Specifying Policies Using UML Sequence Diagrams – An Evaluation Based on a Case Study

Bjørnar Solhaug[1,2], Dag Elgesem[1] and Ketil Stølen[2,3]
[1]*Dep. of Information Science and Media Studies, University of Bergen*
[2]*SINTEF ICT* [3]*Dep. of Informatics, University of Oslo*
Email: {bjors,kst}@sintef.no, dag.elgesem@infomedia.uib.no

Abstract

This paper provides a case study based evaluation of UML *sequence diagrams as a notation for policy specification. Policy rules are defined on the basis of deontic logic and provided a trace based semantics interpreted over Kripke structures. This gives a semantics comparable to the* UML *trace semantics for sequence diagrams, which is utilized in the evaluation. The focus is on requirements with respect to expressivity, utility and human readability.*

1. Introduction

The UML 2.0 [13] is currently the de facto standard for the modeling and specification of information systems. Policy frameworks, see e.g. [17] for a survey, are increasingly being adopted as a means to manage such systems with respect to security, ICT services and networks, business processes, etc., and an obvious issue to investigate is to what extent UML is suitable for the specification of policies for the management of these systems.

A much referred to definition of a policy is the one provided by Sloman [16] who suggests that policies are rules governing the choices in the behavior of a system. The system may consist of human or automated actors, or a combination of the two, in addition to a set of resources such as information and services. There may, moreover, be a number of users external to the system whose access to the system resources should be constrained.

Policy rules can be understood as normative statements about system behavior, in particular when the actors to which a policy applies are human. A policy rule may for example state that all employees are obliged to lock their computer when leaving their working station, yet humans obviously having the choice whether to comply with the rule. Incentive structures can be implemented for the purpose of encouraging normative behavior, however not eliminating the potential, undesired behavior.

The same can be the case for automated actors outside the control of the system owner or designer. Automated actors within the system, however, do usually not have the possibility to disobey policies. The normative aspect is still present in the sense that the functionality of the actors span wider than what is prescribed by the policy. This is a key feature of policies as they "define choices in behavior in terms of the conditions under which predefined operations or actions can be invoked rather than changing the functionality of the actual operations themselves" [17]. By separating the policy from the system, system behavior can be governed by modifying the policy only, leaving the underlying implementation of the system unchanged [16].

We have hence on the one hand the system itself with all its potential behavior, and on the other hand the policy constraining the behavior of the system. In this paper we evaluate the suitability of deploying UML 2.0 sequence diagrams for policy specification.

The next section provides a classification of the type of policy rules that we aim at formalizing in this paper, and a trace based semantics is given for each type of rule. As UML sequence diagrams are explained in terms of traces in the standard [13], we are hence provided a policy rule semantics comparable to the sequence diagram semantics. Section 3 describes our target of evaluation. In Section 4 we present our evaluation method, followed by a set of success criteria in Section 5 describing the requirements that should be satisfied by sequence diagrams as a policy specification language. In Section 6 we present an eLearning scenario that will serve as a case study for the evaluation. Section 7 and Section 8 suggest how particular policy rules for the eLearning scenario can be specified using sequence diagrams. Based on the case study, Section 9 discusses the strengths and challenges of applying sequence diagrams for this purpose by evaluating whether the given success criteria are fulfilled. Finally, we will in Section 10 discuss existing policy specification approaches, specifically some of those using UML, before we conclude in Section 11.

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

2. Policy Classification

A reasonable way of classifying policies is to operate with the tripartition of policy rules into *obligations*, *permissions* and *prohibitions*. This classification is based on deontic logic [21] and several of the existing approaches to policy specification and classification have language constructs of such a deontic type [1, 7, 16, 19, 23]. This categorization is furthermore proposed in standardization works by ISO/IEC and ITU-T on open distributed processing [6].

In addition to prescribe constraints on behavior, a policy must express the conditions under which the various policy rules apply. We shall refer to these conditions as *policy triggers* and distinguish between *event triggers* and *state triggers*. An event triggered policy applies by the occurrence of a given event, whereas a state triggered policy applies in a given set of states. A policy trigger may be a combination of the two, i.e. the occurrence of a specific event in a given set of states. A similar way of classifying policy triggers is proposed by Sloman [16].

A policy must furthermore specify the actors to which the rules apply, i.e. the actors the behavior of which is constrained by the policy. These actors will be referred to as the *addressee* of the policy.

Given a behavior, a trigger and an addressee, we define the three different types of policy rules as follows: A permission (resp. obligation, prohibition) states that when the policy trigger executes or applies, the addressee is allowed (resp. required, forbid) to conduct the behavior.

Figure 1 shows a class diagram describing the elements of a policy and the relations between them.

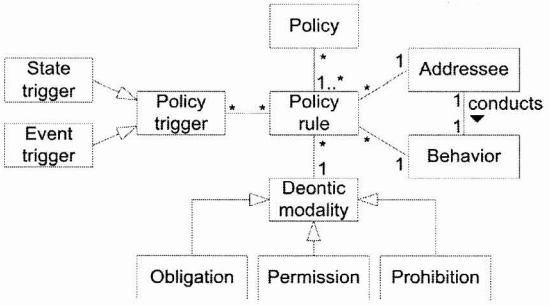

Figure 1. Policy classification

Trace semantics for deontic modalities. We will in the following establish a trace based semantics for deontic modalities. This will later be used as a means to decide whether the UML sequence diagrams have sufficient expressivity to capture these modalities.

