identify the complete set of security requirements. Typical security requirements of GDMS cover the access restriction to authorized users, the protection against transmission in an insecure environment, the availability of files. In the following section, we present the goal-oriented analysis of those requirements.

3. Applying Goal-oriented Requirements Engineering for GDMS

This section presents a concise security requirements model of distributed file systems. This model is built by using the KAOS goal-oriented requirements engineering methodology [Dardenne, 1993]. KAOS is a generic methodology that is based on the capture, structuring and precise formulation of the system goals. A *goal* is prescriptive description of system properties, formulated in non operational terms. A *system* includes not only the design part but also its environment. Goals are refined and operationalized in a top-down manner as the system is designed or bottom up approach while re-engineering existing systems. Goals also help to structure the domain

vocabulary. In our case we do not start from scratch but will rely on existing ontologies such as Virtual Organisation (VO - for high level modelling of Grid systems) [Winton, 2005] and Secure UML (for security model) [Lodderstedt, 2002]. The approach also supports adverse environment, i.e. composed of possibly malicious external agents trying to break the system goal rather than to collaborate in the goal fulfilment, this approach has especially been applied to security requirements [van Lamsweerde, 2004]. As a Grid system is typically composed of a number of interacting nodes immersed in an open and possibly adverse environment, this approach really fits our needs.

A KAOS model is composed of a number of interrelated sub-models:

- The *object model* captures the ontology of the domain, i.e. the relevant vocabulary to express goals and anti-goals. In this part we reuse existing VO/security ontologies in order to be as standard as possible and enable some processing specific to those ontologies.
- The *goal model* captures and structures the assumed and required properties. In our case we will focus on security properties.
- The *agent model* takes care of assigning goal to agent in a realizable way. Discovering responsible agents is the criterion to stop a goal-refinement process.
- The *operation model* details, at state-transition level, the action an agent has to perform to reach the goals he is responsible for.
- The *anti-goal model* captures attacks on the system and how they are addressed. This model is built in parallel with the goal-model and helps discovering goals that will improve the robustness of the system, especially against malicious agent whose aims is to break the high level goals of the system.

In the rest of this section, we present and illustrate each model in turn, using excerpts of our running example. The anti-goal model is presented as the last model as such an analysis is done in a second phase of the system but note it triggers modification in the previous models to enhance the security of the system.

3.1 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. The object model consists of objects pertaining to the stakeholders' domain and objects introduced to express requirements on operational system. There are four concepts that are found in the object model: classical entities (independent passive objects); associations (dependent passive objects) found in entity-relationship models; more specialised concepts such as agents (independent active objects); and events (transient association existing in only one state). Inheritance is available to all types of objects including associations. Objects can be qualified with attributes.

A UML class diagram view is generally used to show the domain ontology. Such a diagram for the GDMS system is depicted in figure 1. The top of the figure depicts more general concepts borrowed for classical VO ontologies such as described in [Winton, 2005], the middle part of the figure describes a number of security related concepts borrowed from [Lodderstedt, 2002]. Those concepts are reused at the GDMS domain level through inheritance. This enables both to capture finer grained concepts and to stay compatible with standard ontologies and the associated processes (such as checks, model derivation, tools…). All the concept and attributes shown in the bottom part of figure 1 will be heavily used when formalising the security goals.

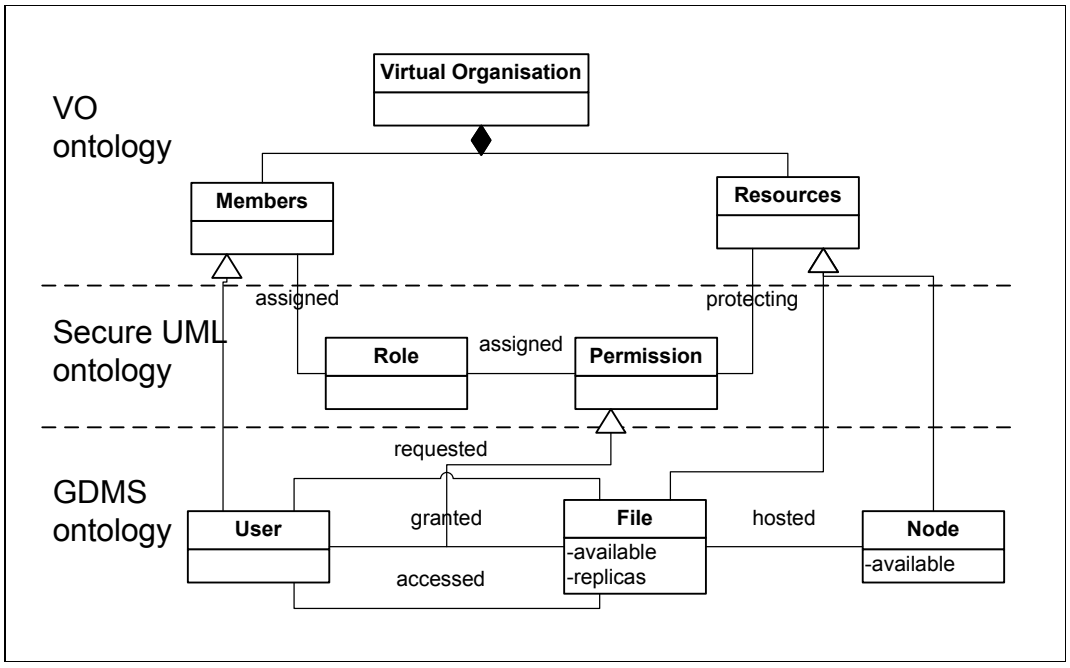

Figure 1: Excerpt of the GDMS object model.

