A Compositional Event & Time-based Policy Model

Helge Janicke, Antonio Cau, François Siewe, Hussein Zedan and Kevin Jones
Software Technology Research Laboratory,
De Montfort University, Leicester, UK LE1 9BH
Email: {heljanic, acau, fsiewe, hzedan, kij}@dmu.ac.uk

***Abstract*— Policies are increasingly used to govern the behaviour of complex distributed systems. Most policy models that allow policy composition, to address the complexity of policies, are only concerned with structural composition. In this paper we argue that it is natural to compose policies also along the temporal axis, i.e. express policies that can dynamically change over time or on the occurrence of events. We present a dynamic policy frame-work that has a sound semantics in Interval Temporal Logic and allows both structural and temporal composition.**

We provide examples of authorisation, delegation and obligation policies that can be activity-based, state-based or history-based, i.e. expressing decisions on the history of execution. Examples for the composition of policies show how the framework can be used to express policies for systems that operate in an environment that is characterised by uncertainty. Finally tool-support for the specification and analysis of dynamic policies is presented.

I. Introduction

Policies are gaining a more and more important role in governing the behaviour of complex systems that manage a multitude of resources for a usually large number of users. The specification and maintenance of policies is a difficult task, and it is often left to system administrators to find their way through complex policies. Policies can contain a vast number of rules. When maintaining policies administrators are facing difficulties to know whether similar rules do already exist, the new rule will encompass other already existing rules or even does contradict previously defined rules.

The rationale of a policy framework is to provide language and tool-support that allows administrators to write and modify policies with ease. Administrators must be able to analyse the policies to be aware of the effect that a modification will have on the system. Policy administration is a highly responsible task that is critical to the correct functioning of the system. An administrator must have a high level of assurance, that a modification in the policy will yield the desired effect.

A policy model is the formal grounding of the policy language, that provides a clean and unambiguous semantic of policies. The language is required to be *expressive*, i.e. be able to specify a wide variety of different policies, for example the BLP [1], Chinese Wall [2] or the BMA policy [3]. At the same time the language must be *simple* to allow the ease of specification. The underlying semantics given by the model allows the development of automated tool-support to provide the desired ability to analyse policies.

In this paper we present a dynamic and compositional policy model. By dynamic we mean that policies can change over time and on the occurrence of events. By compositional we mean that parts of the policy can be specified individually and then be composed to yield the overall system policy. To be able to formally express both aspects within the same mathematical framework, we chose Interval Temporal Logic (ITL) [4] as the underlying logic. Being a temporal logic the expression of temporal dependencies between policies is natural. The compositional proof-system of ITL [5] provides the foundation to infer properties of the overall specification, from its components. A large set of proof-rules has been developed in [6].

The paper is organised as follows. Section II relates and compares our model to other policy models that have been developed. Sections III introduces the underlying logic that provides the logical framework for the semantic of the model. The model itself is introduced in sections IV and V. Where the former describes the rules and simple policies as smallest granularity of the specification and the latter introduces operators for their composition. Section VI presents tool-support, that is available for the specification and analysis of policies. We conclude the paper in section VII and outline the direction of our future research in this area.

II. Related Work

Work on policies has gained more and more attention in the recent past. The main objective of policies is to allow dynamic adaptation of the system's behaviour by updating policies. This allows modification (usually restriction) of the behaviour without modifying the system itself [7]. The application areas are diverse. Examples are legislation changes, respectively changes in their interpretation, within the health-care environment or limiting emergent behaviour of self*-systems developed for the military domain. In the military domain traditionally multi-level secure systems ensure that classified information cannot leak to individuals with insufficient clearance. A classic reference is the Bell La Padula model [1]. Multi Level Security (MLS) has some drawbacks, such as the tendency to over classification and limited flexibility for ad-hoc coalitions. Brewer and Nash [2] identified the need to be able to limit the possibility for fraud in commercial applications and showed that the BLP policy model is not adequate to describe their requirements.

The larger and distributed information systems become, the more challenging the demands on security policies and enforcement. Jajodia et.al. describe in [8] how multiple access control policies can be enforced by their flexible authorization

manager (FAM). We employ a similar mechanism in our model to resolve conflicts between policies, but allow for more flexibility to express resolution (decision) rules. As noted by Bonatti et.al. [9] their model, and many others, did not address policy composition. They rectified the situation by providing an algebra for policy composition in [9]. Their model allows the structural composition of policies in form of hierarchies and shows how conflicts can be resolved. Their algebra is only concerned with structural composition and does not take into account composition along the temporal axis.

Sloman divided authorisation policies in *activity based* and *state based* authorisations [7] and identifies additionally temporal constraints as optional to policies. Temporal constraints would be for example the statement "The policy applies from 9:00 to 17:00 every working day". A third category of authorisation policies would be a natural extension in terms of expressiveness: *history based authorisation* e.g. [10].

The aforementioned models do not allow the expression of temporal constraints. This issue has mainly been addressed in the area of role-based access control (RBAC) [11] specifically with Temporal Role-Based Access Control (TRBAC) [12] and the temporal logic extension to RBAC introduced in [13]. The former addresses temporal constraints in terms of (calender) time and is closely related to Sloman's definition of temporal constraints. The latter bears greater similarities with our model as it allows the specification of rules based on observed behaviour, to capture dynamic separation of duty. However, the important issue of policy composition and the arising difficulties with temporal conflicts are not addressed by any of the models.

As mentioned in the introduction, not only the expressiveness of the policy model is of importance, but also the simplicity. Over the years different approaches to make policies easier to express and maintain have been proposed. The Ponder policy framework [14], for example employs an object-oriented approach to policy definition, that greatly enhances the reuse of existing specifications. The LasCO policy language [15] appeals with its graphical representation to the policy designer. Other approaches, e.g. XACML, are much more concerned with the interoperability of their policy framework and describe their policies in a XML dialect, to simplify the exchange and machine readability of policies. These languages heavily rely on tool-support for specification, since the XML files are difficult to comprehend for the human reader. Whilst we agree that all of the concepts are beneficial or even necessary for the specification of large scale systems, we take the view that simplicity can be achieved through compositionality. Using compositional policies, administrators can "divide and conquer", i.e. focus on logical groups of requirements and compose the resulting policies to obtain the overall policy.

