Towards alignment of architectural domains in security policy specifications

Virginia N. L. Franqueira and Pascal van Eck

University of Twente
Department of Computer Science, Information Systems Group
P.O.Box 217, 7500 AE Enschede, The Netherlands
{franqueirav,p.a.t.vaneck}@ewi.utwente.nl

Abstract. Large organizations need to align the security architecture across three different domains: access control, network layout and physical infrastructure. Security policy specification formalisms are usually dedicated to only one or two of these domains. Consequently, more than one policy has to be maintained, leading to alignment problems. Approaches from the area of model-driven security enable creating graphical models that span all three domains, but these models do not scale well in real-world scenarios with hundreds of applications and thousands of user roles. In this paper, we demonstrate the feasibility of aligning all three domains in a single enforceable security policy expressed in a Prolog-based formalism by using the Law Governed Interaction (LGI) framework. Our approach alleviates the limitations of policy formalisms that are domain-specific while helping to reach scalability by automatic enforcement provided by LGI.

Keywords: Architectural domains, Alignment, Policy specification, Security, Law Governed Interaction (LGI).

1 Introduction

As organizations are becoming more and more complex, with more sophisticated information systems and information technology (IT), enterprise architecture is becoming proportionally more important as an instrument to deal with this complexity. Complexity in an organization has two sources: internal and external. Internally, organizations need to manage their daily operational activities while focusing on business development. Externally, organizations must comply with best practices and standards such as Cobit [1] for IT governance and control, ITIL [2] for IT delivery and support, and ISO 9000 [3] for quality management. In addition to this internal and external pressure, traditional business drivers like cost cutting, increasing customer satisfaction and time-to-market are also important drivers for managing enterprise architecture.

We consider architecture as "some logical construct for defining and controlling the interfaces and the integration of all of the components of the system",

as stated by Zachman [4]. All of these components, however, do not belong to a unique domain but instead belong to several architectural domains [4,5,6]. As a consequence, an integration problem arises because each domain involves (i) different elements, entities and concepts, (ii) different nature and concerns, and (iii) different stakeholders and perspectives. Therefore, a one-to-one mapping between architectural domains is not feasible and alternatively connecting architectural domains evolves into an alignment problem.

In this paper, we study this alignment problem for one specific aspect: IT security. The alignment problem in terms of security relates to the connection of three architectural domains. First, there is the access-control domain which involves aspects like people structured in terms of roles, objects, subjects and actions that subject can perform over objects. Second, there is the network domain which involves communication, usage and configuration of network resources. Finally, there is the physical domain which involves aspects like hardware, location and mobility. Figure 1 shows a representation of these three domains and relationships between them. It aims to provide an insight on the amount of interactions involved in aligning security domains, since the diagram looks already polluted with arrows while only a tiny number of entities is represented.

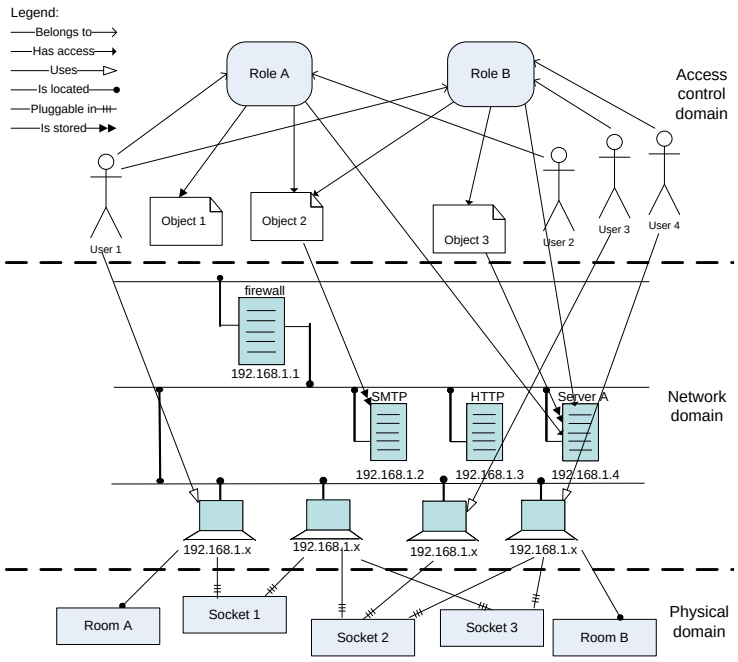

Fig. 1. A model: three domains involved in security and example interactions

Existing approached to the alignment problem in security can be categorized in two main streams. The first stream approaches alignment from the perspective of policy specification languages. These languages tend to be domain-specific because the scope of access control, network and also the physical domain is so broad and diverse. Consequently, for an integrated architecture, more than one language is needed. Therefore, the alignment is not really achieved unless a mapping process between these languages is used and, as a result, consistency between them can become an issue. The other stream relies on model-driven security [7]. It models the connections between all the elements from different domains with the support of graphical tools and then derives security policies through refinement between levels. One drawback of this approach is the complexity of the model in real scenarios and as a consequence, it becomes hard to analyze, considering the number of elements and connections involved.