3.2 Goal Model

Goals are organized in AND/OR refinement-abstraction hierarchies where higher-level goals are in general strategic, coarse-grained and involve multiple agents whereas lower-level goals are in general technical, fine-grained and involve fewer agents. In such structures, *AND-refinement* links relate a goal to a set of sub-goals (called refinement) possibly conjoined with *domain properties* or *environment assumptions*; this means that satisfying all subgoals in the refinement is a sufficient condition in the domain for satisfying the goal. *OR-refinement* links may relate a goal to a set of alternative refinements. Goal refinement ends when every sub-goal is realizable by some individual *agent* assigned to it, i.e. expressible in terms of conditions that are monitorable and controllable by the agent [Letier, 2002]. A *requirement* is a terminal goal under responsibility of an agent in the software-to-be; an *expectation* is a terminal goal under responsibility of an agent in the environment.

Figure 2 depicts a fragment of the goal model of the security requirements of a GDMS. It illustrates that the main goal of the system is to assure that the files are always secure and available. This overall goal *Maintain[FilesSecurelyAvailable]* is refined with the sub-goals *Maintain[FilesAvailable]* and *Maintain[SecureAccessToFiles]*. These sub-goals are further refined to describe the set of auxiliary sub-goals needed to elaborate the upper level goals. For example, a refinement goal *Maintain[FilesAvailable]* is through a goal *Maintain[FileRedundancy]*. Finally a set of requirements is associated with each refined sub-goal to demonstrate the prerequisite of attainment of these goals. For example, the *Maintain[FilesAvailable]* is refined into *Achieve[LowAvailabilityDetected], Achieve[NewNodeIdentified]* and *Achieve[FileReplicatedOnNewNode]*. As agents can be identified to enforce these goals, they are effective requirements and the refinement process stops here. However some further change might occur after formalisation (as some goals might have been overlooked) and the anti-goal analysis (as new goals may be introduced to improve the system robustness). At this informal level, we can already note the refinement follows a temporal pattern called milestone. This will greatly help in the formalisation process.

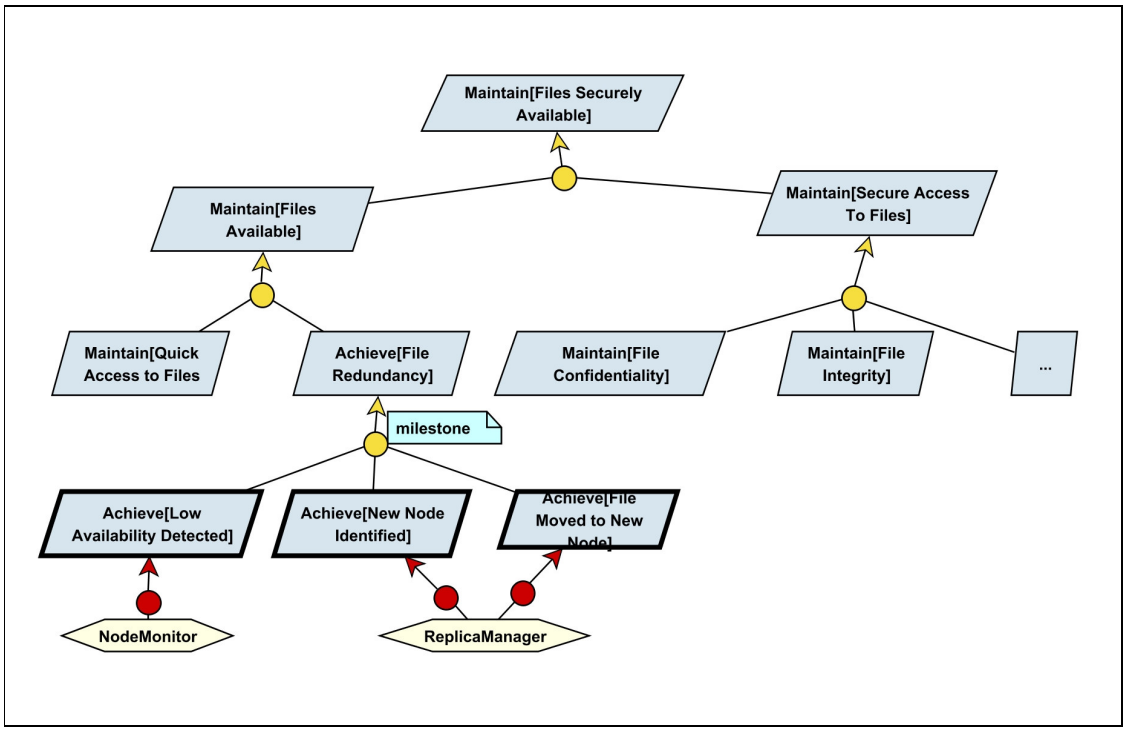

Figure 2: Excerpt of the GDMS Goal Model.