Standard deontic logic (SDL), see e.g. [12], builds upon propositional logic and provides a semantics in terms of Kripke structures of possible worlds. A possible world w describes a possible state of affairs, and for each proposition p in the language, either p holds or the negation $\neg p$ holds, denoted $\vDash_w p$ and $\vDash_w \neg p$, respectively. Assuming a set of possible worlds W describing a set of different states of affairs, the binary relation A defines for each world $w \in W$ the set of worlds that is acceptable in w. Awv denotes that v is an acceptable world in w, and A^w denotes the set of worlds that is acceptable in w, i.e. $A^w = \{v : Awv\}$.

Let $\mathbf{OB}p$ denote "it is obligatory that p". The semantics of this statement is that if $\vDash_w \mathbf{OB}p$, then $\forall v \in A^w : \vDash_v p$. The semantics of the statement $\mathbf{PE}p$ denoting "it is permitted that p" is that if $\vDash_w \mathbf{PE}p$, then $\exists v \in A^w : \vDash_v p$. The semantics of the statement $\mathbf{PR}p$ denoting "it is prohibited that p" is that if $\vDash_w \mathbf{PR}p$, then $\neg\exists v \in A^w : \vDash_v p$.

The various deontic modalities can be expressed in terms of each other as shown by the definitions $\mathbf{PE}p \leftrightarrow \neg\mathbf{OB}\neg p$ and $\mathbf{PR}p \leftrightarrow \mathbf{OB}\neg p$. We shall in this paper not pay attention to the axioms and rules that define the syntax of SDL, apart from noticing the axiom $\mathbf{OB}p \rightarrow \mathbf{PE}p$ which states that everything that is obliged is also permitted. The validity of this axiom, i.e. that it holds in every possible world in all Kripke structures for SDL, is ensured by the requirement that the acceptability relation A on the Kripke structures is serial. Seriality means that for each possible world w, there is at least one acceptable world, i.e. $A^w \neq \emptyset$.

The term *trace* denotes an execution history describing how behavior evolves over time. A possible world is a description of a possible state of affairs which can be uniquely identified by the complete history leading up to that state; each world $w \in W$ hence uniquely corresponds to a specific trace. Since a policy specifies constraints on behavior, the acceptable worlds A^w follows w in time. This means that all states of affairs $v \in A^w$ is given by a trace such that the trace describing w is a prefix. We use the notation $t_1 \frown t_2$ to denote the trace given by the concatenation of the traces t_1 and t_2. If t_1 is finite, $t_1 \frown t_2$ is the trace where t_1 is the prefix and t_2 is the suffix and the length equals the sum of the length of t_1 and t_2. If t_1 is infinite, $t_1 \frown t_2$ equals t_1.

A possible world $v \in A^w$ then corresponds to the trace $w \frown t$ where t is the trace describing the history evolved from w to v. It should be noticed that interpreting traces over Kripke structures like this requires the structures to be forward branching with no loops.

A policy trigger describes the conditions under which a policy rule applies. In terms of the SDL semantics, the trigger refers to a set of possible worlds. Given our interpretation of a possible world as a trace, a policy trigger refers to a property of a trace, and the corresponding policy rule applies to all traces for which the property holds. If W is a set of possible worlds and T is a policy trigger, $W_T \subseteq W$ denotes the set of possible worlds for which the trigger holds.

With this interpretation of possible worlds and the relation between them in terms of traces, we are provided a Kripke semantics for deontic constraints that can be used in

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

the evaluation of sequence diagrams.

A policy rule refers to a certain behavior. Generally, the specification has a degree of abstraction or under-specification to it which means that the behavior can be conducted in various concrete ways. The given behavior hence corresponds to a set of traces describing how the execution history evolves immediately after the possible worlds for which the policy rule applies.

Assume, now, a policy rule for the set of worlds W, with a trigger T and a behavior B, where B is the set of traces each of which represents a concrete way of conducting the behavior. The trace based SDL semantics for the various modalities is then given as follows:
Permission: $\forall w \in W_T : \exists v \in A^w : \exists b \in B : v = w^\frown b$
Obligation: $\forall w \in W_T : \forall v \in A^w : \exists b \in B : v = w^\frown b$
Prohibition: $\forall w \in W_T : \forall v \in A^w : \neg\exists b \in B : v = w^\frown b$

3. Target of Evaluation

The target of evaluation is UML [13] sequence diagrams. Sequence diagrams specify interactions, which describe how messages are sent between entities for the purpose of performing a behavior. The question here is to what extent sequence diagrams are suitable as a policy notation. OCL is in our context used to impose constraints on the execution of interactions. Since OCL is a constraint language, it is a natural candidate for expressing aspects of policies [1, 14].

The UML 2.0 specification [13] provides an informal description of the trace semantics of interactions. A trace is a sequence of event occurrences ordered by time, describing how entities interact for the purpose of conducting a behavior. Interactions define a set of allowed traces and a set of forbidden traces, expressing desired and undesired behavior, respectively. This semantics is formalized with STAIRS [15] which will serve as the semantic basis for interactions in our evaluation.

Within the STAIRS framework, interactions characterize traces as positive, negative or inconclusive. If a trace is positive, the execution of the trace is valid, legal or desirable, whereas a negative trace means that the execution is invalid, illegal or undesirable. A trace is inconclusive if it is categorized neither as positive nor as negative, meaning that the trace is irrelevant for the interaction in question. By specifying policy rules using sequence diagrams and interpreting them in terms of the STAIRS semantics, we provide a policy specification that can be compared to the policy rule semantics given in Section 2 above.

4. Evaluation Method

Our evaluation is conducted over the following steps: (1) A carefully selected scenario is used as a case study; (2) A set of success criteria is formulated; (3) Sequence diagrams are deployed for specifying policies for the case study scenario; (4) The result of the latter step is evaluated against the success criteria.

