Rules for Identity and Access Control

Rieks Joosten[1], Stef Joosten[2,3]
1 TNO Information and Communication Technology, P.O.Box 1416,
9701 BK Groningen, the Netherlands, rieks.joosten@tno.nl
WWW home page: http://www.tno.nl/index.cfm?Taal=2
2 Open University of the Netherlands, P.O. Box 2960,
6401 DL Heerlen, the Netherlands, stef.joosten@ou.nl
WWW home page: http://www.ou.nl/eCache/DEF/36.html
3 Ordina B&E Solutions, P.O. Box 7101,
3430 JC Nieuwegein, the Netherlands, stef.joosten@ordina.nl
WWW home page: http://www.ordina.nl/index.asp?LanguageCode=EN

Abstract. The future of Identity in the automated contexts of information society depends on our ability to (formally) express rules governing their use, and mechanisms to enforce such rules. Experimental tooling has allowed us to convert formally represented Identity and Access Management (IAM) rules into a functional specification for IAM services and a corresponding software implementation, allowing businesses and enterprise architects to efficiently prototype IAM rule-based solutions in order to fine-tune the rules they need and may commit to. Our research proposes a set of IAM business rules, complemented with a specification of IAM services that fully comply to these requirements. On a larger scale, this approach may help to solve cross-domain identity issues e.g. between governmental organizations.

1 Introduction

Information systems deployed by large businesses are like a sunny, hot and busy Arabic marketplace. All sorts of vehicles (information streams) try to go where they want to go today. Drivers argue with one another to make way so that they can pass, without caring all too much about each other. Their state-of-the-art, hi-tech cars do not prevent them getting stuck in these continuous traffic jams. Obviously, traffic flow would benefit if drivers could be persuaded to adhere to rules that govern the right of way. To do this, you need a good set of rules, a mechanism to enforce such rules and an education program for drivers.

This metaphor also applies to Identity in the market place of Information Society. All sorts of businesses try to utilize identities for their own day to day purposes. There is hardly any meaningful alignment of identity information, a lot of discussions take place in an attempt to align the semantics of identity information, but progress is little. State-of-the-art technologies and high-tech systems are deployed, but the problems remain. If businesses could be persuaded to adhere to rules that govern the way we use identity information, information society as a whole will benefit. To do this, we need a good set of rules, a mechanism to enforce such rules and an education program for businesses. Also, we need that these rules and the enforcement mechanism may also operate in an automated context, which sets additional requirements to the precision by which such rules are formulated.

This article describes work that has been done to find such rules for the 'market place' of Identity and Access Management (IAM) in automated contexts. First, we introduce the idea of automatable IAM rules by giving an example. Then we describe a demonstrator with different applications (business processes) using an IAM service layer enforcing the IAM rules. Because these applications use the IAM service layer, they are provably compliant with the set of IAM rules that was used to specify the services, and generate the code that implements them.

The importance of the demonstrator is twofold. First, it makes IAM issues, as well as consequences of IAM rules, tangible to the business. Rather than discussing technicalities (e.g. standards or vendor products) as is currently often done, this work allows business people to focus on what it is they want IAM to do for them and have them express this in terms of business rules. Secondly, it shows that formalizing such rules may lead straight toward the enforcement thereof in the automated systems of that business.

The scientific contribution of this work lies in the formalization of IAM, which yields a thorough understanding of its issues. A compliant service layer has been specified and built as an embodiment of this result.

As the method we use and the associated tooling become mature, we will be in a better position to also address cross-domain Identity issues such as those that governmental agencies struggle with.

2 IAM Rules

Creating rules for IAM is a creative process that captures the essence of IAM. Both this work and its results are comparable to legislative processes: discussions, negotiations and compromise ultimately lead to rules (laws) that, once formulated and approved, are meant to be obeyed. Different sets of IAM rules may exist in different contexts, as different laws exist in (different parts of) different countries. Interoperation between contexts (business units, businesses, or countries) requires that rules are attuned or harmonized, which is basically the same process, albeit that existing rules in specific contexts should be changed in order to remain consistent with the harmonized set.

While judicial laws are to be processed by humans, our rules must also be processable by computers. Therefore, we require that our IAM rules are expressible

in natural language (NL) for use with humans, and also have a formal representation (FR) such as relational algebra or predicate logic. Because FR is more precise than NL, the FR of the IAM rules is authoritative in our work. FRs allow us to do formal reasoning with the IAM rules or rule sets. For example, when trying to harmonize two IAM rule sets, the consistency proof eliminates the need of discussions, whereas any proof of inconsistency precisely defines an issue to be discussed. This alone makes the harmonization process much more efficient.