Goals prescribe intended behaviors; they are optionally formalized in a linear temporal logic [Manna 1992] with real-time extensions [Koymans 1992]. Keywords such as *Achieve, Avoid, Maintain* are used to name goals according to the temporal behavior pattern they prescribe. A temporal history is composed of an infinite linear sequence of state starting from an initial state. State are describe as classical first order logic predicates, with the classic logical operators: $\forall$, $\exists$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. The temporal operators used in this paper are the following:

- $\Box$ P, meaning the formula P is satisfied in all future states. It is used to express invariants
- $\Box_{\leq N}$ P, meaning the formula P is satisfied in the next N future states. It is a less constrained invariant.
- $\bigcirc$ P, meaning the formula P is satisfied in the next state. It typically represents an immediate achievement.
- $\Diamond$ P, meaning the formula P is satisfied in some future state. It is typically used to express progress obligation
- $\Diamond_{\leq N}$ P, is a constrained form of progress, meaning the formula P is satisfied within N future states
- P $\Rightarrow$ Q is a shortcut for $\Box$(P $\rightarrow$ Q). It is a very frequent conditional temporal pattern
- @P is a shortcut for $\neg$ P $\wedge$ $\bigcirc$ P. It represents a state transition, typically used to model some event.

Goal refinement has a formal semantics based on precise definition of completeness, consistency and minimality [Darimont, 1996]. Those are supported by formal tools such as the FAUST toolkit [Ponsard, 2006].

Based on those formal means, the goal [*SecureAccessToFiles*] can now be formalized as:

Goal Maintain[SecureAccessToFile]
InformalDef: a file requested by a user who has access right is granted access.
FormalDef: *($\forall$f:File, u:User) requested(u,f)* $\Rightarrow$ $\bigcirc$ *granted(u,f)*

To avoid conflict with the availability goal, the scope of this goal is only at grant level and not effective access. This part is formalized by the goal *Maintain[FileAvailable]*:

Goal Maintain[FileAvailable]

InformalDef: a file requested by a user should been accessed within the max access time specified for the system

FormalDef: $(\forall f{:}File, u{:}User)\ granted(u,f) \Rightarrow \Diamond_{\leq MAX}\ accessed(u,f)$

Note that this formalisation already takes into account the security dimension through the granted predicate. An initial formulation would have just used requesting but would have resulted in some conflict among those goals. This shows the importance of the formalization in the early discovery and resolution of conflicts which otherwise will inevitably result in subsequent design inconsistencies.

As the father goal is beyond the scope of formalisation, this refinement will not be checked. However further refinement have to comply with the formal semantics of refinement. As the process of proving the refinement is rather complex and the same goal refinement pattern are frequently encountered, a pattern library has also been defined. For example, the *Maintain[FileRedundancy]* goal is refined using the classical milestone pattern of the form: $P \Rightarrow \Diamond Q$ refined by $P \Rightarrow \Diamond R$ and $R \Rightarrow \Diamond Q$. The pattern is instantiated as follows:

Goal Maintain[FileRedundancy]

InformalDef: each goal should be replicated across the network a number of times N

FormalDef: $(\forall f{:}File)\ \Box(\ replicas(f) \geq N\ )$

Where $replicas(n) \equiv \{n{:}Node \mid hosted(f,n)\ and\ available(n)\}$

As the parent goal is too restrictive to fit the pattern, we decide to soften it under the current achieved form that tolerates temporary availability under the threshold. To be safer, we could envisage increasing that threshold.

Goal Achieve[WeakenedFileRedundancy]

InformalDef: each goal should be replicated across the network a number of time N

FormalDef: $(\forall f{:}File)\ (replicas(f) < N) \Rightarrow \Diamond_{\leq T}\ (replicas(f) \geq N)$

The leave goals are requirements under the responsibility of some agents and can be formalized as follows:

Requirement Achieve[LowAvailabilityDetected]

InformalDef: an availability failure event is triggered for all file below the availability threshold

FormalDef: $(\forall f{:}File)\ replicas(f) < N \Rightarrow \Diamond_{\leq T1}\ availabilityFailure(f)$

Resp: DataMonitor

Requirement Achieve[NewNodeIdentified]

InformalDef: a availability failure event triggers the identification of a new node on which to replicate the file

FormalDef: $(\forall f{:}File)\ availabilityFailure(f)$

$\Rightarrow \Diamond_{\leq T2}\ (\exists nn{:}Node)\ available(nn) \wedge \neg hosted(f,nn) \wedge newNodeIdentifed(f,nn)$

Resp: ReplicaManager

Requirement Achieve[FileReplicatedOnNewNode]

InformalDef: when a new node is identified, the file is replicated on this node within a given time bound

FormalDef: $(\forall f{:}File, nn{:}Node)\ newNodeIdentified(f, n, nn) \Rightarrow \Diamond_{\leq T3}\ hosted\ (f,nn)$

Resp: ReplicaManager

A milestone pattern with two intermediate steps is now apparent. An additional real-time constraint is also present: $T1+T2+T3 \leq T$. As the pattern is tuned, additional checks have to be made using formal tools such as the FAUST toolkit which can model-check complex refinements. In our case, it reveals a missing piece when several nodes fail simultaneously. To address this we can add an assumption about the maximal failed node accepted in the systems with respect to the system size and the duplication level.