The case study on which the evaluation is based is the Metacampus eLearning Enterprise Network described in a TrustCoM deliverable [3]. This eLearning scenario is well documented and was deployed during the TrustCoM project [20] for the purpose of testing and demonstrating the project's proposed framework for trust and contract management. As the scenario concerns real world issues, we are provided realistic management problems that can be addressed by means of policy frameworks.

The specific policy rules for the eLearning scenario that are described in this paper were captured through a systematic analysis arranged as a series of workshops involving people of various backgrounds, including security, law and computer science. The analysis was facilitated by the reuse of both documentation of the eLearning scenario and documentation of risk issues related to the scenario [11, 22][1].

5. Success Criteria

C1: *Sequence diagrams can express permissions*; **C2:** *Sequence diagrams can express obligations*; **C3:** *Sequence diagrams can express prohibitions*. The testing of the extent to which sequence diagrams meet these criteria amounts to examine to what extent there are language constructs available that allow specifications the semantics of which corresponds to the semantics for deontic modalities presented in Section 2.

C4: *The formalizations under criteria C1 through C3 respect the axioms and definitions of SDL.* SDL [12] is a normal modal logic distinguished by the axiom $\mathbf{OB}p \rightarrow \mathbf{PE}p$ stating that everything that is obliged is also permitted. The formalization of the deontic modalities in sequence diagrams should preserve the property expressed by this axiom as well as the definitions $\mathbf{PE}p \leftrightarrow \neg\mathbf{OB}\neg p$ and $\mathbf{PR}p \leftrightarrow \mathbf{OB}\neg p$.

C5: *Sequence diagrams allow the composition of deontic expressions.* A policy is built up of a set of policy rules. If each rule of a policy is specified as a sequence diagram, it should be possible to compose the resulting set of diagrams into one specification that represents the policy.

C6: *Sequence diagrams allow the specification of event and state triggers.* Policy triggers specify the conditions under which policy rules apply. In order to successfully express policies, we must be able to accompany the specification of a deontic statement with the specification of the relevant trigger.

C7: *Policies can be expressed in the spirit of UML.* By criterion C7 we mean that the various elements of a policy

[1] The mentioned deliverables [3, 11, 22] are available as downloads on the TrustCoM homepage [20].

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

are naturally and intuitively reflected by notions and constructs of UML sequence diagrams. Specifically, the elements of the class diagram of Figure 1 should have their corresponding language constructs.

C8: *Sequence diagrams allow the specification of policies in a manner suitable for engineers developing and maintaining systems that should adhere to the policies.* Maintainers, designers and other personnel that are responsible for modeling and implementing policies are facilitated by a language in which the various elements of a policy rule are easily captured. The specifications should be easily understandable in that the practitioners can, for each rule, recognize what kind of deontic constraint is represented, who or what is the addressee, what is the trigger and what is the behavior.

C9: *Sequence diagrams allow the specification of policies in a manner that is easy to understand for both decision makers and for the addressees of a policy.* Senior employees, chief executive officers and other decision makers are typically of non-technical background, and a suitable policy specification language should make allowance for this. The same is the case when the addressees of a policy are the employees of an organization or an enterprise. At an organizational level, a policy is usually a document in natural language. A specialized language for policy specification should compared to natural language structure, arrange and present the relevant pieces of information in a better way.

6. The eLearning Scenario

The case study scenario concerns a small to medium sized enterprise (SME) network specializing in providing tailored eLearning services to end-users [3, 11, 22]. An end-user connects to an eLearning portal which will offer access to a large variety of modularized and personalized learning resources. For each tailored learning path (package of learning content), a virtual organization is established for the provisioning of the eLearning services.

The actors involved are the end-users, the learning content providers (LCPs) and the eLearning portal operator (PO). The PO provides the interface towards the end-users and is responsible for the construction of learning paths. The training consultant (TC) is a software owned by the PO that identifies learning paths based on the customer needs and the learning resources made available by the LCPs. An LCP is typically a university or the like and provides modules that may serve as parts of learning paths.

We will specify a policy held by the PO for the purpose of protecting its core assets. The assets on which we will focus is the TC and its associated learning content ontology owned by the PO. In order to match an end-user's learning needs and requirements with the available learning content provided by the LCPs, the TC categorizes the learning content according to the ontology. Figure 2 shows a class diagram capturing the elements of the PO relevant for the policy specified in the sequel.

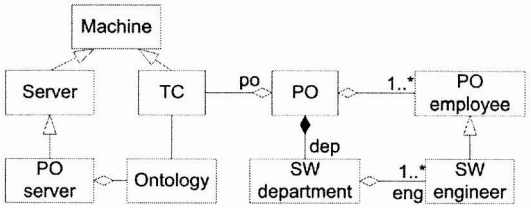

Figure 2. The portal operator

7. Specifying State Triggered Policies

A state triggered policy rule applies in a given set of states, viz. the triggering states. Figure 3 shows in a generic way our approach to specify state triggered rules. The interaction use below the trigger state refers to the behavior relevant to the policy rule, and the lifelines show the involved entities. The addressee is always represented, in addition to any number of other entities, here represented with one lifeline for illustrative purposes. An OCL invariant placed in an attached note imposes constraints on the behavior in accordance with some policy rule.

The triggering state can be represented as a guard or as an OCL invariant. These are Boolean expressions the truth value of which determines whether the execution of the interaction use behavior is legal.

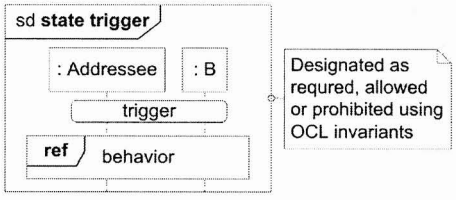

Figure 3. State triggered policy rule

State triggered permissions. A state triggered permission expresses that in a given set of states, the addressee is allowed to conduct the given behavior. We will here specify a permission that constrain the access to do configurations on the TC software. The functionality of the TC is critical, both to ensure that the learning paths suggested to the end-users match their needs and requirements, and to ensure that the selection of LCPs is done fairly, i.e. no LCP is discriminated.

