SINTEF

SINTEF ICT

Address: NO-7465 Trondheim, NORWAY
Location: Forskningsveien 1
Telephone: +47 22 06 73 00
Fax: +47 22 06 73 50

Enterprise No.: NO 948 007 029 MVA

SINTEF REPORT

TITLE

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

AUTHOR(S)

Bjørnar Solhaug, Dag Elgesem and Ketil Stølen

CLIENT(S)

REPORT NO.	CLASSIFICATION	CLIENTS REF.	
A1230	Open		
CLASS. THIS PAGE	ISBN	PROJECT NO.	NO. OF PAGES/APPENDICES
Open	978-82-14-04051-7	90B22000	34/0
ELECTRONIC FILE CODE		PROJECT MANAGER (NAME, SIGN.) Ketil Stølen	CHECKED BY (NAME, SIGN.) Iselin Engan
FILE CODE	DATE	APPROVED BY (NAME, POSITION, SIGN.) Bjørn Skjellaug	
	2007-05-31		

ABSTRACT

This report 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 we provide these with a trace based semantics interpreted over Kripke structures. This gives a semantics along the line of 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.

KEYWORDS	ENGLISH	NORWEGIAN
GROUP 1	Policy	Policy
GROUP 2	Modeling	Modellering
SELECTED BY AUTHOR	UML	UML
	Deontic logic	Deontisk logikk

Specifying Policies Using UML Interactions – 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 report 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 we provide these with a trace based semantics interpreted over Kripke structures. This gives a semantics along the line of 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 [17] is currently the de facto standard for the modeling and specification of information systems. Policy frameworks, see e.g. [21] for a survey, are increasingly being adopted as a means of managing such systems with respect to security, ICT services and networks, business processes, etc. An obvious issue to investigate is to what extent UML is suitable for the specification of policies for the management of these systems.

Sloman's [20] definition of a policy is much referred to and 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. Moreover, there may 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, particularly 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 computers whenever they leave their working station. However, humans obviously have the choice whether to comply with the rule or not. Incentive structures may be implemented for the purpose of encouraging

normative behavior, but these will not necessarily eliminate 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, on the other hand, usually do not have the option of disobeying policies. However, the normative aspect is still present in the sense that the potential behavior or the functionality of the actors go beyond 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" [21]. By separating the policy from the system, the behavior of the system can be governed by modifying the policies only, leaving the underlying implementation of the system unchanged [20].

Thus, the system itself with all its potential behavior provides one perspective, while the policy constraining the behavior of the system provides another. This report evaluates the suitability of deploying UML 2.0 sequence diagrams for policy specification.

The next section provides a classification of the type of policy rules we aim at formalizing in this report, as well as a trace based semantics for each type of rule. Policy rules are interpreted in terms of traces in compliance with the explanation of UML sequence diagrams in the standard [17]. We therefore provide a policy rule semantics along the line of sequence diagram semantics. Section 3 describes our target of evaluation, i.e. the parts of UML on which we will focus in the evaluation. In Section 4 we present our evaluation method. In 5 we give a set of success criteria describing the requirements that should be satisfied by the UML sequence diagrams as a policy specification language. In Section 6 we use class diagrams to specify an eLearning scenario for which a policy will be provided. Section 7 and Section 8 suggest how particular policy rules for the eLearning scenario can be specified using UML 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. In particular, the semantics of the specifications in Section 7 and 8 are compared to the semantics for the policy rules given in Section 2. Finally, in Section 10, we discuss some of the existing approaches to the specification of policies, specifically some of those using UML, before concluding in Section 11.

2 Policy Classification

The various existing approaches to classify policies may differ somewhat both in the definition of a policy and in the classification of types of policy rules. Differences usually depend on the purpose, scope, target, etc. of the

policy framework in question. A common feature, however, is that a policy in some sense constrains the potential behavior of a system; it is a set of rules that determine choices between alternative of behaviors.

A reasonable way of classifying policies is to operate with the tripartition of policy rules into *obligations*, *permissions* and *prohibitions*, based on deontic logic [24]. Several of the existing approaches to policy specification and classification have language constructs that correspond with these deontic types [1, 4, 11, 20, 22, 26]. This way of categorizing policy rules is furthermore proposed in standardization works by ISO/IEC and ITU-T on open distributed processing [10].

In addition to prescribing 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 we 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 [20].

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:

Obligation: An obligation states that whenever the policy trigger executes or applies, the addressee is required to conduct the behavior.

Permission: A permission states that whenever the policy trigger executes or applies, the addressee is allowed to conduct the behavior.