This paper demonstrates the possibility to combine the three architectural domains that are relevant for security in a single policy. We do so by applying the Law Governed Interaction (LGI) framework [8] which provides means for the formal specification of policies in a semantically flexible way. Additionally, LGI allows cross-domain policies to be enforced in a distributed environment. Therefore, LGI enables architectural alignment without interfaces or connections between domains and thus avoiding drawbacks of current approaches.

We proceed presenting a motivating running example in Section 2, an overview of LGI in Section 3 and then the LGI implementation of the example in Section 4. In Section 5 we evaluate our approach through discuss while in Section 6 we review related work. Finally, in Section 7 we summarize our contribution.

2 Running example - overview and formalization

We use part of the IT infrastructure of a large Dutch government organization as a running example in this paper (see Figure 2[1]). Traditionally, this organization employs a number of mainframes for batch-oriented transaction processing (depicted on the right-hand side of Figure 2). Since a few years, in addition to these mainframes, the organization employs a modern infrastructure for Internet-based information exchange with outside parties such as other government organizations, companies and private citizens. With this new infrastructure, these outside parties (represented by a browser at the left-hand side of Figure 2) can directly use, via various electronic channels, the services provided by the government organization.

The nature of the business for which the government organization is responsible is such that the data maintained by the mainframes is highly confidential even for the employees of the government organization. Therefore, the infrastructure is organized in terms of protection perimeters which filters external, internal and maintenance accesses to the area inside the perimeters. The maintenance personnel is responsible for keeping the network infrastructure, hardware and

[1] Based on M.Sc. thesis by Allard Hoeve, University of Twente.

software up-and-running 24 hours a day. Their work is governed by policy F quoted below, which we formulated based on the policy description we found in the internal security manuals of the organization.

> Maintenance work shall be restricted to maintenance personnel, in accordance to job function, and limited to trusted hosts located on physically secured areas.

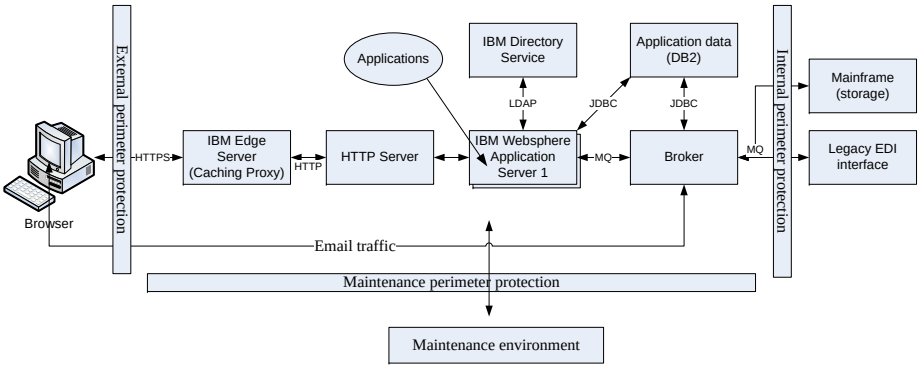

Fig. 2. Network diagram for a large Dutch government organization

We look at this scenario focusing on the aspect of accessing the inner perimeters area by maintenance personnel, i.e. crossing the maintenance and also the internal perimeters which are regulated by packet filter devices such as firewalls. The filtering process is based on fixed rules that allow or deny access to the delimited area by individual Internet Protocol (IP) addresses. In our case however, maintenance personnel should be able to overcome these rules and have these barriers automatically *open* depending upon the personnel's role and location to enable them to do maintenance work. The location here is represented by the combination of (i) room where the workstation to be used by the maintenance personnel is located and (ii) socket used to plug-in the workstation into the network. Although role and location are static during a maintenance session, on the next session a same employee can represent another role and use a different workstation from a pool, i.e. another location. As long as the employee provides a valid digital certificate authenticating his current role and workstation location and as long as an exception rule matches the employee's role, location and the server he wants to access, the filtering rule is bypassed and the access is granted in the basis of an exception to the rule.