The following section presents ITL as the logical foundation to our policy framework.

III. Preliminary

Interval Temporal Logic (ITL) is a flexible notation for both propositional and first order reasoning about periods of time found in descriptions of hardware and software systems. It can handle both sequential and parallel composition unlike most temporal logics [16] since assumption/commitment paradigm and a set of compositional guidelines [17] are applied in ITL. There is a very powerful and practical compositional proof system for ITL [16]. That is, much of the proof of a system specified in ITL can be decomposed into proofs of its parts. It offers powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and timeliness.

1) Syntax and Semantics: The key notion of ITL is an *interval*. An interval σ is considered to be a (in)finite sequence of states $\sigma_0,\ \sigma_1 \ldots$, where a state σ_i is a mapping from the set of variables Var to the set of values Val. The length $|\sigma|$ of an interval $\sigma_0 \ldots \sigma_n$ is equal to n (one less than the number of states in the interval[1]; a one state interval has length 0).

Expressions

$e ::= \quad \mu \mid a \mid A \mid g(exp_1, \ldots, exp_n) \mid \imath a \colon f$

Formulae

$f ::= \quad p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \bullet f \mid$
$\mathsf{skip} \mid f_1 \,;\, f_2 \mid f^*$

Fig. 1. Syntax of ITL

The syntax of ITL is defined in Fig. 1 where μ is an integer value, a is a static variable (doesn't change within an interval), A is a state variable (can change within an interval), v a static or state variable, g is a function symbol and p is a predicate symbol.

The informal semantics of the most interesting constructs are as follows:

- skip: unit interval (length 1, i.e., an interval of two states).
- $f_1 \,;\, f_2$: holds if the interval can be decomposed ("chopped") into a prefix and suffix interval, such that f_1 holds over the prefix and f_2 over the suffix, or if the interval is infinite and f_1 holds for that interval. Note the last state of the interval over which f_1 holds is shared with the interval over which f_2 holds.
- f^*: holds if the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and can be decomposed into an infinite number of finite intervals for which f holds.
- $\imath a \colon f$: choose a value of a such that f holds. If there is no such an a then $\imath a \colon f$ takes an arbitrary value from a's range.
  Example: $\imath a \colon \mathsf{skip} \,;\, (x = a)$ which when evaluated over an interval of length at least one will give the value of x in the second state. If the interval is less than one it will give an arbitrary value.

[1]This has always been a convention in ITL

2) Derived constructs: Following is a list of some derived constructs which are useful for the specification of systems:

- $\bigcirc f \mathrel{\widehat{=}} \mathsf{skip}\,;f$: next f, f holds in the next state. that the value of X in the second state is 1 and the length of that interval is at least 1.
- $\mathsf{more} \mathrel{\widehat{=}} \mathsf{skip}\,;\mathsf{true}$: non-empty interval, i.e., any interval of length at least one.
- $\mathsf{empty} \mathrel{\widehat{=}} \neg\mathsf{more}$: empty interval, i.e., any interval of length zero (just one state).
- $\mathsf{inf} \mathrel{\widehat{=}} \mathsf{true}\,;\mathsf{false}$: infinite interval, i.e., any interval of infinite length.
- $\mathsf{finite} \mathrel{\widehat{=}} \neg\mathsf{inf}$: finite interval, i.e., any interval of finite length.
- $\Diamond f \mathrel{\widehat{=}} \mathsf{finite}\,;f$: sometimes f, i.e., any interval such that f holds over a suffix of that interval.
- $\Box f \mathrel{\widehat{=}} \neg\Diamond\neg f$: always f, i.e., any interval such that f for all suffixes of that interval.
- $\mathsf{fin}\ f \mathrel{\widehat{=}} \Box(\mathsf{empty} \supset f)$: final state, i.e., any interval such that f holds in the final state of that interval.
- $\mathsf{fin}\ exp \mathrel{\widehat{=}} \imath a\colon \mathsf{fin}\,(exp = a)$: end value, i.e., the value of exp in the last state of the interval.
- $len \mathrel{\widehat{=}} in\colon \exists I \cdot I = 0 \wedge \Box(\bigcirc I = I + 1) \wedge \mathsf{fin}\,(n = I)$: any interval with length n.
- $\lceil f \rceil^n \mathrel{\widehat{=}} f \wedge len = n$: any interval of length n over which f holds.
- $\boxdot \mathrel{\widehat{=}} f\,;\mathsf{true}$: some initial sub-interval.
- $\boxminus \mathrel{\widehat{=}} \neg\boxdot\neg f$: all initial sub-intervals.
- $f \frown g \mathrel{\widehat{=}} f\,;\mathsf{skip}\,;g$: weak sequence, see $f\,;g$, but intervals do *not* share a common state.
- $f^+ \mathrel{\widehat{=}} f\,;f^*$: iteration of f.
- $f^\oplus \mathrel{\widehat{=}} f\,;(\mathsf{skip}\,;f)^*$: iteration of f, but sub-intervals do not share a common state.
- $f \sqcap g \mathrel{\widehat{=}} f \vee g$: Non-deterministic Choice.
- $w?f : g \mathrel{\widehat{=}} (w \wedge f) \vee (\neg w \wedge g)$: Conditional Choice.

The use of rules to express security requirements has been shown to be effective in nearly all policy models. The operator *always-followed-by* is defined to be able to capture the semantic of a rule in ITL. The definition of the operator is given below.

Definition 1 (Always Followed By) *We define the operator* always-followed-by*, denoted by the symbol " $\mapsto$ ", as follows:*

$$f \ \mapsto\ w \mathrel{\widehat{=}} \boxminus((\Diamond f) \supset fin\ w)$$

where f stands for any ITL formula, and w is a state formula.

The definition (1) states that whenever the formula f holds in a (finite) subinterval then the state formula w must hold in the final state of that subinterval. That is, f is always followed by w.

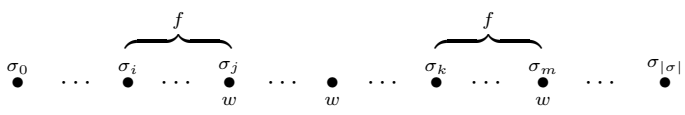

Fig. 2. Graphical illustration of $f \mapsto w$.