Informally, the permission policy is stated as follows: *Access to configure the TC is granted to PO employees that are software engineers. TC configuration may occur between 10 p.m. and 4 a.m. only.* The latter clause is added to ensure that configurations are not done during office hours.

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

Figure 4 shows the sequence diagram in which this policy rule is expressed. The OCL operation oclIsTypeOf applies to all objects. For any object o, the expression o.oclIsTypeOf(t) evaluates to true if and only if o is of type t.

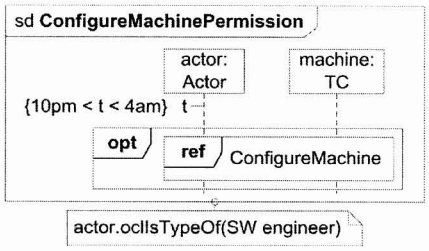

Figure 4. Access to configure the TC

By placing ConfigureMachine within an opt, the trace in which the actor skips the behavior is positive. As defined in Section 2, a permission only requires that the behavior is allowed, which means that specifying the behavior as optional is somewhat stronger than required. In practice, however, it is usually reasonable to interpret a permission as something optional. In the general case, a permission is not optional, for if it was we would be prevented from specifying a behavior as both a right and a duty. Nor would we preserve the SDL property that an obligation implies a permission.

The other positive traces are those in which ConfigureMachine is executed when actor is of type SW engineer and the time is within the required interval.

In a policy specification, the prohibitions and permissions together characterize the allowed behavior. There are three approaches to capture the allowed behavior, and we need to consider all three since the choice of which may have implications for how policy rules should be specified.

Firstly, we may assume that everything is prohibited unless explicitly specified as permitted. We shall call this approach "prohibition by default". Secondly, we may assume that everything is permitted unless explicitly specified as prohibited, and refer to this as "permission by default". The third approach is to explicitly specify all behavior as either permitted or prohibited. The choice of which approach to adopt usually depends on the type of system in question, an issue we will not pursue in this paper. The problem of detecting and resolving policy conflicts that may arise in the third approach is also outside our scope.

What we need to address is that if the approach is not prohibition by default, the permission specification must ensure that only the actors *explicitly* allowed to configure the the TC are permitted to do so. In our case, the constraint is given as an OCL invariant. The ConfigureMachine traces in which the client is not of the required type are hence negative. This means that the permission specification we suggest in Figure 4 implicitly is a prohibition also.

State triggered obligations. A state triggered obligation expresses that in a given set of states, the addressee is required to conduct the given behavior. Since the functionality of the TC is critical to the PO, a requirement on software testing is imposed: *When the TC has been configured, the PO software personnel is responsible for running tests on the TC software.* The addressee of this obligation is the SW department of the PO organization, see Figure 2, and their responsibility is to assign the task to one or more of their software engineers.

Basically, this obligation states that one particular behavior, viz. software configuration, requires another behavior to follow, viz. software testing. We will specify the given obligation policy as an OCL invariant. We assume the TC has a Boolean attribute configured that evaluates to true when the configuration is finalized.

Figure 5 suggests how this obligation can be specified. The OCL expression refers to navigations over associations in the class diagram of Figure 2. Navigations over associations results in collections the properties of which are accessed by an arrow "->" followed by the property name. In Figure 5, the exists operation is used. If s is a set, the expression s->exists(Boolean-expression) evaluates to true if the Boolean expression holds for at least one element of s.

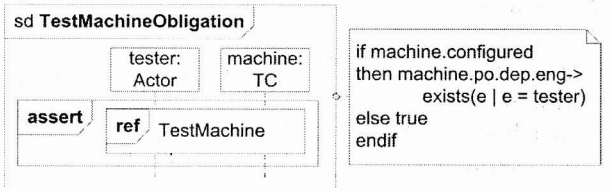

Figure 5. Software testing

The OCL invariant is in the form of what Castejón and Bræk [2] refer to as a role binding policy. It states the conditions under which there must be at least one software employee bound to the role of tester. We assume that the instantiation of a role or a lifeline implies the participation in the behavior. The same approach is found in [1].

By this assumption, all the traces in which a software engineer plays the tester role when machine.configured evaluates to true are positive. The negative traces are those in which machine.configured evaluates to true and there is no software engineer in the tester role. The assert operator furthermore characterizes all other traces than those corresponding to TestMachine as negative at this point of the execution.

Notice that the else clause is set to always evaluate to true. This means that there are no constraints when configurations have not been conducted; the TestMachine traces are positive when machine.configured evaluates to false.

State triggered prohibitions. A state triggered prohibition expresses that in a given set of states, it is forbidden for the addressee to conduct the behavior. We will here address separation of duty.

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

We have already specified policy rules with respect to the configuration and testing of the TC. The following requirement attend to the combination of the two activities: *If the TC has been configured, the software shall be tested by a software engineer different from the one having conducted the configuration.*

The combination of the two behaviors is expressed in Figure 6. There are two lifelines representing tester and conf, the latter the actor configurating the TC. Both roles are of type Actor, and there is nothing in the specification that prevents one Actor instance to play both roles simultaneously. The attached OCL invariant, however, ensures that the instances must be different: The diagram categorizes all traces given by concatenating a trace from ConfigureMachine and a trace from TestMachine in which conf and tester are instanced by the same entity as negative.