Assumption[LimitedSimultaneousFailures]

FormalDef: #{n:Node| failure(n)} < #{n:Node} - N

In our refinement there are also obvious conflicts with resource usage goals. For example, the current design only increases copies and will also result in resource usage conflict. Resource optimization will result in a dual

goal refinement decreasing the number of copies. The number of replicas is also left open and can be used to optimize the design with respect with other goals and the simultaneous failure assumption.

3.3 Agent Model

Goal refinement ends when every sub-goal is realizable by some individual *agent* assigned to it, that is, expressible in terms of conditions that can be monitored and controlled by the agent. Hence, those agents are also usually shown in the goal model (see figure 1). Those agents are either human or automated (hardware or software).

In our design, the responsibility of the requirements *Achieve[LowAvailabilityDetected]* is assigned to the *DataMonitor,* while the requirements *Achieve[NewNodeIdentified]* and *Achieve[FileReplicatedOnNewNode]* are both assigned to the *ReplicaManager*. The DataMonitor agent can be formally described as follows

Agent DataMonitor
Monitors: File
Controls: failureDetected
Responsible of: *Achieve[NewNodeIdentified]* and *Achieve[FileReplicatedOnNewNode]*

When related to security goals, the monitor and control constructs are general KAOS constructs that can be mapped on the SecureUML meta-model using the following "gluing" meta-constraint. This kind of constructs ensure the seamless integration of our meta-models.

Meta-constraint MonitoringAllowed:
$\forall$ *dm: DataMonitor, n:Node; Permission (dm,node)* $\Leftrightarrow$ *Monitors(dm,node)*

3.4 Operation Model

Requirements are operationalized into specifications of operations to achieve them [Letier, 2002]. An *operation* is an input-output relation over objects; operation applications define state transitions along the behaviors prescribed by the goal model. In the specification of an operation, an important distinction is made between (descriptive) domain pre-/post-conditions and (prescriptive) pre-, post- and trigger conditions required for achieving some underlying goal(s):

- *a pair (domain precondition, domain postcondition)* captures the elementary state transitions defined by operation applications in the domain;
- *a required precondition* for some goal captures a permission to perform the operation only if the condition is true;
- *a required trigger condition* for some goal captures an obligation to perform the operation if the condition becomes true provided the domain precondition is true (to produce consistent operation models, a required trigger condition on an operation implicitly implies the conjunction of its required preconditions);
- *a required postcondition* defines some additional condition that any application of the operation must establish in order to achieve the corresponding goal.

Note that operation specifications are still abstract construct making no assumption on the real implementation at design level. These specifications can be statically developed in the component implementation of some agent or deployed as policies using relevant mechanisms provided by the supporting infrastructure. This second option is the one we will elaborate in the next section of this paper.

Figure 3 shows the operation model of the problem statement considered in this section. Each requirement is operationalized by a single corresponding operation. As the pattern is a milestone, those operations are also logically chained. This is graphically apparent: first DetectAvailability triggers an appropriate event when a file becomes unavailable, IdentifyNewNode then decides on which node(s) new replica(s) will be deployed, and finally ReplicateFile performs the replication.

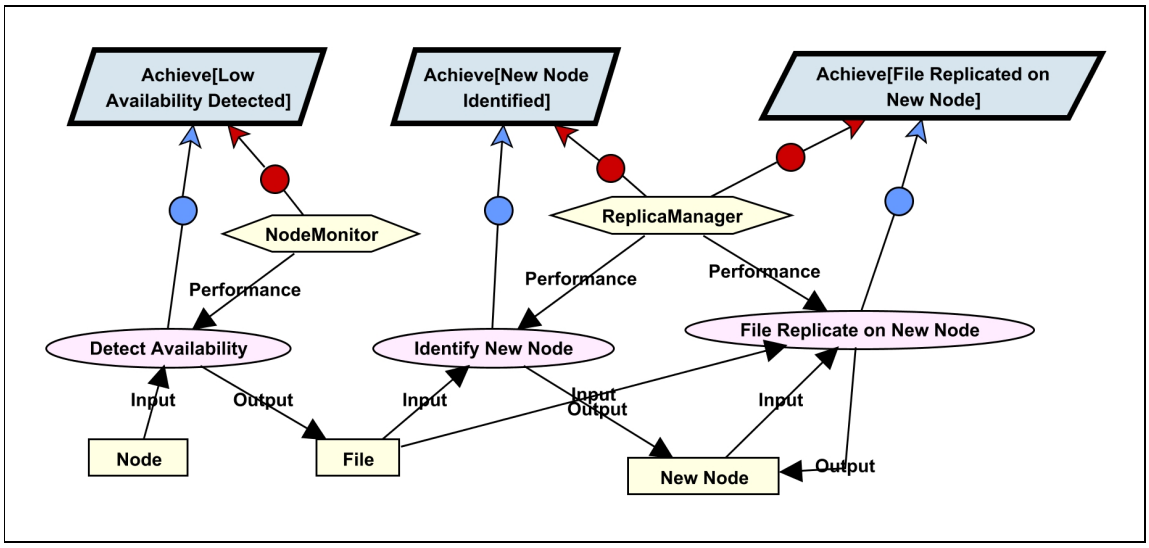

Figure 3: Excerpt of the GDMS Operation Model.