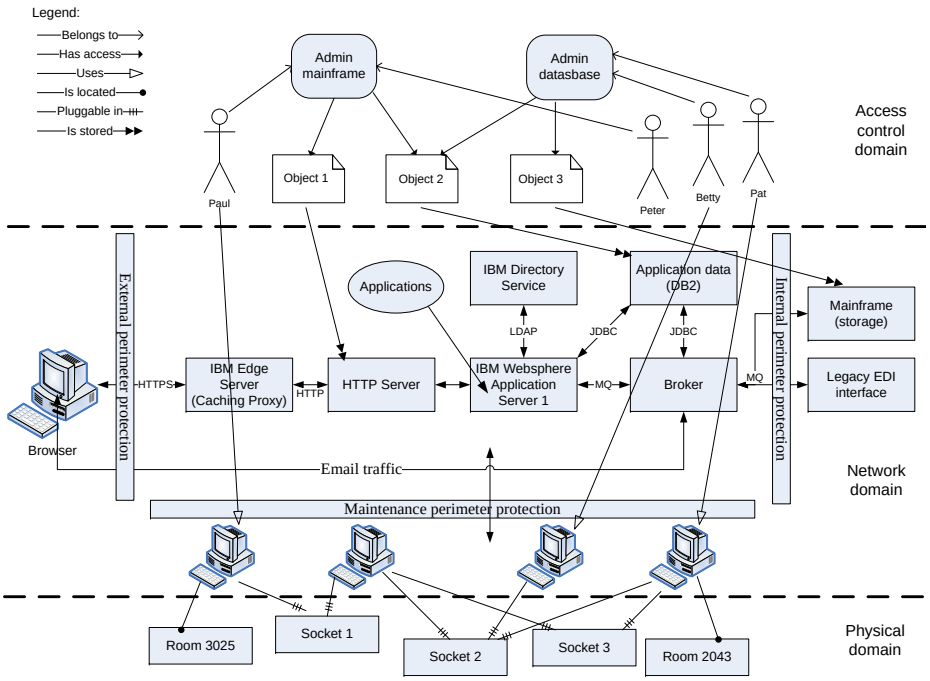

Fig. 3. The large Dutch government organization model: interactions between security domains

The enforcement of policy F comprehends entities belonging to different security domains, as described in Section 1. Figure 3 shows a model representation of entities relevant to our running example and their interactions. These entities are (i) employee, server and role from the access control domain (ii) TCP/IP(v4) packet from the network domain and (iii) location from the physical domain. These entities are formalized in a Prolog-like[2] notation as atoms. Note that the entities related to maintenance personnel and server to be serviced are implicitly represented inside the packet notation by source and destination IP (ips and ipd) and port (Ps and Pd). Thus, we restrict ourselves to the representation of role, packet and location.

```
role(RoleName).
packet(ips(IPs1,IPs2,IPs3,IPs4),Ps,ipd(IPd1,IPd2,IPd3,IPd4),Pd).
location(Room,Socket).
```

[2] The choice for Prolog notation is bound to LGI framework we use.

As an example, the atoms can be instantiated as follows.

```
role(adminDatabase).
packet(ips(130,89,148,26),11,ipd(192,168,1,157),1050).
location(zi4026,1).
```

Besides entities as atoms, we represent the filtering rules and exceptions to the rules as Prolog-like facts. Rules can be stated in terms of (i) wild card atoms *anyIP* and *anyPort*, (ii) atoms ips and ipd for IP source and IP destination, respectively, and (iii) atoms ipRangeD(_,_,_,rangeFrom,rangeTo) and portRangeD(rangeFrom,rangeTo) for ranges of IP and port destination, respectively. Exceptions can be stated in terms of roles, IP and port destination, location and a permission which opposes to a rule, either in the form of allow or deny. Note that these rules combine concepts from all three domains mentioned in Section 1.

```
ruleSet(anyIP,anyPort,ipRangeD(192,168,1,0,254),portRangeD(1023,16384),allow).
ruleSet(anyIP,anyPort,ipd(192,168,1,1),anyPort,deny).
ruleSet(anyIP,anyPort,ipd(192,168,1,2),25,allow).

ruleException(role(adminDatabase),ipd(192,168,1,1),1000,location(zi3067,2),allow).
```

In the next section we set the basis of LGI for the implementation of the running example.

3 Law Governed Interaction (LGI) - overview

Law-Governed Interaction (LGI) [8,9,10,11] is a coordination mechanism developed by Naftaly Minsky and colleagues at Rutgers University. LGI can be viewed as a decentralized architecture that offers middleware components for interaction between agents. These components actively monitor message exchange by checking compliance with a formal policy set, called the *law*. This policy takes the form of ECA-rules, which can be represented in either a Prolog-based or Java-based language. In this paper, we use the Prolog-based representation. It is important to note that Prolog rules in LGI should not be read as if-then rules. Instead, each Prolog rule in LGI represents an ECA pattern [12] of the form: e :- $c_1, ..., c_n, do(o_1), ..., do(o_n)$, where e is an event, $c_1, ..., c_n$ is a possibly empty list of conditions and $do(o_1), ..., do(o_n)$ is a list of operations (i.e. actions). Besides the concepts of event and operation, LGI also incorporates the concept of state (called Control State or CS) as a snapshot of an agent.