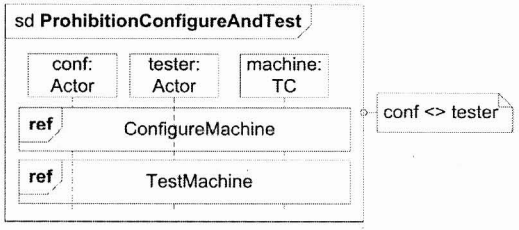

Figure 6. Separation of duty

Implicitly, this diagram specifies a permission also, since the given behaviors are allowed to be conducted should the OCL expression evaluate to true. It is hence irrelevant whether the approach is permission by default.

Notice, incidentally, that policy rules for ConfigureMachine and TestMachine are given above. These can be made explicit here by replacing the references to them in Figure 6 with references to ConfigureMachinePermission, cf. Figure 4, and TestMachineObligation, cf. Figure 5, respectively.

8. Specifying Event Triggered Policies

An event trigger can be represented as a receive event on the addressee lifeline as illustrated in Figure 7. The challenge is to find suitable ways of relating the trigger event and the given behavior. In addition to the addressee, there may be arbitrary many other roles involved. For illustrative purposes, we have shown only one additional lifeline.

Event triggered permissions. Event triggered permissions can be expressed by placing the relevant behavior immediately after the triggering event as illustrated to the left in Figure 8.

The sequence diagram characterizes all traces resulting from the concatenation of the event trigger and a trace corresponding to the behavior as positive. This way of specifying permissions suffice when the approach is prohibition by

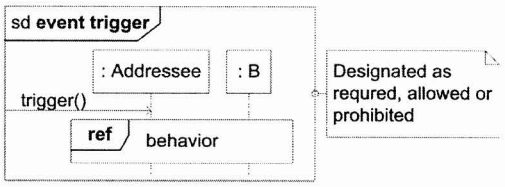

Figure 7. Event triggered policy rule

default. If the approach is that all behavior is to be explicitly specified as permitted or prohibited, we need to specify the set of traces corresponding to the behavior as negative if they do not follow immediately after the event trigger. This can be done by using the neg operator on the behavior.

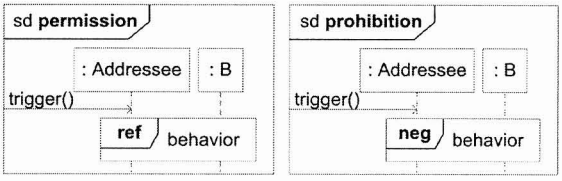

Figure 8. Permission and prohibition

Event triggered obligations. Event triggered obligations can be expressed by placing the required behavior within an assert fragment as shown to the left in Figure 9. The traces corresponding to the concatenation of the triggering event with the traces representing the execution of the behavior are then positive and all other traces are negative.

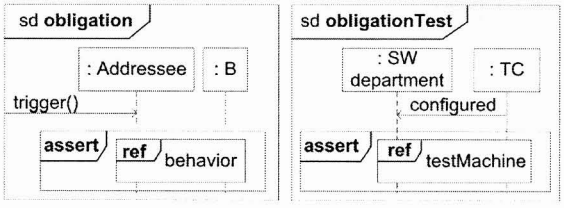

Figure 9. Event triggered obligations

As a concrete example of an event triggered obligation, we revisit the state triggered obligation of Figure 5. Here we assume that the SW department is alerted by the TC when a configuration has been conducted, as shown to the right in Figure 9.

Event triggered prohibitions. The neg operator is the obvious construct for expressing event triggered prohibitions, shown to the right in Figure 8 above. The semantics is a set of negative traces corresponding to the trigger event followed by the traces for the behavior. The only allowed trace is the one that solely consists of the reception of the trigger event.

If behavior is permitted by default it is sufficient to spec-

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

ify prohibitions only. If all behavior is to be explicitly categorized as one or the other, the behavior must be specified as permitted when the event trigger does not occur.

9. Evaluation

In this section we evaluate the results of the previous two sections against the success criteria in Section 5.

C1: Permissions. The first success criterion states that sequence diagrams can express permissions. The diagrams in Figure 4 and to the left in Figure 8 show our attempts to specify state triggered and event triggered permissions, respectively. With the trigger in place, the specifications are given as sequence diagrams that characterize the traces corresponding to the relevant behavior as positive.

The trace based interpretation of a permission over Kripke structures is that if a behavior is permitted in world w, then $\exists v \in A^w : \exists b \in B : v = w^\frown b$, where B is the set of traces representing the behavior. A sequence diagram characterizing the set B as positive, as do our permission specifications, corresponds to this interpretation.

By characterizing B as positive, none of these traces can consistently be characterized as negative, so the SDL implication $\mathbf{PE}p \rightarrow \neg\mathbf{PR}p$ holds.

Consider, now, the SDL implication $\mathbf{PE}p \rightarrow \neg\mathbf{OB}\neg p$. If this does not hold in our specifications, the trace set B can be characterized as positive while consistently using assert on the complement to B. The latter, however, characterizes B as negative, which is inconsistent with the permission. The implication hence holds.

Observe, importantly, that for the state triggered permission we exemplified, the specification is stronger than what is required since a prohibition is implicit. In Figure 4, the traces corresponding to ConfigureMachine are negative should actor be of a type other than as required by the invariant. In terms of our SDL trace semantics, this means that if a state triggered permission $\mathbf{PE}p$ holds in the set of worlds $W_T \subseteq W$, the implicit prohibition $\mathbf{PR}p$ holds in the complementary set of worlds $W \setminus W_T$.

An invariant is an assertion, and this can in some cases be desirable, but generally the user should be free to specify permissions independent of prohibitions and vice versa.

C2: Obligations. Criterion C2 states that sequence diagrams can express obligations. Our suggestions to specify state and event triggered obligations are shown in Figure 5 and Figure 9, respectively. Obligations are captured by applying the assert operator. This characterizes the traces corresponding to the behavior as positive, and all other traces as negative.