Operation can be formalized using first order logic to describe the various fields stated earlier: domain pre/post-conditions, and required trigger, pre and post conditions in order to achieve some goal. For example, the operation ReplicateFile can be formalized as:

Operation ReplicateFile
PerformedBy: ReplicaManager
Input : f: File, nn: Node
Output: nn:Node
DomPre : ¬hosted(f, nn)
DomPost : hosted(f, nn)
ReqTrig for Maintain[FileAvaibility] : (*replicas(f)* < *N*)

In this formulation, the domain operation is simply the replication of a file on a node where it is not yet present. To achieve the goal *Maintain[FileAvailability]* is enforced through the trigger condition which detects the low availability of the node.

3.5 Anti-Goal Model

In this section, we apply a combined obstacle/anti-goal analysis. The aim of this analysis is to be able to discover intentional or unintentional circumstances under which the system will fail to fulfil its goals. We call:

- *Obstacle (*to some goal): a condition whose satisfaction may prevent that goal from being achieved.
- *Anti-goal:* an intentional obstacle, i.e. an obstacle not occurring by (lack of) chance but deliberately followed by some malicious agent.

The analysis process is fully described in [van Lamsweerde, 2000] and [van Lamsweerde, 2004]. It can be briefly described as follows:

1. get initial anti-goals by negating the relevant security goals. In the scope of this paper, we will only consider the *Availability* goal.
2. elicit potential agent who might benefit from the anti-goal and check for rationale.
3. refine the anti-goal using AND-OR graphs until reaching realizable anti-goals, for example exploiting some design vulnerability.
4. address those by generating appropriate resolution counter-measures.

Figure 4 shows the analysis of our example: the Availability goal is first negated and three potential causes are

identified: some file could possibly not be reachable; reachable but not in time (as there is a time constraint in our formalization); or not available at all due to complete deletion. At this level, some problems might be intentional or otherwise. Refining further we can identify:

- *Lack of reachability* can be instigated by a file or network failure. Some agent might try to bring down some link but more likely problems could also be caused by a lack of infrastructure adjustment to resource requirements. New goals developed to address this issue are: *Achieve[EnoughReplicaManaged*] and *Maintain[RedundantNetworkLinks]*. Such kinds of goals are of course expected to be in conflict with cost; therefore optimization (possibly dynamic) will occur here.
- *Lack of timeliness* can be instigated by some infrastructural overload. This can be either caused by regular requests or irregular requests (e.g. Denial of Service attack). To address the unintentional failure, the infrastructure has to be managed in a way to accommodate its demands; for the intentional one, appropriate management of irregular requests have to be put in place, for example by identifying the originating user and cancelling or restricting his privileges of using the system.
- *Deletion of all copies* is a purely intentional goal as the system is designed to keep *N* copy in the system. To avoid such a situation, the delete operation and associated attributes should be managed very carefully.