3.1 Local and distributed aspects of LGI

LGI makes available a set of regulated events and a set of primitive operations that are triggered by events. Figure 4 shows an example interaction between two

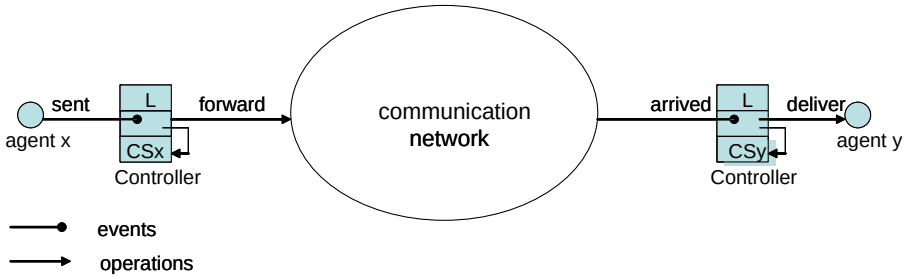

Fig. 4. LGI-agents interacting

computational nodes that LGI calls agents. Suppose agent x wants to send a message to agent y. This triggers a *sent* event at the local controller (a middleware component that is part of LGI). When an agent triggers a *sent* event containing a message, the controller uses *law* $\mathcal{L}_{\mathcal{F}}$ to assess whether agent x is allowed to contact agent y. If this is the case, it is forwarded by its controller to agent y. When the message arrives at the other agent's controller, an *arrived* event is triggered and its controller actually delivers the message if the *law* allows it. The *sent* and *arrived* events in the controller are interpreted such that the event is considered a Prolog goal. If the conditions are satisfied, this goal generates a to-do list with operations provided by the command $do(op)$, where op represents an operation. To check a condition, a controller can consult the so-called control state of the agent (CS_x and CS_y in Figure 4) by a process of unification between a given term in the format $t@CS$ and the content of its state. This interpretation of the *law*, called in LGI the *ruling* of the *law*, is therefore locally performed by individual controllers.

LGI controllers can enforce well-formed and available *laws* under the context of each agent's CS. They run as independent processes and thus can be placed in any host or server in the network. Agents engage themselves at their own discretion under a law which can regulate an entire community of so called LGI-agents. Besides, a single controller can be shared by several agents. LGI scalability is discussed elsewhere [13].

In summary, LGI provides a distributed enforcement of laws although their ruling is performed locally.

3.2 Flexibility of LGI

The control state of an agent can contain any Prolog term, including atoms, lists, and the combination of both. The semantic of these terms will be given by the *law* being enforced. The same happens with the message exchanged between agents since the semantic of its content is determined by the *law* under which the agents are operating. This characteristic allows LGI to be a very flexible coordination mechanism. Additionally, conditions under the ECA rules can be Prolog procedures which are goals that will be satisfied or not. These procedures

take advantage of the full power of a declarative language (Prolog) and enable sophisticated reasoning over the policy.

4 Running example - implementation in LGI

We implemented in LGI a firewall rule set subject to exceptions, as described in Section 2, compliant with policy F. Exceptions are based on the role and location of the user (i.e. maintenance employee) at the time he engages himself in the *law* $\mathcal{L}_{\mathcal{F}}$. For our running example scenario, we wrap maintenance utilities such as telnet into LGI messages. We discuss next some implementation details based on the two aspects of LGI emphasized in Section 3.

4.1 Local and distributed aspects of LGI

Figure 5 shows how LGI needs to be employed to control the network environment of the government organization. Example LGI controllers, highlighted in the diagram, enforce *law* $\mathcal{L}_{\mathcal{F}}$ which allows access by maintenance personnel to servers to be serviced based on exceptions to general firewall rules. LGI firewall agents placed on the perimeters are responsible for filtering the requests, in the format of packets, and decide if they should be (i) allowed to reach the destination, (ii) denied with notification sent to the sender and entry recorded in the firewall audit log or (iii) discarded with an audit log entry.