Prohibition: A prohibition states that whenever the policy trigger executes or applies, the addressee is forbidden from conducting the behavior.

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

2.1 Trace semantics for deontic modalities

We will in the following establish a trace based semantics for deontic modalities. The semantics will later be used as a means to deciding whether the UML sequence diagrams are sufficiently expressive to capture these modalities.

Standard deontic logic (SDL), see e.g. [16], 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,

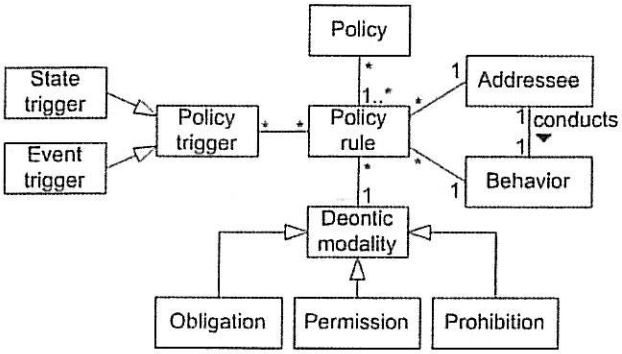

Figure 1: Policy classification

denoted $\models_w p$ and $\models_w \neg p$, 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 $\models_w \mathbf{OB}p$, then $\models_v p$ for all acceptable worlds $v \in A^w$. The semantics of the statement $\mathbf{PE}p$ denoting "it is permitted that p" is that if $\models_w \mathbf{PE}p$, then there is at least one acceptable world $v \in A^w$ such that $\models_v p$. The semantics of the statement $\mathbf{PR}p$ denoting "it is prohibited that p" is that if $\models_w \mathbf{PR}p$, then there is no acceptable world $v \in A^w$ such that $\models_v p$.

We shall not pay attention to the axioms and rules that define the syntax of SDL in this report, 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 states that for each possible world w, there is at least one acceptable world, i.e. $A^w \neq \emptyset$.

Notice that there is an alternative Kripke semantics for SDL in which the possible worlds are ordered by degree of ideality with respect to which the satisfiability of deontic expressions is defined. The semantics presented above is a special case of such a semantics, namely one in which there are only two degrees, acceptable and unacceptable. The latter variant is sufficient for the purposes of this report.

The various deontic modalities can in SDL be expressed in terms of each other as shown by the definitions $\mathbf{PE}p \leftrightarrow \neg\mathbf{OB}\neg p$ and $\mathbf{PR}p \leftrightarrow \mathbf{OB}\neg p$. Figure 2 illustrates the semantics. The squares depict the acceptable worlds and the propositional statements inside the squares describe what must be satisfied in these worlds for the deontic statements to be true.

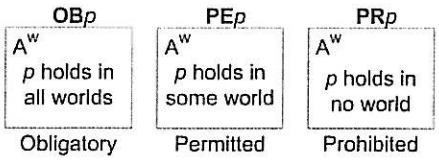

Figure 2: Truth conditions for deontic statements

We use the term *trace* to denote 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 of affairs. That is to say, each world $w \in W$ uniquely corresponds to a specific trace. The relation A between worlds in the semantics for SDL captures the set of acceptable worlds in a given world w. Since a policy specifies allowed and obligated 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 interactions 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 that we interpret 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 specified as a propositional logical expression, $W_T \subseteq W$ denotes the set of possible worlds for which the trigger holds.

By moving from one possible world w to another world $v \in A^w$, some sort of behavior has been conducted. Since the trace describing w is a prefix of the trace describing v, the behavior conducted from w to v is given by the trace for v with the prefix removed.

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 the evaluation of UML interactions.

A policy rule refers to a certain behavior. Generally, the specification is to a certain degree abstracted or underspecified, which means that the behavior can be conducted in various concrete ways. Each of the concrete specifications is described by a trace of events, so a given behavior corre-

sponds to a set of traces.

Now, assume 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 [17] sequence diagrams. Sequence diagrams specify interactions, which describe how messages are sent between entities for the purpose of performing a behavior. We evaluate to what extent they are suitable as a notation for the specification of policy rules. A central question is to what extent sequence diagrams are capable of capturing the various deontic constraints. OCL is in this 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, and some existing approaches have used OCL for this purpose, e.g. [1, 4, 18].

We use class diagrams to describe the static properties of the system for which policy rules are specified. They are discussed only in relation to interactions for policy rule specification, and are outside the focal point of the evaluation.