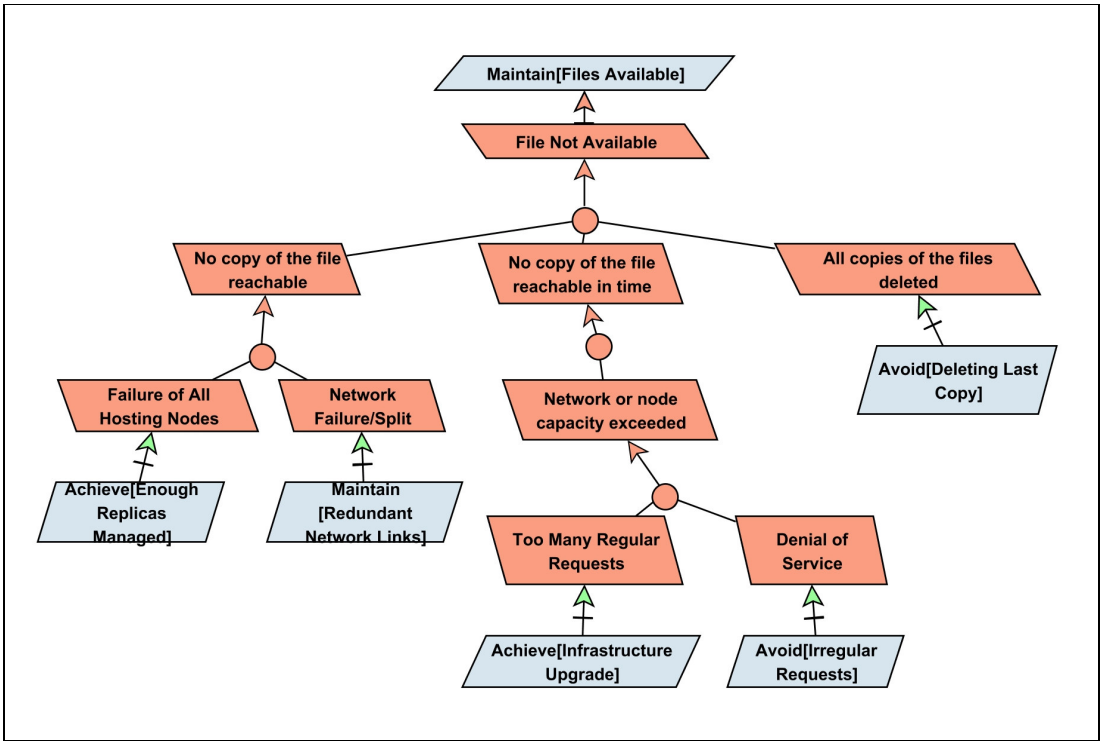

Figure 4: Anti-goal Model

Considering our original goal model of figure 2, we can consider now how to deploy those goals, for example some *InfrastructureManager* be assigned the responsibility of adapting the number of copies based on average and instantaneous number of replicas and the space left. Any increase in the number of copies should automatically be detected by the *DataMonitor* and result in a new replication by the *ReplicaManager*.

4. Derivation of Virtual Organisation Security Policy from the Requirements Model

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.

4.1 VO Policies

Following [Wasson 2003], we can define three types of policies in VOs. (1) *VO-wide operational policies*, which describe the state of the VO as a whole and not any single node or service. These policies are statements about the intended operational state of the VO, following mainly from *domain invariants*, properties know to hold in every state of some domain object. (2) *VO resource policies*, which describe the VO's rules for the behaviour of its member resources. These policies correspond to rules that pertain to particular resources within the VO rather than across the entire organisation. An example of this type of policies for the case of our system includes "*if a file Fi contains un-classified information then encryption is disabled*". This type of policies follows directly from the object model, where attributes (properties) associated to particular type of object is defined. (3) The last type of policies is *VO agent policies*, which specify rules for agents (users) of the VO's resources, from the perspective of the VO itself. These policies include permit/deny action on various possible agent operations, but more importantly they may specify pre-condition for any given operation, as well as obligations that the agent receives for performing an action. An example of this type of policies is in a M*oveFile* operation, the *Data Manager* agent must disable encryption if the file-transfer path includes trusted domains only". The main source of VO agent policies is the operation model, where pre/post conditions are defined for the operations performed by agents.

Considering our requirements model, the first category will rely on a number of domain properties identified as the goal model is elaborated: such properties will prove necessary to ensure the correct global operation of the system. The second and third category will be related to the requirements operationalisation step either from the target entities (for resource policies) or the enforcing agent point of view (for agent policies).

4.2 From Requirements to Policies

In this section we show how to derive security policy from the requirements model and define templates with attributes such as subjects, subject's responsibilities, objects, object's activities, etc. As in the requirement analysis, these templates will be populated both using natural language (for ease of comprehension) and mathematically (for its eventual use in developing precise semantics and in-dept formal processing). We will only focus on resource and policies.

The policy refinement procedure is the following:

1. Check the requirement that will be enforced dynamically through a policy process. This requires two conditions. First the operation should be controllable by the GDMS infrastructure, second the enforcement conditions (pre/post/trig) of the operation as to be monitorable by the infrastructure. Note that a candidate operation must not automatically be enforced dynamically even if this is seen as more flexible mode. This is a design decision to be taken according to the operational conditions. On the other hand, some actions can also be taken to address the lack of monitorability or controllability of some feature which could greatly benefit from a policy deployment.
2. Translate the enforcing operation into a policy using the informal template and the mapping described in section 4.3. The formalization will be considered in section 4.3 as a first refinement step.

4.3 Policy Structure

Requirements templates have been used in security requirements models to describe the various concepts both in natural language and formally [van Lamsweerde, 2001]. Here we are following a similar approach in which security policy templates are developed. Policy templates are designed to formulize policy statements. Every policy must have a core set of attributes and may have a set of optional/additional attributes. The proposed attributes are given in the following table along with the way to derive them from the requirements model. All the attributes support formalization except ID and Description.

	Attribute description	**Mapping with requirements**
ID	Policy identifier	ID can be derived from the enforced goal

Description	Explanation of the policy parameters	Description can be derived from the informal description of the enforced goal
Subject	Active entity that manages object(s) through a set of actions	Agent responsible of the enforced requirement
Object	Passive entity that is managed by subject(s) through a set of actions	Objects controlled by the responsible agent (operation output)
Action	Task to be executed by a subject on object(s)	Domain post-condition of the enforced operation
Authorization	Privileges given to the subject to perform actions on the object. Authorization maybe restricted by constraints	Security related required pre-condition of the enforced operation
Constraint	Conditions that need to be fulfilled before an action is initiated.	Constraint part of the required precondition of the enforced operation
Event	Condition that triggers the policy	Required trigger condition of the enforced operation

To illustrate this step, we can consider the goal Achieve[FileReplicatedOnNewNode]. At the informal level, the formulation is the following.

ID	NFRG-1
Description	*New replica of file is generated when an existing storage node is failed*
Subject	ReplicaManager
Object	Grid data storage nodes
Action	Replica generated
Authorization	Create files replica
Constraint	Availability of nodes
Event	Replica-host node failed

This is a structured representation of a policy element; however, it cannot be implemented in this form as it requires refinement and transformation into operational policies.