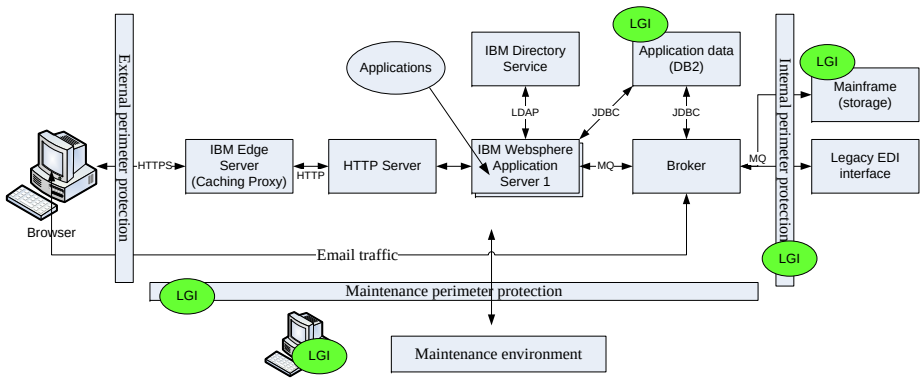

Fig. 5. LGI middleware components controlling network perimeters

4.2 Flexibility of LGI

In law $\mathcal{L}_{\mathcal{F}}$ we have two kinds of agents. First, the workstations of the maintenance personnel who provide their current role and location (i.e. room and socket) when engaging into the law. (The information is recorded in the agent CS when the agent adopts the *law*.) Second, the firewall and the servers which do not need to provide special information. Listing 1.1 shows an extract of law $\mathcal{L}_{\mathcal{F}}$. In this listing we use LGI's variant of Prolog. This variant provides a number of extensions to Prolog, only one of which is used in this paper: the $@CS$ operator. The meaning of it is as follows: a term $t@CS$ evaluates to true if t is present in the control state of the agent for which the rule is evaluated. The clauses follow the format of ECA rules what means that when the event is triggered and the conditions are met, the operations described by the $do(\_)$ atoms are executed.

By clause R2, only agents that have role and location on its CS are able to send messages (e.g. for telnet access to a server for maintenance). In this case, information related to the sender (i.e. role, room and socket he is using) and the destination are re-directed to the firewall.

When the message arrives at the firewall, exceptions are checked through the procedure *checkRuleExceptions*. Clauses R3 and R4 represent an if-the-else structure where R3 will evaluate to true if an exception applies, and therefore the permission will be retrieve from the exception fact, and R4 will evaluate to true if no exception applies, and the permission will be 'none'. Once more the list of information previously received, now attached with the permission and packet in case an exception applies, is forwarded to the firewall.

Clauses R5, R6 and R7 refer to the case where permission unifies with 'none'. Thus, the procedure *checkRuleSet* selects the applicable firewall rule based on the subject and destination of the message and returns the permission and packet. Based on the permission returned, appropriate operations will be taken depending on if the packet is allowed, denied or discarded.

Because of lack of space, clauses R8, R9 and R10 are not shown in Listing 1.1. They are, however, similar to clauses R5, R6 and R7 although they refer to the case where permission is already received from the exception rule. So again, depending on the permission, the packet is allowed, denied or discarded.

As an end to the chain of forward and arrived pairs, any message arrived from the firewall is delivered by means of clause R11.

We demonstrated by this extract of code how LGI is flexible in terms of content and semantics of agents' CS and messages exchanged. Next we see another extract of code in Listing 1.2, this time containing an extract of the procedure *checkRuleSet*.

Clause R14 contains the goal *checkRuleSet(S,D,PA,PE)*. S and D, which are of the format S@domainS and D@domainD, respectively, unify with the subject and destination of the message and will be used to retrieve their IP and port information. PA unifies with the packet that is formed with S and D information and PE unifies with a permission, to be retrieved from the rule set. Each rule set has four arguments A1 to A4 in the *checkRuleSet* procedure: A1 corresponding to IP source *ips*, A2 corresponding to source port *Ps*, A3 corresponding to IP

destination *ipd* and finally A4 corresponding to destination port *Pd*, as seen in Section 2. Therefore, for each rule set, compareA1 to compareA4 are performed to compare each rule argument with the packet. When all arguments unify, the cut applies and the permission is returned to the main code. Listing 1.2 only contains compareA1 (R15, R16 and R17) and compareA2 clauses (R18, R19, R20), for brevity. The remaining six other clauses for A3 and A4 are similar to the ones presented.

In summary, LGI is flexible enough to accept any well-formed Prolog procedure as conditions in its ECA rules. Whether access is granted is determined dynamically in real time depending on the physical location of the workstation and the role of the maintenance employee. It is not needed to manually *open* a firewall to allow maintenance.

5 Discussion

As shown in the previous section, the middleware architecture and policy specification framework provided by LGI can be used to describe the security architecture for our running example in one integrated way across the three domains identified in the introduction. This is done in a declarative style using event-condition-action rules expressed in Prolog. In our example, the infrastructure is described directly in the form of Prolog facts, but it is also possible to use external databases as fact bases. With the appropriate interfaces, existing configuration databases and user directories can be used. Arbitrarily complex routines can then be built to deal with the inherent complexity of the security setup of a large organization. Using Prolog also provided the ground for a formal representation of policies. In addition to this, LGI provides the ability to automatically enforce policies in a distributed environment although at the expenses of installing its middleware on all computational nodes that need to be subjected to a policy.