After having created the rules, they must be put to use, which is to say: they must be complied with. As an example, consider the following (subset of the real) IAM rules:

1. Any service (function, or method) that requires a permission may only be executed from sessions in which that permission is available.
2. A permission is available within a session if and only if that session has activated a role to which this permission has been assigned.
3. A role is activated within a session if and only if (a) sessions of this type are designed to activate this role and (b) the session's user has been assigned this role.
4. Every contract must have been signed by all contract parties.

Now consider the situation where we have a user, John, who wants to review a contract, and digitally sign it after having agreed to it. Suppose an application called CRM exists that he might use to do this, as CRM is programmed to activate roles such as 'Customer' which has been assigned permissions P1 and P2, where P1 is the permission required by the service 'get_contract' which retrieves contract information and P2 is required by 'approve_contract', the service used for the digital signing of contracts. Also assume that CRM uses the IAM service 'AuthUser' for authenticating a user's credentials (e.g. username and password).

The first thing the CRM application does when John requests a session with it, is that it invokes 'AuthUser' to check John's credentials and verify that John is really John. However, as soon as John's credentials have been authenticated, rule 3 calls for the activation of the 'Customer' role in the session as the session was designed to activate this role and the session's user (John) has been assigned this role. As soon as this role is activated, rule 2 demands that permissions P1 and P2 are made available within the session, as the role of 'Customer' has been activated in that session and both permissions are assigned to this role. From this we see that IAM rules such as 2 and 3 not only specify functionality that systems should exhibit, but that this functionality can be automatically provided by 'AuthUser'. Rules of this type are called 'Automatable Operative Rules', which is a further distrinction from the notion 'Operative Rules' as defined in [2].

Now that the invocation of 'AuthUser' has made permissions P1 and P2 available within the CRM session, John requests CRM to show his contract information. To do this, CRM invokes 'get_contract'. This service starts by checking whether P1 exists in the session it is called from because it must comply with rule 1. As the permission exists, 'get_contract' returns the required information. Here we see that rule 1 specifies constraints on behavior rather than the behavior itself as rules 2 and 3 did. Rules of this type are called 'Structural Rules'.

Note that all this time, the contract existed and had not been signed by all contract parties, implying that rule 4 was being violated all this time. We want rules like this to exist as they specify a desirable business situation, and violations of such rules signal that (manual) work needs to be done; that is why we call them 'Manual Operative Rules'**Error! Bookmark not defined.**. In fact, rules like this can be used to drive a process engine [3]. The fact that John's contract hasn't been signed yet may trigger John to review and sign the contract, and may trigger other parties involved to either get other parties to sign, or to destroy the contract as either outcome would satisfy rule 4.

3 A Rule-based IAM Demonstrator

Once a business has established its set of IAM rules, a service layer for IAM services can be specified directly from the (FR of) these rules. This rule-based specification has the property that it includes all functions the business may ever need to become and remain compliant to these rules, and all functional requirements in the specification can be traced back to one or more rules. Also, it can be proved that any information system built to these specifications will maintain all IAM rules when each service complies to its specification and all specified services have been realized, and non compliance can be proved from the specification.

We have created an IAM demonstrator in which

1. portals are simulated: one for a financial context, another for the business context (ZM) and the third for the consumer context (CM).
2. business services may be called: one for on-line bill checking (IOL), another one for SOx[1] accounting.
3. an IAM service layer provides all necessary IAM functionality.

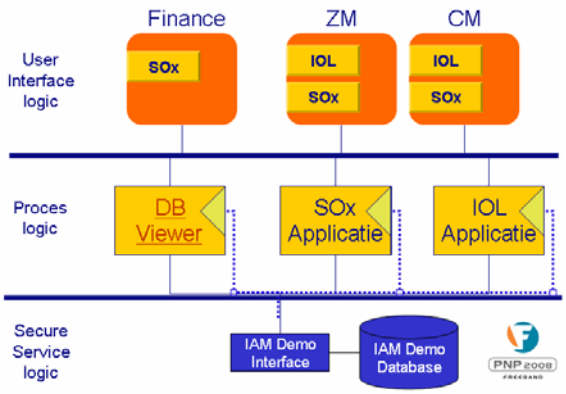

Figure 1: IAM Demonstrator

The IAM service layer has been generated directly from a set of both IAM and SOx rules, and consists of a PHP functional layer on top of a MySQL database. The business services have been programmed in PHP by hand, and run on an Apache web server. Figure 1 shows the 'home page' of the demo, which has been created such that clicking on the IOL-box in the CM portal invokes the IOL business service logic as if it were called from the consumer portal.