4.4 Towards the Refinement to Operational Policies

Policy NFRG can be further refined to provide more realistic security policy. To support this step, the requirements analysis greatly helps. First, it provides means to identify realizable operations, based on observable or controllable information. Second, through the formalization process, a number or missing or conflictial conditions are discovered, avoiding the design of erroneous policies. Applying this to our previous template yield the following template:

ID	NFRG-2
Description	*When the number of available file replicas becomes less than the threshold number, the monitoring agent will generate new replica by negotiating the security compatibility of the nodes with the file security requirements.*
Subject	ReplicationManager
Object	File to replicate and backup/unused Grid data storage nodes **FormalDef**: *f: File; nn:Node |available(nn)*
Action	Replica file generated on the compatible nodes **FormalDef**: *hosted(f,nn)*
Authorization	Locate compatible storage nodes and create files replica **FormalDef**: *Autorized(replicator, nn, WRITE)*

Constraint	Availability of compatible nodes **FormalDef**: *Available(nn) and not hosted(nn,f)*
Event	Number of available replicas becomes less than threshold value. **FormalDef:** *Replicas(f) < N*

The refined policies provide more details, including some object (file replicas), attributes (number of available replicas), and agents (Monitoring agent). However, they still lack the description of exact techniques and technologies to be employed in the real systems, typically relying on standard protocols (such as WS-agreement, WS-federation) and specific deployment languages. These details are provided in the implementation policies. Implementation policies are specific to a particular system and cannot be directly applied to other systems.

Unlike generic policies, which are quite abstract, operational policies contain specific details of a particular system. An example operational policy derived from the same example policy could be as:

> *When the number of available replicas of Cosmic_Gravitational_Waves.xls file becomes less than ninety percent of the total number of replicas over the LCS gird, the Grid Data Monitoring Tool will generate new replica by negotiating the security compatibility of the nodes with the security requirements of Cosmic_Gravitational_Waves.xls file by using the Web-Service Agreement protocol.*

Now the operational policy may look like:

ID	NFRG-3
Description	NFRG: New File Replica Generation policy is to be implemented in the *Laser Interferometer Gravitational-Wave Observatory (LIGO)* environment as part of *LIGO Scientific Collaboration (LSC)* Grid
Subject	Grid-Data Monitoring Tool (DMT)
Object	LSC Grid nodes
Action	Replica of file *Cosmic_Gravitational_Waves.xls* generated
Authorization	DMT can employ *Web-Services Agreement (WSA)* protocol to negotiate the security parameters and evaluate the compatibility of the node where replica is to be generated
Constraint	Availability of the nodes that correspond to the storage and security requirements of *Cosmic_Gravitational_Waves.xls* file
Event	Number of available replica-host nodes becomes less than 90% of the total number of replicas.

This shows how the policies could be derived from the requirements model and then refined and translated into the operational policies. A comprehensive set of such policies for the example case of LSC will also consist of secure transfer of file contents to the newer nodes. An operational policy for such data transfer will provide low-level details such as encryption algorithm to be employed, key-length to be maintained etc.

5 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 [van Lamsweerde, 2004]. Tropos [Mouratidis 2006] is a similar methodology for modelling organizations in terms of actors, goals, and dependencies to early requirements and provides a basis for extending early requirements to late requirements, architectural design, and detailed design. Tropos emphasizes the need to identify organizational concerns, separate them from implementation concerns, and give them first-class treatment. Toward this end, Tropos posits five main classes of concern: actors, resources, goals, soft goals, and tasks. A particular requirements model will contain multiple instances of each of these classes. Properties are not represented directly in the Tropos schema but may be captured in hard or soft goals. Tropos incorporate several types of concern relationship, including decomposition, means-ends, and dependency relationships. In some contrast to Tropos, the KAOS approach adopts a more explicitly multidimensional perspective

on requirements. Conceptually, requirements in the abstract and elements in a model can be associated with different aspects. Goals are subject to disjunctive and conjunctive refinement. Multiple views of the requirements are supported including refinement, operationalisation, entity-relationship, and agent. It also takes a view of concerns that is appropriate to requirements and separated from implementation and provides downstream continuity, from requirements to architectural refinement.

As for KAOS, security extensions have been proposed to Tropos [Giorgini, 2004]. 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 in [Bandara, 2004]. 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 could be further refined using their approach. Close to Bandera's work is the work of [Rubio-Loyola, 2005], which refines policies by applying requirement engineering and model checking techniques based on a temporal logic formalisation similar to the one used in this paper. His approach allows one to find system executions aimed at fulfilling low-level goals that logically entail high-level strategic 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. An important distinction here is that the approach described in [Rubio-Loyola, 2005] yields instances level policies whose generation is difficult to control and that have then to be generalised carefully. Our approach is more constructive and stays at the predicate level all the way: the RE model is directly mapped onto policies. The FAUST model-checking tool is applied to correct the model and not to generate policies directly.

6 Conclusions and Perspectives

This paper has presented our work of modelling security requirements of critical information infrastructures (CII). We considered - Grid Data Management Systems (GDMS) - an emerging example of CII as a reference system in this paper. This work addresses issues related to storage management policies by modelling security requirements at the application level.