Informally, the meaning of the operator can be portrayed as in Figure 2, where each bullet represents a state. Note that in the Definition 1, w can be true in a state even if f does not hold in the left neighbourhood of the state. This allows the definition of different rules with the same consequence within one policy.

IV. Policy Model

This section introduces our policy model using a bottom up approach. In section IV-A the general notion of a rule is explained. We then explain specific classes of rules for different types of requirements.

A. Policy Rules

A policy rule is expressed by the operator *always-followed-by* i.e. when the past behaviour f is observed, a specific decision (expressed by w) is taken by the enforcement mechanism in the system. Enforcement mechanisms interpret the policy to determine for example access control decisions or ensure conformance with obligations. The mechanisms themselves are part of the trusted computing base (TBC) on which the conformance of the systems behaviour is dependent.

The model allows the expression of rules for authorisation, delegation and obligation. In each of these classes rules can be present in negative and positive form, which is indicated by the superscripts $^+$ respectively $^-$. The expression of both forms, can easily lead to conflicts in the specification so we follow an approach similar to the one presented by Jajodia et.al. in [8]. A conflict resolution rule (or decision rule) decides whether to give precedence to the positive statement or its negation. The right-hand side of conflict resolution rules does not carry a superscript.

Using conflict resolution rules it is possible to express open, closed, hybrid policies and more sophisticated resolution techniques like subgroup over-riding, denial takes precedence, etc. We will see some of these conflict resolution rules in subsequent examples.

The completeness of a policy specification is achieved, by applying a completeness algorithm [6] that transforms the implication relationship between premise and consequence of a rule into an equivalence relation. The algorithm implements the closed world assumption, in which whenever no positive decision can be derived, it defaults to the negative outcome. This means if the specification does not explicitly state an access right, the access is denied. Similarly, if the system does not explicitly state the right to delegate, the default is that the right cannot be delegated. For obligation this is intuitive: unless explicitly stated a subject does not have any obligation.

B. Authorisation

A positive authorisation rule states that a subject is allowed to perform a specific action on an object. A negative authorisation rule respectively denies a subject to perform a specific action on an object. Usually authorisation policies form the largest part of security policies. Therefore we will emphasise in the discussion of rules on authorisation rules.

The following gives simple examples of authorisation rules, taking well known rules from MLS research. Example 1 shows an activity based authorisation. Example 2 shows a state based authorisation, where the state is the current security level of the object, respectively the clearance level of the accessing subject.

Example 1 (Positive Authorisation) *Explicit allowance for all subjects to read from all objects.*

$$true \;\mapsto\; Autho^{+}(S, O, read)$$

Any subject S is allowed (denoted by the $^{+}$ superscript of $Autho$) to read objects O.

The uppercase S,O and A denotes universally quantified variables that range over the set of all subjects, objects and actions respectively (see Section V-B).

Example 2 (Negative Authorisation)
The no read-up rule first described by Bell LaPadula in [18].

$$\lceil Level(O) > Clearance(S) \rceil^{0} \;\mapsto\; Autho^{-}(S, O, read)$$

Any subject S is denied (denoted by the $^{-}$ superscript of $Autho$) to read objects O if the security level of O is greater than the clearance level of the S.

Combining both rules in the same specification will lead to a conflict. The rule in Example 1 grants unrestricted access to information, whilst the rule in Example 2 denies subjects with insufficient clearance. If there exists one subject whose clearance level is lower than the security level of any object then there is a conflict. The conflict is resolved by the conflict resolution rule. Example 3 shows such a rule, that in this case gives denials precedence, thus providing the semantics of the no-read-up rule if all rules were in the same specification.

Example 3 (Authorisation, conflict resolution) *Denial takes precedence over allowance.*

$$\lceil \neg Autho^{-}(S, O, A) \wedge Autho^{+}(S, O, A) \rceil^{0} \;\mapsto\; Autho(S, O, A)$$

Access decisions have a positive outcome, when the policy defines a positive authorisation, and no negative authorisation can be derived for this access.

The Examples 1 to 3 captured the activity based and state based authorisation case. However, policies like the Chinese Wall Policy [2] require information on the history. The Chinese Wall policy is formalised in our model in Example 4.

Example 4 (Chinese Wall) *"Once a subject has accessed an object, the only other objects accessible by that subject are within the same company data-set or within a different conflict of interest class". [2]*

$$\begin{array}{l} \Diamond Do(S, O, access) \wedge \\ dataset(O) = dataset(O') \end{array} \;\mapsto\; Autho^{+}(S, O', access)$$

Here the predicate $Do(S, O, A)$ denotes that the subject S is performing action A on object O. $dataset(O)$ denotes the company data-set the object O belongs to.

$$\begin{array}{l} \Diamond Do(S, O, access) \wedge \\ ciclass(O) \neq ciclass(O') \end{array} \;\mapsto\; Autho^{+}(S, O', access)$$

Here $ciclass(O)$ denotes the conflict of interest class the object O belongs to.

The example shows how the conflict of interest can be expressed in the model using the derived temporal operator *sometime* ($\Diamond$). The interesting property of the policy is that it leaves the initial choice, which information to access (i.e. O), to the user and is then subsequently limiting the access for this user, based on this initial choice.

The Chinese Wall policy does not address the issue of changing conflict of interest classes over time. We feel that the ability to model these changes is important. A company may enter competition to another company, that was before in the same conflict of interest class. In this case it becomes more difficult, to decide whether all information is made inaccessible, or whether the policy needs to be refined to express the more complex relationships between the information in the classes. The actual policy does highly depend on the requirements of the companies under concern. In the following we provide three examples of policies in this situation.

Example 5 *A subject should be granted access to other objects, when these objects **are now** within a different conflict of interest class.*

$$\begin{array}{l} \Diamond Do(S, O, access) \wedge \\ \mathsf{fin}\,(Ciclass(O) \neq Ciclass(O')) \end{array} \;\mapsto\; Autho^{+}(S, O', access)$$

To denote the ability of conflict of interest classes to change over time, we write the function $Ciclass(O)$ with an uppercase initial letter (See Section III). The operator fin does denote the final state of the reference interval. For the interpretation of rules this is the state in which the access decision is made (See Definition 1).