[1] SOx refers to the Sarbanes-Oxley act of 2002 [1], which establishes stringent financial reporting requirements for companies doing business in the United States

Using this demo, we show that one business service is capable of dealing with identities from different businesses in different contexts. For example, the IOL service is equally capable of dealing with the creation of a new customer in a business context as it is in the consumer context. Also, in the business context it is equally capable of providing functionality to the business (e.g. for creating/deleting a customer) as it is for customers (e.g. for creating or deleting additional customer accounts. In fact, the demo shows that decisions with respect to how the IOL service should operate in the CM context do not affect its operation in the ZM context, even though it is the same service.

The demo cannot show rule violations, as all rules are upheld by the IAM service layer and both business services use this layer for all IAM (and SOx) functionality. It can however show the consequences thereof: a customer that is logged in to the ZM portal can only see its own data and its own users, whereas a properly authorized user from the business can see all customer accounts.

The demo also shows that when functions that are necessary for the financial people within a large organization, such as inspection of a SOX-log, can be made available to other business units as well, by simple administrative actions. Simply providing the ZM-administrator with the permission to look into the SOX-log is sufficient. After all, the rule stating that financial information may only be seen either by the domains that have a direct interest, or the financial administration, cannot be violated as the service layer upholds it. Therefore, we need not fear that the ZM-administrator will see financial information from the CM-context.

Similarly, and this is new for many businesses, this functionality can be made available in exactly the same way to customers. The rules enforced by the IAM service layer ensure that each customer can only see or do things within the room defined by these rules. Businesses can now easily provide customers with information that is relevant for their SOX report.

4 Rules used in the demonstrator

In order to give the reader an idea of what rules look like, we provide most of the IAM rules that we used for our demonstrator. These rules define how responsibilities are modeled in relation to performing actions, a simple form of authentication using 'tokens' (a generic notion, covering username/passwords as well as certificates) and authorizations based on RBAC [4] and a rule implementing 'Chinese walls':

1. Every domain, i.e. a named set of responsibilities, has at least one domain-manager that bears all domain responsibilities.
2. Whatever happens in a session is the responsibility of precisely one domain.
3. Every session is of precisely one type (the sessiontype).
4. Sessions of a given type may only run within a domain if there exists a valid sessiontype approval within that domain for this sessiontype.
5. A tokenadministration consists of entries, each of which is uniquely characterized by a token, the type of that token and the token's issuer.
6. Each entry in the tokenadministration has precisely one userid.

7. Each entry in the tokenadministration has precisely one 'responsible domain', i.e. the domain that bears all responsibility for every use of the token.
8. If one userid is associated with multiple tokenadministration entries, each of them has the same responsible domain.
9. Logging into a session means providing a token, its tokentype and its issuer.
10. A sessiontoken is a login-token where the provided token, tokentype and issuer identify an entry in the tokenadministration.
11. A sessionCoactor is the userid associated with a sessiontoken.
12. A sessionCodomain is the domain that is responsible for every use of a sessiontoken.
13. There is at most one sessionCoactor and one sessionCodomain at any time.
14. Whenever a token, tokentype and tokenissuer combination is presented in a session that already has or has had a sessiontoken, this token shall only become a sessiontoken if its associated userid equals the sessionCoactor.
15. A session shall only access dataobjects containing a list of Codomains if the sessionCodomain appears in that list. Note that this access always requires a valid login. (This rule helps to define so-called 'Chinese walls')
16. Every action whose execution implies taking a risk, must require a permission.
17. An action that requires permissions may only execute in sessions that have all such permissions.
18. The permissions a session has is the union of all permissions of all sessionroles.
19. A sessionrole for a session of a certain type is any role that (1) has been assigned to the sessionCoactor, and (2) has been defined as a role that may be activated for sessions of this type.
20. A role may only be assigned to existing userid's.
21. A token can only become a sessiontoken (i.e.: you may only login) in a session of a certain type if the userid associated with that token has been assigned at least one role that is relevant for sessions of that type.

5 Results

We have applied the above approach to define business rules for an IAM service layer for a large Telco in the Netherlands. Talking to people from various business departments made us particularly aware of how diverse the ideas with respect to IAM really are. For example, for the business unit ZM (corporate market), IAM is equivalent with a part of customer care, where ZM-customers can create accounts and accompanying permissions for their own employees. The finance department sees IAM primarily in the context of having to be compliant with the Sarbanes-Oxley act [1].