The interpretation of obligations over Kripke structures is that if a behavior is obligated in world w, then $\forall v \in A^w : \exists b \in B : v = w^\frown b$, B representing the behavior. Since the assert operator ensures that the traces corresponding to the behavior are the only allowed continuations at that point in the execution, our obligation specifications match the desired semantics.

As the complement to the set of traces representing the behavior is characterized as negative, the SDL implication $\mathbf{OB}p \rightarrow \mathbf{PR}\neg p$ holds. Characterized as negative, no element of this complementary set can consistently be characterized as positive, so $\mathbf{OB}p \rightarrow \neg\mathbf{PE}\neg p$ also holds. It is furthermore easy to see that the SDL axiom $\mathbf{OB}p \rightarrow \mathbf{PE}p$ holds.

C3: Prohibitions. The third success criterion states that sequence diagrams can express prohibitions. Figure 6 suggest how a state triggered prohibition can be specified by an OCL invariant on a sequence diagram. This specification characterizes the traces corresponding to the behavior as negative when the invariant evaluates to false. The diagram to the right in Figure 8 specifies an event triggered prohibition in which the traces corresponding to the behavior is characterized as negative by the neg operator.

The trace based interpretation of a prohibition over Kripke structures is that if a behavior is prohibited in world w, then $\forall v \in A^w : \neg\exists b \in B : v = w^\frown b$, B the traces representing the behavior. A sequence diagram characterizing the set B as negative, as do our prohibition specifications, corresponds to this interpretation.

Since B is negative, no element of B can consistently be characterized as positive, so the SDL implication $\mathbf{PR}p \rightarrow \neg\mathbf{PE}p$ holds. By characterizing B as negative, the set of positive traces is a subset of the complement to B. Hence the implication $\mathbf{PR}p \rightarrow \mathbf{OB}\neg p$ also holds.

Just like a state triggered permission implicitly specifies a prohibition, as observed above, state triggered prohibitions imply a permission: If the OCL invariant evaluates to true, the traces corresponding to the behavior are characterized as positive. Interpreted over Kripke structures, if a state triggered prohibition $\mathbf{PR}p$ holds in the set of worlds $W_T \subseteq W$, the implicit permission $\mathbf{PE}p$ holds in the complementary set of worlds $W \setminus W_T$.

C4: SDL axioms and definitions. Criterion C4 states that the formalizations under criteria C1 through C3 respect the axioms and definitions of SDL.

The axiom schema $\mathbf{OB}p \rightarrow \mathbf{PE}p$ is preserved as argued in the above evaluation on obligation specification. The fact that the definitions $\mathbf{PE}p \leftrightarrow \neg\mathbf{OB}\neg p$ and $\mathbf{PR}p \leftrightarrow \mathbf{OB}\neg p$ hold follows as a corollary to the six other implications shown to hold.

C5: Composition of deontic expressions. Criterion C5 states that sequence diagrams allow the composition of deontic expressions. The natural candidate for composing policy rules expressed in sequence diagrams is combined fragments such as par for parallel composition, seq for sequential composition and alt for behavioral alternatives. An example of sequential composition is given in Figure 6. As explained there, the sequential composition of ConfigureMa-

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

chinePermission, cf. Figure 4, and TestMachineObligation, cf. Figure 5, gives a set of traces where each trace is the concatenation of one trace from the first and one trace from the second. The set of concatenated traces then adheres to both of the separate policy rules, and further deontic constraints can be imposed on the composition as shown in Figure 6.

Composition of policy rules can also be done at the level of OCL expressions by combining constraints. Assume that personnel other than SW engineers, e.g. SW consultants, were allowed to configure the TC, cf. Figure 4. This can be specified by extending the OCL expression to actor.oclIsTypeOf(SW engineer) or actor.oclIsTypeOf(SW consultant).

Ideally we should be able to separately specify policy rules that can be composed into a policy without much overhead. However, if there are several policy rules that refer to the same behavior and lifelines, care must be taken. Consider again the permission to configure the TC given in Figure 4. If access for SW engineers and SW consultants were specified separately, the one characterizes ConfigureMachine as positive when the client is SW engineer and the other characterizes ConfigureMachine as positive when the client is SW consultant. The problem is that by using an OCL invariant in these cases, the first characterizes the second as negative and vice versa. In the general case, several constraints can hence not consistently be specified separately.

For the composition of a set of permissions where each permission is optional and have the same addressee, the STAIRS xalt operator can be utilized. STAIRS distinguishes between the two choice operators alt and xalt for potential and mandatory choice, respectively. In a composition using alt, the positive traces of each operand specifies legal behavior, but a system adhering to the specification needs to ensure only that at least one of the choices are available. Given a set of permissions, however, each of these must be enforced. The xalt operator captures this. Not only do the positive traces of each operand specify legal behavior; an implementation of the specification must ensure that all of the choices are explicitly available.

C6: Policy triggers. The sixth criterion states that sequence diagrams allow the specification of both event and state triggers. Our suggested solution in Section 7 and Section 8 was to refer to state triggers by means of OCL invariants or guards, and represent event triggers by receive events on lifelines.

In terms of the trace semantics for interactions formalized with STAIRS, a system state can be represented by the trace describing the complete system history of events leading up to that state. Our event trigger refers to the states the trace of which lead up to this event, while the state trigger refers to more general properties of traces. This interpretation corresponds directly to the interpretation of triggers over Kripke structures as provided in Section 2. Sequence diagrams and OCL hence do have the required expressivity in this respect.