Example 6 *A subject should be granted access to other objects, when these objects **have always been** within a different conflict of interest class.*

$$\begin{array}{l} \Diamond Do(S, O, access) \wedge \\ \Box(Ciclass(O) \neq Ciclass(O')) \end{array} \;\mapsto\; Autho^{+}(S, O', access)$$

Here the derived temporal operator always *($\Box$) denotes that the state formula, describing the inequality of the conflicting interest classes, has held over all suffix-intervals.*

Example 7 *A subject should be granted access to other objects, when these objects **have always been** within a different conflict of interest class, **since the first object has been accessed**.*

$$\begin{array}{l} \Box(\neg Do(S,O,access))\ ;\ \mathsf{skip};\\ Do(S,O,access)\ \wedge \qquad\qquad \mapsto\ Autho^{+}(S,O',access)\\ \Box(Ciclass(O) \neq Ciclass(O')) \end{array}$$

Whenever the reference interval can be decomposed in an initial interval in which subject S did never access O and a suffix interval[2] *in which S accessed O in the initial state and from thereon O and O' were always in distinct conflict of interest classes, then S is granted access to O'.*

The Examples 1 to 7 show the versatility of the security model to express authorisation requirements, through the expression of behaviour in the premise of rules. Actions, as presented in the model, are not parametrised. Parameters can be formally seen as an abbreviation for many concrete instances of the action. They are a convenient way of describing a large, but finite set of actions. We will therefore use parametrised actions in the next section, where we show how delegation rules are expressed in the model using authorisation rules.

C. Delegation

Delegation is the process of transferring an access right from one subject to another subject, to act on his behalf. Sloman states that

> There is a need to control to whom the managers [the subjects] can delegate and what operations they can delegate. This type of policy requires two subject domains — for the delegator and delegatee. [7]

This leads to two questions that a delegation model needs to address, controlling the access to delegation and the effect that delegation has on future access control decisions. We first show how delegation rules can be expressed as authorisation rules with parametrised actions.

1) Controlling delegation: Delegation can be seen as an action, that transfers an access right from the delegator to the delegatee. It can therefore be expressed as a parametrised action of the following form.

$$delegate(delegator, delegatee, object, action)$$

To express the first requirement for delegation, we use authorisation rules to express who is allowed to delegate.

$$f\ \mapsto\ Autho^{+}(S, DM, delegate(D, D', O, A))$$

The above rule state that the subject S is authorised to perform the parametrised action $delegate(D, D', O, A)$ on the object DM that represents the delegation monitor. The parameters are as in the delegation action definition, D the delegator, D' the delegatee, O the object, A the action.

The delegation monitor is a component of the trusted computing base, that keeps track of which rights have actually been delegated to other subjects. This captures the *effect* of the delegation and influences future access control decisions as described in section IV-C.2.

[2]the *skip* describes that both intervals do not share a common state

To avoid recursion we disallow the delegated action to be a delegate action itself. We feel that this does not represent a major limitation to the model. If there should need arise, this restriction can be relaxed, by limiting the amount of cascaded delegations to a finite number.

It is generally assumed that rights can only be delegated by the delegator himself, but we feel that this form of restriction should be explicitly mentioned in the delegation policy. The delegation rule in Example 8 makes this requirement explicit.

Example 8 (Delegation) *A subject can only delegate rights from itself to other subjects.*

$$\mathsf{true}\ \mapsto\ Autho^{+}(S, DM, delegate(S, D', O, A))$$

Here S is both subject and delegator, ensuring above requirement.

Other requirements w.r.t. delegation may be time related and state that a delegated right is only valid for a certain amount of time. There are different ways to express this requirement in our model, via obligation as described later in Example 13, or by constraining the right to delegate in the delegation policy as in Example 9.

Example 9 *No subject may delegate a right longer that 10 time-units.*

$$\lceil \Box DM.deleg(D, D', O, A) \rceil^{10} \\ \mapsto\ Autho^{-}(D, DM, delegate(D, D', O, A))$$

Here $DM.deleg(D, D', O, A)$ represents a query to the delegation monitor testing if the right (O,A) has been delegated by D to D'. The rule then states an explicit denial to delegate. That this does indeed have the same effect as revoking the delegated right will become clear in Example 11.

A delegation action is not sufficient. A delegation policy must also address the issue of withdrawing rights that have been delegated. We model revocation of rights in the same way as delegation via a parametrised action $revoke(D, D', O, A)$.

Whilst the assumption that only subjects can delegate their own rights to others (Example 8) is reasonable for the delegation of rights, policies for revocation should be aware of the potential of collaborative intrusions. By collaborative intrusion we mean that more than one subject is involved in the attack and rights have been delegated between the attackers. In such a scenario it is sensible, that for example security administrators have the right to explicitly revoke a delegated right. In some settings it may be reasonable to allow superiors to revoke delegated rights of their team-members.

Having described, how delegation and revocation of rights can be controlled using authorisation rules, we now move on to show the effect delegation has on the access control decision.

2) Delegation and Access Control Decisions: Authorisation rules determine whether a subject is allowed to perform a specific action on an object. The subject may not hold this right himself, but has been delegated the right by another subject. This case needs to be taken into account, when specifying the decision rules for authorisation.

We first need to define the effect of the delegation action. Whenever a delegation action has been successfully performed, the query to the delegation manager (DM)

$$DM.deleg(D, D', S, O)$$

will result in true from that state onward, until the right is revoked. Whenever a revocation action has been successfully performed, the corresponding query to the delegation manager will from there on result in false, until the right is delegated again. Due to space limitations we are not presenting the formalisation of the delegation manager in this paper.

There are different views of what actually constitutes delegation, and many can be expressed in our policy model. We will give examples of three different notions and their effect on access control decisions.

Example 10 *When a right is delegated, it is permanently granted to the delegatee. It does not affect future access control decisions whether the delegator looses the right (authorisation) or looses the right to delegate (delegation). The right remains with the delegatee until it is explicitly revoked. This form of delegation is often found in data-base systems.*

$$DM.deleg(D, D', S, O) \;\mapsto\; Autho(D', O, A)$$

This form of delegation is simple to enforce because no additional checks need to be performed when the access is actually made.