According to our previous survey [14], LGI is able to express other access control policy models such as the Mandatory Access Control (MAC) and the Discretionary Access Control (DAC), in addition to the Role-Based Access Control Model (RBAC). Besides, it is able to express different types of access control policies such as obligation that triggers deadline events and sanction operations automatically, authorization and delegation. Thus, although not the focus of the present paper, LGI is potentially able to express cross-domain policies involving other models and types of access control as well. Future work in this direction is in our plans.

LGI, as a coordination framework, binds the specification of policies to the concept of message exchange. On the one hand, this way of thinking fits well with an important type of middleware; namely message-oriented middleware. Also modern approaches such as SOA/Web Services are often used in a message-oriented way. On the other hand, LGI forces everything to be specified in terms of message exchange, even when this is not intuitive. For instance, local access control (e.g. a workstation accessing its own resources) has to be modeled in terms of message exchange.

Despite the beneficial aspects of Prolog and LGI, the task of debugging Prolog *laws* can be complex. Prolog supports debugging aids in the format of predicates like trace, spy and even print [12]. The LGI toolkit Moses supports debugging and testing operations like show, discloseCS and enterTest [8] and also provides a Law Tester module. However, including working Prolog code into a LGI *law* code does not necessarily result in a guaranteed success. Other sources of problem can turn into confusing situations. As a first example, syntax errors on commands specific to LGI such as primitive operations are not detected and causes rules after the one which contains the problem to be ignored at run-time. As a second example, we found out in our example that Prolog operators like => and =< are ignored in LGI laws so, to avoid this, the interval inside a range must be tested with > and < and then bounds must be tested separately. As a last example, problems in a law usually disable the debugging aiding operations and it can turn into a deadlock until the source of the problem is found.

6 Related work

Our work is both related to security policy specification languages and to model-driven security.

In the **security policy specification** literature, most languages are domain specific. Thus, we have the domain of access control e.g. Security Policy Language [15], Authorization Specification Language [16] and Temporal Role-Based Access Control [17] and the domain of network e.g. Path-based Policy Language [18], Routing Policy Specification Language [19] and Clark's Policy Term [20]. However, we also have some overlaps between domains. Ponder [21] policies, for example, can combine parameters of Quality of Services (QoS) within networks with access control elements. Additionally, a recent stream of research is looking into physical conditions such as the location of a user in access control policies [22,23,24]. Therefore, although we can identify languages that allow interceptions between two domains, we are not aware of any language that provides the ability to combine all these three domains in a single policy, like it is possible with LGI.

Model-driven security uses graphic models to make the connection between architectural domains explicit. Luck et al. [25] propose a graphical tool which allows linking elements from different domains resulting in configuration files that can be implemented by the Linux kernel extension IPchains. Their approach has two drawbacks. The first drawback is visual pollution which compromises administration and scalability when representing real-life network environments. Albuquerque et al. realized this gap and proposed an intermediate level of abstraction to their model [7,26]. However, still a one to one relationship between the number of objects in this level and the number of users remains. Therefore, probably in large organizations the number of connections between objects and users would not be understandable graphically. Second, it seems that the rules generated automatically [7] by the model will be imported as files to be enforced by IPchains. Therefore, if the model changes, rule files will

have to be changed manually. Furthermore, rules at the level of IPchains do not understand the abstractions of the upper levels of the model, thus in the model-based approach we have a refinement process and not an alignment of domains in the sense that we cannot use elements of different domains concurrently in a combined policy. In summary, by looking at available model-driven security approaches we observe that: (i) they are not scalable in terms of the amount of cross-domain connections to be represented graphically, (ii) they are not enforced in terms of automatic configuration of network resources and (iii) they are based on refinement and so policies are derived at the lower level and cannot combine elements from the upper levels. Our approach, however, avoids a high level of granularity by dealing directly with roles on exception rules. Additionally, the policy can be enforced on any node across the network, considering LGI controllers are installed on all computational nodes involved. Finally, we can deal with elements from three domains without restrictions.

7 Conclusion

IT security in large organizations spans three architectural domains: access control, network layout and physical infrastructure. It is important that these domains are kept aligned as a way to manage the growing complexity of organizations. Current approaches in the area of policy specification and model-driven security either do not allow for the combination of elements from all these different domains in a single policy or suffer from the scalability of a graphical representation. We have demonstrated in this paper the ability to specify and enforce a cross-domain policy using a framework called Law Governed Interaction (LGI), which provides a modeling framework that alleviates these problems. LGI is flexible enough while still preserving the benefits of formal specification and automatic enforcement.