Our approach is a pioneer work towards the formalisation of CII security requirements modelling. We showed how goal-oriented requirements engineering techniques can be deployed in the GDMS field to provide a solid framework for the elaboration of a high quality formal security requirements models from which system's security policy can be derived. Such an approach also provides strong arguments for security rationale [Common Criteria 2006] that yields security guarantees for the various stakeholders. These security assurances play vital role in the service level agreements (SLA) in the relevant businesses.

Our future directions include formal derivation of security policy followed by the conception of formal models for policy refinement to operational policies.

Acknowledgements

This research work is supported by European Commission funded projects CoreGRID (project reference number 004265) and GridTrust (project reference number 033817). Details of these projects are available at www.coregrid.net and www.gridtrust.eu respectively.

References

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

Common Criteria Evaluation and Validation Scheme, version 3.1, http://www.commoncriteriaportal.org, 2006

Cukier K, Mayer-Schoenberger V., Branscomb L., Ensuring (and Insuring?) Critical Information Infrastructure Protection, Report of the 2005 RUESCHLIKON Conference on Information Policy, 2005

Dardenne A., Lamsweerde A. and Fickas S., Goal-Directed Requirements Acquisition, Science of Computer Programming Vol. 20, North Holland, 1993, pp. 3-50.

Darimont R., van Lamsweerde A., *Formal Refinement Patterns for Goal-Driven Requirements Elaboration,* Proceedings FSE-4 - 4th ACM Symp. on the Foundations of Software Engineering, San Francisco, Oct. 1996.

Feather M.S., Fickas S., van Lamsweerde A., Ponsard C., R*econciling System Requirements and Runtime Behaviour* Proceedings de IWSSD'98 - 9th International Workshop on Software Specification and Design, IEEE, Isobe, Japan, April 1998.

Foster I., Kesselman C., Tuecke S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications*, 15(*3), 2001.

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.

Heitmeyer C, Applying `Practical' Formal Methods to the Specification and Analysis of Security Properties, in Proc. Information Assurance in Computer Networks (MMM-ACNS 2001), LCNS 2052, Springer-Verlag, St. Petersburg, Russia, May 21-23, 2001

Hluchy, L. Habala, O. Maliska, M. Simo, B. Tran, V. Astalos, J. Babik, M., Grid Based Flood Prediction Virtual Organization, IEEE International Conference on e-Science and Grid Computing, 2006. e-Science '06, Amsterdam, The Netherlands, December 2006

Jagatheesan A., Moore R., Watson P., Paton N., Grid Data Management Systems & Services, 29th International Conference on Very Large Databases (VLDB 2003), September 11, 2003, Berlin, Germany

Koymans R., Specifying Message Passing and Time-critical Systems with Temporal Logic, LNCS 651, Springer-Verlag, 1992.

Letier E., van Lamsweerde A., Deriving Operational Software Specifications from System Goals, In: Proceedings de FSE'10: 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Charleston, November 2002

Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In UML 2002

Manna Z. and Pnueli A., The Temporal Logic of Reactive and Concurren Systems, Springer-Verlag, 1992

Mouratidis H. and Giorgini P.. Secure Tropos: Dealing effectively with Security Requirements in the development of Multiagent Systems, in Safety and Security in Multiagent Systems – LNCS, Springer-Verlag, forthcoming, 2006

Naqvi S., Poitou O., Massonet P., Arenas A., Security Requirements Analysis for Large-scale Distributed File Systems, CoreGRID Workshop on Grid Middleware, Dresden (Germany), August 28-29, 2006.

Ponsard C., The FAUST Project: Formal Analysis by Using Specification Tools - http://faust.cetic.be, 2006

Ponsard C., Massonet P., Molderez J.F., Rifaut A., van Lamsweerde, Tran Van Hung, Early Verification and Validation of Mission Critical Systems, in Journal of Formal Methods in System Design (Springer), vol 30, nr 3, June 2007

Rubio-Loyola J., Serrat J., Charalambides M., Flegkas P., Pavlou G., Lafuente A., Using Linear Temporal Model Check-ing for Goal-oriented Policy Refinement Frameworks. Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005

Steward, Interview of Walter Stewart, in Grid Today Magazine, 21 March 2005

van Lamsweerde A, Letier E, Handling Obstacles in Goal-Oriented Requirements Engineering IEEE Transactions on Software Engineering, Special Issue on Exception Handling, Vol. 26 No. 10, October 2000, 978-1005.

van Lamsweerde Axel, Goal-Oriented Requirements Engineering: A Guided Tour, In: Proceedings of RE'01 - 5th IEEE International Symposium on Requirements Engineering, Toronto, Août 2001, IEEE Press, 2001, p. 249-263.

van Lamsweerde A., Elaborating Security Requirements by Construction of Intentional Anti-Models, Proceedings of ICSE'04, 26th International Conference on Software Engineering, Edinburgh, May. 2004, ACM-IEEE , pp 148-157.

Winton L.J., A Simple Virtual Organisation Model and Practical Implementation, in Proceedings of the Australasian workshop on Grid computing and e-research, vol44, Newcastle (Australia), 2005

Wasson G., and Humphrey M., Toward Explicit Policy Management for Virtual Organizations, 4th International IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 03), pp. 173-182.