Example 11 *When a right is delegated, the access decision is based on the premise, that the delegator holds the right himself and is allowed to delegate this right at the time of the access check.*

$$\left\lceil \begin{array}{l} DM.deleg(D, D', O, A) \wedge \\ Autho(D, O, A) \wedge \\ Autho(D, DM, delegate(D, D', O, A)) \end{array} \right\rceil^{0} \;\mapsto\; Autho(D', O, A)$$

The advantage of this notion of delegation is, that if the original delegator of that right is denied the access himself due to changes in the policy, the delegated right is automatically deactivated until the delegator is himself granted the right again. If for example a fraud investigation demands a change in the access control policy to ensure that the main suspect cannot delete traces of his wrong-doings, this mechanism does ensure that none of his, until then undetected accomplice, can do so on his behalf.

The next delegation example requires two rules in combination. How rules are combined into one policy is described in section IV-E.

Example 12 *Similar to Example 11, when a right is delegated, the access decisions based on that right are made on the premise, that the delegator holds the right itself and is allowed to delegate this right at the time of the access check* ***and the delegatee is not explicitly denied the right****.*

$$\left\lceil \begin{array}{l} DM.deleg(D, D', O, A) \wedge \\ Autho(D, O, A) \wedge \\ Autho(D, D', O, A) \end{array} \right\rceil^{0} \;\mapsto\; Autho^{+}(D', O, A)$$

$$\left\lceil \begin{array}{l} \neg Autho^{-}(S, O, A) \wedge \\ Autho^{+}(S, O, A) \end{array} \right\rceil^{0} \;\mapsto\; Autho(S, O, A)$$

Here the combination of the above rule with a hybrid authorisation policy yields the required result.

The combination of delegation rules and authorisation rules yields an expressive framework for the specification of access control requirements. In the following section we focus on a conceptually different class of requirements: Obligations.

D. Obligation

Obligation rules are fundamentally different to authorisation and delegation rules: they do not deal with access control. Obligation rules describe, what a subject *must* and *must not* do. For obligation we take an approach that is similar to the one used for delegation. We define the obligation manager (OM) as a component of the TBC. The OM is tracking which subject has which obligations and whether it complied with its obligations.

The OM can be queried, to check whether a subject S has the obligation to perform an action A on an object O, in a similar way the delegation monitor is queried.

$$OM.oblig(S, O, A)$$

Whenever a subject has a specific obligation, this is reflected by the above query returning true. When the obligation is fullfilled, the query will return false until the obligation arises again.

It is important to note, that an obligation for a subject denotes its responsibility to perform the action on the object. It does not necessarily mean that the subject must perform the action itself. This distinction is important in any system that allows delegation.

Example 13 (Obligation to revoke a right) *The delegator of a right is obliged to revoke the delegated right after 10 time-units.*

$$\lceil \Box DM.deleg(D, D', O, A) \rceil^{10} \;\mapsto\; Oblig(D, DM, revoke(D, D', O, A))$$

This rule does only state, that the subject is responsible that the revoke action is performed on the delegation manager in question. It does not in any way enforce the rule as such.

Since the full investigation of enforcement requires a concrete description of the underlying computational model we

will not further detail on the point in this paper, but refer the reader to [6] for examples on the enforcement of access-control policies. However, it is of interest to see how the security policy itself can express rules to punish the non-conformance with a subjects obligations. Example 14 shows how obligations can be enforced using access-control mechanisms.

Example 14 *Whenever a subject did not comply with its obligations within 5 time-units, the subject is denied to perform any other action before complying with the obligation.*

$$\lceil(\Box oblig(S,O,A)) \wedge O' \neq O \wedge A' \neq A\rceil^5 \mapsto Autho^-(S,O',A')$$

This does not ensure, that the subject is complying with its obligation, but prevents it from doing anything else in the system.

In a more realistic scenario, the restrictions are likely to be less drastic and are highly dependent on the obligation and its criticality. For example it would be sensible to allow the subject to delegate the right to another subject.

Obligations can also lead to conflicts. For example there can be the case that a subject is obliged to perform an action, but can never obtain the right to do so. Additionally a subject may have several obligations at a time. Here a precedence relation between obligations can be established to resolve the conflict. This is however highly dependent on the mechanisms used in the enforcement and will therefore not be discussed here in greater detail.

E. Simple Policies

A simple policy represents a collection of rules. The simple policy forms the basic building block for the composition of policies discussed in the subsequent sections. We write a simple policy as a set of rules. When we say a simple policy is enforced, it means that all rules that are in the policy apply in conjunction.

The following example shows the rules from Example 1 to 3 combined as a simple policy.

Example 15 *A simple policy that captures the no read-up rule of the BLP policy.*

$$\left\{\begin{array}{r} true \mapsto Autho^+(S,O,read), \\ Level(O) > Clearance(S) \mapsto Autho^-(S,O,read), \\ \neg Autho^-(S,O,A) \wedge Autho^+(S,O,A) \mapsto Autho(S,O,A) \end{array}\right\}$$

Siewe et.al. defined in [19] operations to manipulate simple policies and showed that the set-theoretic operations are applicable. We will not discuss these operations here, but refer the reader to [19]. At the heart of our model is the composition of simple policies, which is discussed in the following sections, first presented in [19].

V. Policy Composition

Under policy composition we understand the process of composing policies out of other, less complex policies. Simple policies represent the unit elements of the composition. We distinguish between two forms of composition: Event (and Time) -based Composition and Structural Composition. The benefits in both cases are that smaller policies, which can be analysed and comprehended easier, are combined to express more complex requirements. The composition-operators are sound and can be used to prove properties of the composition. Table I summarises the different operators.

Temporal Compositions		
Operator	Description	Reference
p	Simple Policy	Section IV-E
$P\,;Q$	Sequence	Section III-.1
$P^\frown Q$	Weak Sequence	Section III-.2
$\langle w\rangle P$	Unless	Definition 3
$[w]P$	As Long As	Definition 2
$\lceil P\rceil^t$	Timed	Section III-.2
P^+	Iteration	Section III-.2
$P^\oplus$	Weak Iteration	Section III-.2
$P \sqcap Q$	Non-deterministic Choice	Section III-.2
$w?P:Q$	conditional Choice	Section III-.2
Structural Compositions		
$(S,O,A):P$	Scope	Section V-B
$P \mid\mid Q$	Strict Parallel	Definition 6
$P \bowtie Q$	Join	Definition 7

TABLE I
Operators for Policy Composition

Where p denotes a simple policy, w denotes a state formula, t a natural number, P, Q and R range over policies. S, O and A are sets of subjects, objects and actions respectively. The syntax and semantics of the event and time-based operators is given in the following definitions.