As future work, we intend to compare LGI, as a formal specification language based on Prolog, with Alloy [27], a first-order formal specification language. The goal is to discuss benefits and drawbacks of each approach based on running examples involving interactions between agents. More specifically, we want to investigate the preventive characteristic of LGI at policy run-time, achieved by governing agents according to a *law*, against the detective characteristic of Alloy at policy design-time, achieved by the possibility to reason about policy inconsistencies for example.

Acknowledgments

This research is supported by the research program Sentinels (www.sentinels.nl). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

References

1. IT Governance Institute: CobiT 4.0 - Control Objectives for Information and related Technology (2005) http://www.itgi.org.
2. Office of Government Commerce: IT Infrastructure Library (ITIL) (2000) www.itil.co.uk/.
3. International Organization for Standardization: ISO 9000: Quality Management (2003) http://www.bsi-global.com/Global/iso9000.xalter.
4. Zachman, J.A.: A framework for information systems architecture. IBM Syst. J. **26**(3) (1987) 276–292
5. van Eck, P., Blanken, H.M., Wieringa, R.: Project graal: Towards operational architecture alignment. International Journal Cooperative Information Systems **13**(3) (2004) 235–255 http://dx.doi.org/10.1142/S0218843004000961.
6. et al., M.L.: (Enterprise Architecture at Work - Modeling, Communication and Analysis)
7. de Albuquerque, J.P., Krumm, H., de Geus, P.L.: Policy modeling and refinement for network security systems. In: POLICY '05: 6th IEEE International Workshop on Policies for Distributed Systems and Networks, IEEE Computer Society (2005) 24–33 http://dx.doi.org/10.1109/POLICY.2005.24.
8. Minsky, N.: Law governed interaction (LGI): A distributed coordination and control mechanism, version 0.9.2. http://www.moses.rutgers.edu/documentation/manual.pdf (2005)
9. Ungureanu, V., Minsky, N.H.: Establishing business rules for inter-enterprise electronic commerce. In: DISC'00: Proceeding of the 14th International Symposium on Distributed Computing. LNCS 1914, Springer-Verlag (2000) 179–193
10. Ao, X., Minsky, N.H.: On the role of roles: from role-based to role-sensitive access control. In: SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies, New York, NY, USA, ACM Press (2004) 51–60
11. Ao, X., Minsky, N., Nguyen, T.D.: A hierarchical policy specification language and enforcement mechanism for governing digital enterprises. In: POLICY'02: Proceeding of the Third IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, IEEE Computer Society (2002) 38–49 http://doi.ieeecomputersociety.org/10.1109/POLICY.2002.1011292.
12. Bratko, I.: Prolog Programming for Artificial Intelligence. 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
13. Minsky, N.H., Ungureanu, V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. ACM Transactions on Software Engineering and Methodology (**9**(3))
14. Franqueira, V.N.L.: Access control from an intrusion detection perspective. Technical Report TR-CTIT-06-10, University of Twente, CTIT (2006) http://wwwhome.cs.utwente.nl/~franqueirav/Publication/AccessReport.pdf.
15. Ribeiro, C., Guedes, P.: SPL: An access control language for security policies with complex constraints. Technical Report RT-0001-99, INESC, Lisboa, Portugal (1999) citeseer.ist.psu.edu/ribeiro99spl.html.
16. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A logical language for expressing authorizations. In: SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (1997) 31–42
17. Bertino, E., Bonatti, P.A., Ferrari, E.: TRBAC: a temporal role-based access control model. In: RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control, New York, NY, USA, ACM Press (2000) 21–30 http://doi.acm.org/10.1145/344287.344298.