C7: Policy specification in the UML spirit. Criterion seven states that policies can be expressed in the spirit of UML, meaning that the elements of a policy as we have defined them have their natural and intuitive correspondences within the sequence diagram notions and constructs. More precisely, a language customized for policy specification should be equipped with constructs that matches policy trigger, addressee, and deontic modality.

The notion of event trigger corresponds well with the UML notion of event. As shown in Figure 7, we suggest to represent event triggers as a receive event on the lifeline representing the addressee.

A UML state is a condition or situation of an object, and a set of states can be combined into a composite state. States and composite states relate closely to our notion of a state trigger, which is why we have chosen to represent them as such. Examples are provided throughout Section 7 above.

The notion of addressee refers to a set of actors in the system whose behavior is constrained by the policy. An addressee is in our specifications represented with a lifeline which is a role that can be instantiated by a set of objects. Although there is no direct correspondence to the notion of addressee in UML, the lifeline is a close match.

Finally, we have the deontic modalities. Sequence diagrams are not designated for capturing deontic constraints, so at this point there are no direct correspondences. Using the combined fragments assert and neg for obligations and prohibitions, respectively, is nevertheless quite intuitive.

C8: Utility for policy specification. The eighth criterion states that sequence diagrams allow the specification of policies in a manner suitable for engineers developing and maintaining systems that should adhere to the policies.

As compared to natural language, a policy specified in UML sequence diagrams may be more easily carried over to the implementation level, but the value of this must be balanced against potential difficulties in both specifying and understanding a policy given in sequence diagrams.

An engineer should be able to recognize the various elements of a policy rule. In some cases it is not obvious which type of deontic modality is represented, cf. Figure 4 and Figure 6. The former is supposed to capture a permission and the latter a prohibition, but as we have observed, both of them express a permission and a prohibition simultaneously. This is unfortunate not only because it is at the cost of understandability, but also because it reduces the flexibility.

As to the policy triggers, we concluded that sequence diagrams have the required expressivity. In our suggested policy rule specifications, the event triggers are quite easily recognized as they represent the first event of the diagrams. However, there is nothing in the event per se that distinguishes it as a policy trigger, which clearly is a disadvantage for a language that is supposed to capture policy rules as we

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

have defined them. State triggers as represented in OCL are even less recognizable. In the permissions specified in Figure 5, for example, we cannot immediately tell where the triggers are represented. Nor can we easily tell the triggers and the addressees apart.

The addressee is represented by a lifeline, but given a sequence diagram specifying a policy rule it is not always obvious which of the lifelines represents the addressee. In Figure 4, the addressee is represented by the lifeline of an anonymous actor, and the annotated OCL expression must be consulted in order to precisely understand to which set of actors the policy rule refers. This clearly makes both specification and human interpretation of difficult.

The continuous use of OCL for specifying state triggered policy rules in Section 7 represents a general disadvantage. They easily become quite complicated, cf. Figure 5, which make them tedious and hard to read. The specification of OCL expressions is moreover error-prone.

C9: Understandable to non-technicians. Criterion C9 states that sequence diagrams allow the specification of policies in a manner that is easy to understand for both decision makers and for the addressees of a policy.

As discussed wrt. criterion C8 above, the fact that the specifications of policy rules do not have explicit constructs corresponding to the various elements of a policy rule is an obvious weakness. At points where an engineer has difficulties in interpreting a policy rule, a non-technician is unlikely to comprehend the specification without much guidance. Specifically for OCL, which is purely non-graphical and tedious to read, some background in programming, first-order logic or set theory is required.

A general observation that summarizes much of the evaluation of this section is that the main obstacle to readability and understandability for non-technicians is the fact that a policy specification in sequence diagrams is not intuitively recognized as such.

10. Related Work

Live sequence charts (LSC) [4] relates closely to the STAIRS formalism, and could serve as an alternative for interpreting policy rules. The facility of precharts that specifies a behavior the conduct of which forces a following behavior to be conducted can be utilized for capturing policy triggers. LSC furthermore provides constructs for differing between behavior that must be conducted in all system runs and behavior that must be conducted in at least one run, which relates to the deontic modalities of obligations and permissions, respectively. The STAIRS formalism is, however, closer to the UML standard, and the assert and xalt operators can be further examined with respect to capturing obligations and permissions.

Most of the existing policy languages, see e.g. [17], are developed either for the purpose of specifying policies for being implemented on computerized systems or in order to do formal analysis of specifications, so requirements to human readability are not core issues.

Koch and Parisi-Presicce [8] evaluates the UML as a policy specification language where the requirements are readability, understandability and detection of potential policy conflicts. The first two requirements correspond closely to success criteria C8 and C9 in Section 5 of this paper. Their conjecture is that a visual notation supports comprehensibility, but a thorough examination and evaluation of this issue is left for future work. Their evaluation is moreover limited to RBAC although they refer to policies as a means to manage the general behavior of complex systems.

The reference model for open distributed processing [6] (RM-ODP) establishes concepts for the specification of distributed systems, including a deontic notion of policy. Efforts have been made for the purpose of analyzing and formalizing these concepts [1, 9, 10].

Aagedal and Milošević [1] suggest OCL pre-conditions for the capture of permissions and prohibitions. The pre-condition is represented by the reference to a Boolean attribute on an object and serves as a guard on the relevant behavior. A permission is specified with the pre-condition pre: Boolean-expression, so the behavior is allowed when the expression evaluates to true. A prohibition is specified with the pre-condition pre: not(Boolean-expression), and the behavior is hence not allowed when the expression evaluates to true. Principally there is then no difference between permissions and prohibitions; in both cases the specification allows a certain behavior when the guard evaluates to true and prohibits the behavior otherwise. This is a shortcoming corresponding to our state triggered permissions and prohibitions, but the issue is not discussed in [1]. As an example of an obligation, Aagedal and Milošević suggest an OCL invariant. This is of the form of a role binding policy similar to the one we expressed in Figure 5.