A. Event and Time-based Composition

Often security requirements distinguish clearly between different situations and conditions of the system in question. For example HIPAA [20] distinguishes between rules that apply under national emergencies and rules that apply otherwise. It is natural to capture both cases in different policies and to apply each policy when required. Many more examples can be found, for example library access in universities, traffic shaping policies are dependent on the congestion status, etc. Backes et.al. also mention sequential composition of policies as a further extension to their algebra to compose enterprise privacy policies [21].

The syntax of the sequential composition of policies is denoted by $P;Q$ with the semantics described in section III-.1. Sequentially composed simple policies will typically not agree on the shared state. Here the weak sequential composition $P^\frown Q$ is favourable (see III-.2). To capture the change of policy as the result of an event, we define the following operators:

Definition 2 (As Long As: $[w]P$) *The system is governed by policy P as long as w holds.*

$$\begin{aligned}\mathcal{M}([w]P) \mathrel{\hat{=}} & (\mathcal{M}(P) \wedge \Box w) \vee \\ & (((\mathcal{M}(P) \wedge \Box w); skip) \wedge fin\ \neg w) \vee \\ & (empty \wedge \neg w)\end{aligned}$$

Definition 3 (Unless: $\langle w \rangle P$) *The system is governed by policy P unless w holds. The state formula w can here indicate the occurrence of an event.*

$$\mathcal{M}(\langle w \rangle P) \mathrel{\hat{=}} \mathcal{M}([\neg w]P)$$

The operators *as-long-as* and *unless* are used to define the event dependency of policies. Whilst the former defines criteria that must be maintained for the policy to apply the latter defines conditions/events, on which the policy stops applying. Together with the operator $\lceil P \rceil^t$ they are used to define the temporal scope of a policy and thus define the time of the dynamic change. Figure 3 depicts the informal semantics of the operators *unless* and *as-long-as*.

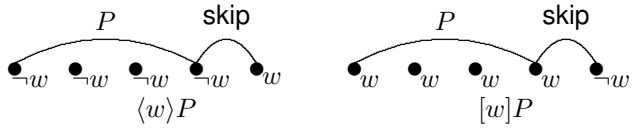

Fig. 3. Unless and As-Long-As

Having defined the sequential composition and operators to define the temporal scope of a policy we can express the dynamic change of polices, as mentioned in the beginning of this section.

Example 16 *Suppose the HIPAA privacy rules that are applied in the case of a national emergency are captured by the policy Q and the rules that apply otherwise in policy P. We can express the requirement for a dynamic change in policy on the event that a national emergency is declared by the following composed policy.*

$$\langle emergency() \rangle P \,;\, Q$$

where the predicate $emergency()$ indicates whether a national emergency is declared or not.[3]

The advantage is the level of modularity, and the ease of maintenance. If, for example, the legal requirements change for the emergency case only, it is sufficient to modify the policy Q. The administrator requires only assurance, that the new rule did not introduce conflicts in Q, as the rules in P are independent. Additionally the policy is more comprehensible, due to the fact that the separation is an explicit part of the policy and not encoded in complex rules. To understand the normal mode of operation it is sufficient to study policy P.

[3]We assume, that the system that the mechanisms that are enforcing the policy are capable of observing the event.

Often sequences like the one defined in Example 16 are repeated. The iteration of policies can be expressed using the compositions P^+ and $P^{\oplus}$ with their semantics given in III-.2.

Example 16 captured the dynamic change of policy in case of a national emergency, but did fail to capture that the requirement actually states that these change is constantly repeating. Example 17 shows the extension to capture the iterative behaviour.

Example 17 *Following on from the previous Example 16, we want to capture that the policy Q applies as long as national emergency is declared, and that afterwards the policy P is restored.*

$$(\langle emergency() \rangle P \,;\, [emergency()]Q)^+$$

We have shown how sequential composition and iteration of policies can be used to express the dynamic selection of policies over time. It is also important to be able to select policies based on some condition, or even leave the choice non-deterministic. The operators $w?P : Q$ and $P \sqcap Q$ allow this form of composition (see III-.2).

One of the motivation for non-deterministic choice is to decrease the predictability of system behaviour, for an external observer. This is especially important for systems that are developed for the military domain, where every predictable behaviour is a potential point of exploitation by the enemy [22].

The operators discussed up to now deal with policy composition along the temporal axis. Policies by default apply to the whole system, but to be able to restrict their scope, and to be able to form hierarchies of policies other compositions are required. These are introduced in the sequel.

B. Introducing Scope

Policies, by default apply to all subjects, objects and actions in the system. Before we introduce hierarchical policies, it is important to define an explicit scope for a policy. This allows the administrator to limit the domain on which the policy is enforced.

Definition 4 (Scope) *We denote the scope of a policy by:*

$$(S, O, A) : P$$

where S is the set of subjects, O is the set of objects and A is the set of actions to which the policy P applies. Scope represents a bounded *universal quantification.*

Formally we treat S, O, A as variables in a policy. These are then assigned by the scope operator to concrete values. The completeness algorithm presented in [6] is adapted to quantify over these variables, rather than the universal sets.

We denote the test whether a specific access control triplet (x, y, z) is within a specific scope (S, O, A) by $(x, y, z) \in (S, O, A) \mathrel{\hat{=}} x \in S \wedge y \in O \wedge z \in A$. Other set operations can be defined on scopes in a similar fashion.

C. Structural Composition

Policies with different, potentially overlapping, scopes can apply at the same time. Given the semantics of rules (See Definition 1), no conflict can arise in the mathematical sense when two policies are conjuncted (using $\wedge$ semantic). However, from a specification point of view the conjunction of policies represents a widening of the access rights granted by the policy (as the conjunction of policies is a conjunction of implications, see Section IV-E and 1). We take the view that widening of access should only occur when explicitly stated by the administrator and therefore introduce different operators to capture precedences between policies.