18. Stone, G.N., Lundy, G., Xie, G.G.: Network policy languages: A survey and a new approach. IEEE Network **15**(1) (2001) 10–21 http://www.cs.nps.navy.mil/people/faculty/xie/papers/PPL-survey-IEEENework01.pdf.
19. Meyer, D., Schmitz, J., Orange, C., Prior, M., Alaettinoglu, C.: Using RPSL in Practice (1999) Network Working Group Request for Comments: 2650 - http://www.rfc-archive.org/getrfc.php?rfc=2650.
20. Clark, D.: Policy Routing in Internet Protocols (1989) Network Working Group Request for Comments: 1102 - http://www.rfc-archive.org/getrfc.php?rfc=1102.
21. Lymberopoulos, L., Lupu, E., Sloman, M.: An Adaptive Policy Based Framework for Network Services Management. Journal of Network and Systems Management **11** (2003) 277–303
22. Ardagna, C.A., Cremonini, M., Damiani, E., di Vimercati, S.D.C., Samarati, P.: Supporting location-based conditions in access control policies. In: ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, New York, NY, USA, ACM Press (2006) 212–222 http://doi.acm.org/10.1145/1128817.1128850.
23. Ray, I., Kumar, M.: Towards a Location-Based Mandatory Access Control Model. Computers & Security (**25**)
24. Ray, I., Yu, L.: Short paper: Towards a location-aware role-based access control model. In: SECURECOMM'05: Proceeding of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks, New York, NY, IEEE Computer Society Press (2005) 234–236 http://doi.ieeecomputersociety.org/10.1109/SECURECOMM.2005.50,.
25. Luck, I., Schafer, C., Krumm, H.: Model-based tool-assistance for packet-filter design. In: POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks. LNCS 1995, London, UK, Springer-Verlag (2001) 120–136
26. de Albuquerque, J.P., Krumm, H., de Geus, P.L.: On scalability and modularisation in the modelling of network security systems. In: ESORICS '05: Proceedings of the 10th European Symposium on Research in Computer Security. Volume 3679 of Lecture Notes in Computer Science., Springer (2005) 287–304
27. Jackson, D., Shlyakhter, I., Sridharan, M.: A Micromodularity Mechanism. In: ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA, ACM Press (2001) 62–73 http://doi.acm.org/10.1145/503209.503219.

Listing 1.1. Extract from *law* $\mathcal{L}_{\mathcal{F}}$

```
R2. sent(Source,Message,Dest) :- role(Role)@CS,
        location(Room,Socket)@CS,
        do(forward(Source,[Message,Dest,Role,Room,Socket],firewall)).

R3. arrived(Source,[Message,Dest,Role,Room,Socket],firewall) :-
        checkRuleExceptions(Source,Dest,
        [role(Role),location(Room,Socket)],Permission,Packet),
        do(forward(Source,[Message,Packet,Dest,Role,Room,Socket,Permission],
        firewall)).

R4. arrived(Source,[Message,Dest,Role,Room,Socket],firewall) :-
        not(checkRuleExceptions(Source,Dest,
        [role(Role),location(Room,Socket)],Permission,Packet)),
        do(forward(Source,[Message,Dest,Role,Room,Socket,none],firewall)).

R5. arrived(Source,[Message,Dest,Role,Room,Socket,none],firewall) :-
        checkRuleSet(Source,Dest,Permission,Packet),
        (Permission=allow),
        do(forward(firewall,Message,Dest)).

R6. arrived(Source,[Message,Dest,Role,Room,Socket,none],firewall) :-
        checkRuleSet(Source,Dest,Permission,Packet),
        (Permission=deny),
        do(add(auditRecord([packetDenied,Packet]))),
        do(forward(firewall,[packetDenied,Packet],Source)).

R7. arrived(Source,[Message,Dest,Role,Room,Socket,none],firewall) :-
        checkRuleSet(Source,Dest,Permission,Packet),
        not(Permission=allow), not(Permission=deny),
        do(add(auditRecord([packetDiscarded,Packet]))).
        .
        .
        .
R11. arrived(firewall,Any,Any) :- do(deliver).
```

Listing 1.2. Extract from *law* $\mathcal{L}_{\mathcal{F}}$ - procedure *checkRuleSet*

```
R14. checkRuleSet(S,D,PE,PA) :-
        dnsPortLookup(S,IPs1,IPs2,IPs3,IPs4,Ps),
        dnsPortLookup(D,IPd1,IPd2,IPd3,IPd4,Pd),
        PA=packet(ips(IPs1,IPs2,IPs3,IPs4),Ps,
                  ipd(IPd1,IPd2,IPd3,IPd4),Pd),
        ruleSet(A1,A2,A3,A4,PE),
        compareA1(PA,A1),
        compareA2(PA,A2),
        compareA3(PA,A3),
        compareA4(PA,A4),!.

R15. compareA1(packet(ips(IPs1,IPs2,IPs3,IPs4),X,Y,Z),
        anyIP).

R16. compareA1(packet(ips(IPs1,IPs2,IPs3,IPs4),X,Y,Z),
        ips(IPs1,IPs2,IPs3,IPs4)).

R17. compareA1(packet(ips(IPs1,IPs2,IPs3,IPs4),X,Y,Z),
        ipRangeS(IPs1,IPs2,IPs3,IPs4From,IPs4To)) :-
        (IPs4 => IPs4From), (IPs4 =< IPs4To).

R18. compareA2(packet(W,Ps,Y,Z),
        anyPort).

R19. compareA2(packet(W,Ps,Y,Z),
        Ps).

R20. compareA2(packet(W,Ps,Y,Z),
        portRangeS(PsFrom,PsTo)) :-
        (Ps > PsFrom), (Ps < PsTo).
```