Linington et al. [10] relates the RM-ODP notion of policy to deontic logic. They address the issue that SDL express the static picture of a normative situation, and hence does not capture the dynamics of interacting actors. Based on [5], possible worlds are structured into a forward branching tree similar to our interpretation of traces over Kripke structures in Section 2. A deontic statement then partitions the tree into paths that do and paths that do not satisfy the statement. In [9], Linington discusses the specification of policies as defined in RM-ODP using the UML. He argues that most of the information held by a policy can be expressed within the UML, however that policy declaration is an aspect of the development process that go beyond the scope of UML.

Castejón and Bræk [2] discuss policy specification in relation to service oriented architecture. In their approach, services are specified using UML collaborations, and a pol-

Eighth IEEE International Workshop on
Policies for Distributed Systems and Networks (POLICY'07)
0-7695-2767-1/07 $20.00 © 2007 IEEE

icy framework is suggested for the governing of the execution of services. The policies are, however, informally depicted using notes, so the UML is not deployed for expressing policy rules.

11. Summary and future work

Sequence diagrams have excessive flexibility and expressivity with respect to characterizing interactions as positive or negative, and, as our evaluation in Section 9 showed, we were able to express close to all the various types and elements of policy rules. To what extent sequence diagrams are suitable for policy specification is hence not so much a question of expressivity. The problem is rather on pragmatical issues such as utility for designers and maintainers, readability and understandability to non-technicians. The reason for these pragmatical shortcomings lies very much in the fact that our policy rules do not conform with the spirit of UML; sequence diagrams are not designed for the purpose of capturing deontic constraints.

The success criteria given in Section 5 can be seen as requirements that should be satisfied by any language supposed to express policy rules as we have defined them. Our future objective is to develop a customized policy specification language that meets these requirements. We will take advantage of the expressivity of sequence diagrams with the STAIRS formalism as the starting point. By first establishing within STAIRS a precise understanding and representation of policies, we will, inspired by sequence diagrams, establish a more suitable notation.

The full report on which this paper is based provides a more thorough presentation of our evaluation [18].

Acknowledgment. The research on which this paper reports has partly been funded by the Research Council of Norway project ENFORCE (164382/V30) and partly by the European Commission through the S3MS (Contract no. 27004) project under the IST Sixth Framework Programme.

References

[1] J. Ø. Aagedal and Z. Milošević. ODP Enterprise Language: UML Perspective. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 60–71. IEEE CS Press, 1999.

[2] H. N. Castejón and R. Bræk. Dynamic Role Binding in a Service Oriented Architecture. In *The 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 2005)*. Springer, 2005.

[3] T. García. Baseline prototype infrastructure for the Aggregated Services scenario. TrustCoM Deliverable 11, 2005.

[4] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[5] J. F. Horty. Combining Agency with Obligation (Preliminary Version). In *Deontic Logic, Agency and Normative Systems, Proceedings of DEON'98, 3rd Int. Workshop on Deontic Logic in Computer Science*, pages 98–123. Springer, 1996.

[6] ISO/IEC. *Information Technology - Open Distributed Processing - Reference Model - Enterprise Viewpoint*, 2000.

[7] L. Kagal, T. Finin, and A. Joshi. A Policy Language for a Pervasive Computing Environment. In *4th International Workshop on Policies for Distributed Systems and Networks*. IEEE, 2003.

[8] M. Koch and F. Parisi-Presicce. Visual Specifications of Policies and their Verification. In *Proceedings of the 6th International Conference on Fundamental Approaches to Software Engineering (FASE 2003)*, volume 2621 of *LNCS*, pages 278–293. Springer, 2003.

[9] P. Linington. Options for Expressing ODP Enterprise Communities and Their Policies by Using UML. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 72–82. IEEE CS Press, 1999.

[10] P. Linington, Z. Milošević, and K. Raymond. Policies in Communities: Extending the ODP Enterprise Viewpoint. In *Proceedings of the 2nd International Conference on Enterprise Distributed Object Computing (EDOC'98)*, pages 11–22. IEEE CS Press, 1998.

[11] T. Mahler. Report on Legal Issues. TrustCoM Deliverable 15, 2005.

[12] P. McNamara. Deontic Logic. Stanford Encyclopedia of Philosophy, 2006. http://plato.stanford.edu/entries/logic-deontic/.

[13] Object Management Group. *Unified Modeling Language: Superstructure, Version 2.0*, 2005. www.omg.org.

[14] J. E. Y. Rossebø and R. Bræk. A Policy-driven Approach to Dynamic Composition of Authentication and Authorization Patterns and Services. *Journal of Computers*, 1(8):13–26, 2006.

[15] R. K. Runde, Ø. Haugen, and K. Stølen. Refining UML Interactions with Underspecification and Nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[16] M. Sloman. Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[17] M. Sloman and E. Lupu. Security and Management Policy Specification. *Network, IEEE*, 16(2):10–19, 2002.

[18] B. Solhaug, D. Elgesem, and K. Stølen. Specifying Policies Using UML Sequence Diagrams – An Evaluation Based on a Case Study. Technical Report A1230, SINTEF ICT, 2007.

[19] M. Steen and J. Derrick. Formalising ODP Enterprise Policies. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 84–93. IEEE CS Press, 1999.

[20] The TrustCoM Project. http://www.eu-trustcom.com/.

[21] G. H. von Wright. Deontic Logic. *Mind*, 60:1–15, 1951.

[22] F. Vraalsen and T. Mahler. Legal risk management for Virtual Organisations. TrustCoM Deliverable 17, 2006.

[23] R. Wies. Policy Definition and Classification: Aspects, Criteria, and Examples. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operation and Management*, 1994.