To be able to restrict the parallel composition it is necessary to capture the behaviour of the policies P and Q locally and define how conflicts can be resolved. Conflicts are for example, P allows the access and Q denies the access when enforced individually. Definition 5 defines the generic parallel composition of two policies.

Definition 5 (Generic Parallel Composition) *The generic parallel composition of policy $sc_P : P$ and policy $sc_Q : Q$ under the conflict resolution rule r is defined as follows.*

$$
\begin{aligned}
&\mathcal{M}(sc_P : P \,||^r\, sc_Q : Q) \mathrel{\hat{=}} \\
&\quad \forall\ (s_P, o_P, a_P) \in sc_P \wedge (s_Q, o_Q, a_Q) \in sc_Q \ \cdot \\
&\quad \exists\ Autho_P(s_P, o_P, a_P), Autho_Q(s_Q, o_Q, a_Q) \\
&\qquad Autho^+_P(s_P, o_P, a_P), Autho^+_Q(s_Q, o_Q, a_Q) \\
&\qquad Autho^-_P(s_P, o_P, a_P), Autho^-_Q(s_Q, o_Q, a_Q) \ \cdot \\
&\quad \mathcal{M}(P[Autho_P/Autho]) \wedge \mathcal{M}(Q[Autho_Q/Autho]) \wedge \\
&\quad \left(\begin{array}{r} \lceil (x,y,z) \in sc_P \cap sc_Q \wedge Autho_P(x,y,z) \wedge \\ Autho_Q(x,y,z) \rceil^0 \mapsto Autho(x,y,z) \end{array} \right) \wedge \\
&\quad r
\end{aligned}
$$

where sc_P and sc_Q are the respective scopes of policy P and Q. $f[y/x]$ denotes that every occurrence of x in f is replaced by y.

The effect of the policies is captured in the local Boolean state variables $Autho_P$ and $Autho_Q$ with the respective superscripts for positive and negative authorisation. By rewriting the $Autho$ in both policies P and Q their effect is captured in the local variables $Autho_P$ and $Autho_Q$ respectively. The general parallel composition defines the strictest form of parallel composition, by allowing access only if the access is allowed by both policies P and Q, as stated by the rule in above Definition 5. The additional conflict resolution rule r may then grant additional access, as shown in subsequent definitions.

Definition 6 (Strict Parallel) *Strict parallel composition of policy P and Q contains only those authorisations, that are stated by both policies.*

$$\mathcal{M}(P \,||\, Q) \mathrel{\hat{=}} \mathcal{M}(P \,||^{true}\, Q)$$

The motivation behind this restrictive definition originates from the semantics of the model. Rules represent a form of implication and it is therefore always possible to widen the access rights granted by a policy by adding (conjoining) an additional rule. However, the contrary cannot be achieved as negation on the right-hand side of rules is not permitted. As a result the *strict* definition is introduced as a base from which less restrictive parallel compositions are derived.

Definition 7 (Join) *Defines, that both policies need to agree on authorisations in the intersection of their scopes, but for the disjunction the outcome of the respective policy is taken.*

$$
\begin{aligned}
&\mathcal{M}(sc_P : P \bowtie sc_Q : Q) \mathrel{\hat{=}} \mathcal{M}(sc_P : P \,||^r\, sc_Q : Q) \\
&\textit{with } r = \\
&\lceil (s,o,a) \in sc_P \setminus sc_Q \wedge Autho_P(s,o,a) \rceil^0 \mapsto Autho(s,o,a) \\
&\wedge \\
&\lceil (s,o,a) \in sc_Q \setminus sc_P \wedge Autho_Q(s,o,a) \rceil^0 \mapsto Autho(s,o,a)
\end{aligned}
$$

The operators for parallel composition can be used to structure policies according to organisational hierarchies as proposed in e.g. [9]. Besides the better reflection of organisational structures, the parallel composition can also be used to define extension points within the policy. By extension points we mean that our model does allow to *define* when and what part of the policy can be updated. This allows for the integration of management requirements for policy evolution. The following example gives an illustration of this.

Example 18 (Policy Evolution) *When defining a dynamic policy we are aware of some of the changes in the policy that are triggered by known events (See Section V). We capture these requirements in the composed policy taken from Example 17 and express with the joint policy R the part of the policy that is not affected by a national emergency.*

$$R \bowtie (\langle emergency() \rangle P \,;\, [emergency()] Q)^+$$

However, it is often the case that system administrators need to react to events that where not known at the time the initial policy was developed. As stated in the introduction it is important that the policy can evolve, i.e. administrators can substitute the policy against another more adequate policy.

Since such updates are especially important in emergency situation and cannot easily take place otherwise we replace the policy Q by the following.

$$(< Done(S, TBC, upload(X)) > Q \,;\, X)^+$$

Where X is a variable that is not defined until the file containing the new policy is uploaded. $Done(S, O, A)$ denotes that subject S finished performing action A on object O. On this event the policy changes to the newly uploaded policy X.

The advantage of this approach is, that management policies can be written that determine, who can upload new policies and the effect the new policy has can be restricted by the definition of the extension point in the original policy. In Example 18, only the emergency policy can be updated and the

update cannot interfere with policies that apply under normal conditions.

VI. Tool-support for Policy Specification

As we noted in the introduction, tool-support for specification and analysis is a key factor to the success of a policy-framework. For the presented model we are developing the SANTA workbench, that allows the specification of policies, their high-level analysis and their verification at run-time.

To support the specification of dynamic policies we developed a policy editor that takes advantage of the model's compositionality. The policy editor allows the composition of policies in the form of structural diagrams. Key functions that increase the usability for more complex compositions are for example *folding*, i.e. policy components can be collapsed, and *zooming*.

For the analysis SPAT (Security Policy Analysis Tool) has been developed [6], [23]. In SPAT policies and their compositions are written in Tempura [24] and executed. The tool supports querying and the visualisation of access control decision that where determined by the policy. Decisions can be traced back to the rules that had influence, allowing for a better comprehension and maintainability of the policy.