Abstracting from the use-cases provided by the business people and reconciling their needs, resulted in a set of unambiguous and consistent business rules which we could both represent formally and in a way that the business could understand. In our experience, good rule sets tend to remain stable, meaning that each time they are used, only slight variations occur. Judging by this criterion, the demonstration rules

have some good parts, whereas other parts still need work. Earlier versions of this work are documented in [5, 6]

Experimental tooling for generating a PHP service layer on top of a MySQL database allowed us to evaluate various rule sets 'hands on'. Such exercises have been invaluable in discovering which rules we need, how they should be formulated, and how to conceptually think about Identity management.

From a reasonable rule set (described above) a service layer was generated allowing us to demonstrate the effects such rules would have for the business. A service layer such as ours, that guarantees compliance with a set of rules, goes a step beyond work as described e.g. in [7] where a tool only checks compliance.

Also, the ability to create functional specifications for the rule set, allows us to give the business a pretty good estimate of what actual implementation of the service layer is going to cost in terms of function points, which is the basis on which IT organizations make their offerings. For example, the functional specification for the demo has 118 function points. With a price of say 1000 euro per function point, a business implementation of the service layer would cost about 118.000 euro.

While creating the demo, we noticed that the application programming required limited knowledge of the rules (as we expected): programmers only need to know how to use the IAM service interface. For the business, this means that rules may be changed at will as long as this does not affect the functional interface specification.

We also noticed that programming actually becomes easier as programmers no longer need to calculate permissions from roles or check whether or not a function might be executed. All such concerns are hidden, and taken care of in the IAM service layer. This not only limits the amount of code to be written, but also frees the minds of programmers of IAM concerns, allowing them to keep their attention focused on the actual business service to be programmed.

The demo shows that it is possible to share the same IAM functionality in contexts that did not use to do this before. The reason for this is that instead of implementing IAM for a particular context using the context's particular vocabulary and views, we have abstracted from use-cases of multiple contexts, and created rules that describe all of them. Then, obviously, a service layer implementing such rules is useful for every such context.

Showing the demo in workshops with business architects puts the message across that if identity situations similar to Arabic marketplaces are to be avoided, cross-domain IAM issues are to be considered as a coherent set of issues rather than individual sets of concerns. Also, the demo helps discussions to stay much more focused on what the business wants rather than on technicalities such as the systems or standards to use for implementation.

6 Conclusions and future work

Abstracting from multiple use-cases in multiple business contexts, we have derived a set of formal Identity and Access Management (IAM) rules, from which we have generated IAM service layer software that enforces these rules. We have built a demonstrator on top of this, consisting of multiple applications and simulated

portals that are provably compliant with these IAM rules. We have found that the short turnaround time for building a demo for a set of rules is an invaluable instrument for fine tuning of the IAM rule set. We also found that the final demonstrator helps the business to focus on the real IAM issues (rather than on technicalities), putting them in a position to commit to such rules. This work shows that it is practically feasible to reconcile different business needs in such a way that a single set of automatable services can do the job, which is what is not only needed for IAM within large businesses, but also for Identity management over multiple countries.

Future research will work towards IAM rule sets that address other issues such as privacy, token management and claim based access control. We intend to further interoperability across businesses, in particular where businesses have decided to use different rules. Another focus will be on making the relation explicit with areas such as process architecture and/or commercial products. Additional research is required to professionalize the tools we have been working with, in an attempt to provide all necessary artifacts that state-of-the-art software factories need to produce commercial products.

Acknowledgement

The work presented here has partly been carried out in the collaborative project PNP2008 [8], which is supported by the Freeband Communication technology program of the Dutch Ministry of Economic Affairs.

References

1. United States Code: Sarbanes-Oxley Act of 2002, HR 3763, PL 107-204, 116 Stat 745. Codified in sections 11, 15, 18, 28, and 29 USC (2002).
2. OMG, Semantics of Business Vocabulary and Business Rules (SBVR), 2005-08-01.
3. S. Joosten and R. Joosten, Specifying business processes by means of rules, in: Proceedings European Business Rules Conference, Amsterdam, June 2005.
4. American National Standards Institute, ANSI INCITS 359-2004 for Information Technology – Role Based Access Control, February 2004.
5. R. Joosten and B.Beute, Requirements for Personal Network Security Architecture Specifications, Freeband PNP2008 Deliverable 2.4, April 2005.
6. E. van Essen, B. Beute and R. Joosten: Realizing PNP architecture descriptions, Freeband PNP2008 Deliverable 3.2, May 2005.
7. S. Höhn and J. Jürjens, Automated Checking of SAP Security Permissions, Sixth IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems (IICIS 2003).
8. Freeband PNP2008 project: http://pnp2008.freeband.nl (2005)