VII. Conclusion and Future Work

We highlighted in this paper the benefits of composition and presented a policy frame-work, that allows to compose polices hierarchically as well as along the temporal axis. We presented the expressiveness of the model by specifying variations of well known policies, in which policy decisions are based on the history of execution. The examples showed how the framework is applied to the specification of authorisation, delegation and obligation policies and how policy composition can be used to define policies for systems that are operating in an environment that is characterised by a high degree of uncertainty. Here dynamic policies can express changes in the policy that are anticipated and those that are unknown at the time the policy has been designed (Example 18) . We briefly presented tool-support that has been used in the specification and analysis of dynamic policies taken from different domains, such as Electronic Patient Records (EPR), Traffic Shaping Policies or Resource Control in Military Networks.

Our future work will concentrate on the enhancement of the provided tool-support for specification and analysis. Additionally we are investigating the integration of trust in our policy model. This allows to link events with trust-revision and to define dynamic changes based on revised trust-levels, capturing the effect of trust-revision.

References

[1] L. J. LaPadula and D. E. Bell, "Secure Computer Systems: Mathematical Foundations," MITRE Technical Report 2547, Tech. Rep., 1973.

[2] D. Brewer and M. Nash, "The Chinese Wall Policy," in *IEEE Symposium on Research in Security and Privacy*. Oakland, California, USA: IEEE, May 1989, pp. 206–214.

[3] R. Anderson, "Security in clinical information systems," for British Medical Association (BMA), Computer Laboratory University of Cambridge Pembroke Street Cambridge CB2 3QG, Tech. Rep., January 1996. [Online]. Available: http://www.cl.cam.ac.uk/ rja14/Papers/policy11.pdf

[4] A. Cau, B. Moszkowski, and H. Zedan, "The ITL homepage," online. [Online]. Available: http://www.cse.dmu.ac.uk/STRL/ITL

[5] B. Moszkowski, "Compositional Reasoning using Interval Temporal Logic and tempura," in *Compositionality: The Significant Difference*, ser. LNCS, W.-P. d. Roever, H. Langmaack, and A. Pnueli, Eds., vol. 1486. Berlin: Springer Verlag, 1998, pp. 439–464.

[6] F. Siewe, "A Compositional Framework for the Development of Secure Access Control Systems," Ph.D. dissertation, Software Technology Research Laboratory, Department of Computer Science and Engineering, De Montfort University, Leicester, 2005.

[7] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, pp. 333–360, 1994. [Online]. Available: citeseer.ist.psu.edu/sloman94policy.html

[8] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian, "Flexible support for multiple access control policies," *ACM Trans. Database Syst.*, vol. 26, no. 2, pp. 214–260, 2001.

[9] P. Bonatti, S. Vimercati, and P. Samarati, "An Algebra for Composing Access Control Policies," *ACM Transactions on Information and System security*, vol. 5, no. 1, pp. 1–35, February 2002.

[10] M. Abadi and C. Fournet, "Access control based on execution history," in *10th Annual Network and Distributed System Symposium (NDSS'03)*. Reston, Virginia, USA: The Internet Society, February 2003, pp. 1–15. [Online]. Available: citeseer.ist.psu.edu/abadi03access.html

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[12] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 191–233, 2001.

[13] T. Mossakowski, M. Drouineaud, and K. Sohr, "A temporal-logic extension of role-based access control covering dynamic separation of duties," in *10th International Symposium on Temporal Representation and Reasoning / 4th International Conference on Temporal Logic (TIME-ICTL 2003)*. Cairns, Queensland, Australia: IEEE Computer Society, July 2003, pp. 83–90.

[14] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Specification Language," in *Workshop on Policies for Distributed Systems and Networks (Policy2001)*, January 2001.

[15] J. Hoagland, "Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on Objects," Ph.D. dissertation, University of California, 2000.

[16] B. Moszkowski, "Some very compositional temporal properties," in *Programming Concepts, Methods and Calculi*, ser. IFIP Transactions, E.-R. Olderog, Ed., vol. A-56, IFIP. Elsevier Science B.V. (North–Holland), 1994, pp. 307–326.

[17] H. Zedan, A. Cau, and S. Zhou, "A Calculus for Evolution," in *Proc. of The Fifth International Conference on Computer Science and Informatics (CS&I'2000)*, 2000.

[18] D. Bell and L. Lapadula, "Secure computer system unified exposition and multics interpretation," MITRE, Bedford, MA, Tech. Rep. MTR-2997, 1975.

[19] F. Siewe, A. Cau, and H. Zedan, "A Compositional Framework for Access Control Policies Enforcement," in *ACM Workshop on Formal Methods in Security Engineering (FMSE'03)*, M. Backes, D. Basin, and M. Waidner, Eds. Washington, DC: ACM Press, October 2003, pp. 32–42.

[20] U. D. of Health & Human Services, "HHS - Office for Civil Rights - HIPAA," 2004. [Online]. Available: http://www.hhs.gov/ocr/hipaa/

[21] M. Backes, M. Durmuth, and R. Steinwandt, "An Algebra for Composing Enterprise Privacy Policies," in *Proceedings of 9th European Symposium On Research in Computer Security (ESORICS)*, P. Samarati, D. Gollmann, and R. Molva, Eds., September 2004, pp. 33–52.

[22] P. Beautement, D. Allsopp, M. Greaves, S. Goldsmith, S. Spires, S. Thompson, and H. Janicke, "Autonomous Agents and Multi–Agent Systems (AAMAS) for the Military - Issues and Challenges," in *Proceedings of 1st Workshop on Defence Applications for Multi-Agent Systems (DAMAS)*, ser. Lecture Notes in Computer Science, Robert Ghanea-Hercock and Mark Greaves and Nick Jennings and Simon Thompson, Ed., vol. 3890. Utrecht, The Netherlands: Springer, July 2005, pp. 1–13.

[23] H. Janicke, F. Siewe, K. Jones, A. Cau, and H. Zedan, "Analysis and Run-time Verification of Dynamic Security Policies," in *In Proceedings of The First Workshop on Defence Applications for Multi-Agent Systems (DAMAS'05)*, ser. Lecture Notes in Computer Science, Robert Ghanea-

Hercock and Mark Greaves and Nick Jennings and Simon Thompson, Ed., vol. 3890. Utrecht, The Netherlands: Springer, July 2005, pp. 92–103.

[24] B. Moszkowski, *Executing Temporal Logic Programs*. England: Cambridge University Press, 1986.