The UML 2.0 specification [17] provides an informal description of the trace semantics of interactions. A trace is a sequence of events occurrences ordered by time describing how entities interact for the purpose of conducting some system behavior. Interactions defines a set of allowed traces and a set of forbidden traces, expressing desired and undesired behavior, respectively. This semantics is formalized with STAIRS [8, 19] 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. Correspondingly, a negative trace means that the execution is invalid, illegal or undesirable. A trace is inconclusive if it is categorized as neither positive nor negative. This means 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.

In the following section we describe how the evaluation will be conducted, and we give a set of criteria that the relevant parts of UML should satisfy in order to be a suitable notation for policy specification.

4 Evaluation Method

Our evaluation of sequence diagrams as a language for policy specification is conducted over the following steps:

1. A carefully selected scenario is used as a case study for policy capturing
2. The success criteria are 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 [6]. This eLearning scenario is well documented and was deployed during the TrustCoM project [23] 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 report 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 the documentation of the eLearning scenario and the documentation of risk issues related to the scenario [15, 25][1].

5 Success Criteria

In Section 7 and Section 8 we try to use UML sequence diagrams to specify policies of relevance for the selected case. Subsequently this attempt is evaluated against the following nine success criteria.

C1: Permissions may be expressed by sequence diagrams.

C2: Obligations may be expressed by sequence diagrams.

C3: Prohibitions may be expressed by sequence diagrams.

[1]The mentioned deliverables [6, 15, 25] are available as downloads on the TrustCoM homepage [23].

The sequence diagram notation is not a policy specification language, and there are no explicit sequence diagram constructs available for specifying the deontic modalities of obligation, permission and prohibition. The testing of the extent to which sequence diagrams meet criteria C1 through C3 then amounts to examine to what extent there are language constructs available that allow specifications, possibly annotated with OCL expressions, the semantics of which corresponds to the semantics of deontic modalities as presented in Section 2.

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

Standard deontic logic [16] 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 by 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 both event and state triggers, as well as the combination of the two.

The policy triggers specifies the conditions under which the 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 may be expressed in the spirit of UML

By criterion C7 we mean that the various elements of a policy 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 within the sequence diagram notation.

C8: Sequence diagrams allow the specification of policies in a manner suitable for engineers developing and maintaining systems that should adhere to the policies.

This criterion is related to the pragmatics of the language, i.e. who are supposed to use the language and for which purposes. 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 the addressee is, 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.

As the previous criterion, this addresses the pragmatics of policy specifications. 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 a high 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.

The pragmatics is a crucial aspect, since if the language is hard to understand and does not facilitate policy specification it will be of low value even though the expressivity and the semantics are as desired.

6 The eLearning Scenario

In this section we use UML class diagrams for the specification of the structure of and the relationships between the entities involved in an eLearning scenario.

The case study scenario concerns a small to medium sized enterprise (SME) network specializing in providing tailored eLearning services to end-users [6, 15, 25]. 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 (tailored package of learning content), a virtual organization (VO) 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 tailored learning paths that meet the customer needs and requirements. 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. Each module consists of learning content in some form, e.g. pdf or video files.

Figure 3 shows the context of the analysis. The AS part describes the elements that are involved in the establishment of VOs for the purpose of providing aggregated services (AS) to the end-users.

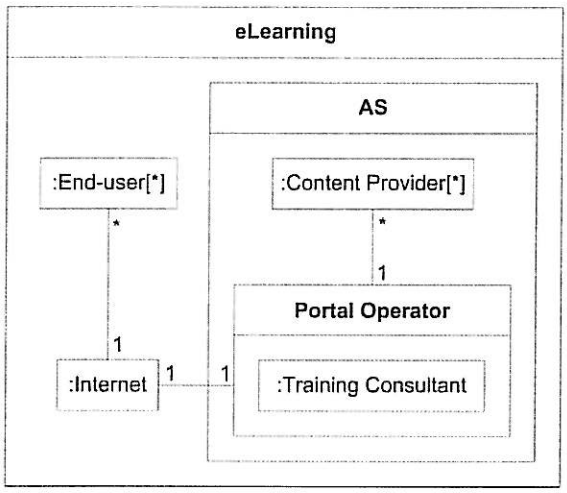

Figure 3: eLearning context

A given LCP must be accepted as a member of the SME eLearning network before it can provide its services by participating in the eLearning VOs. The PO is the party contracting with the LCPs and is as such responsible for and in control of which LCPs are to be allowed to enter the network. We will address the relations between the PO and one particular LCP, referred to as NoContent. NoContent is assumed to currently being a potential cooperation partner, and the objective is to specify a policy on the behalf of the PO regulating future interactions with NoContent.

Both the PO and NoContent are organizations that have a number of employees, are structured into departments and that have some machines, i.e. automated software or hardware actors, providing automated services. A general specification of an organization is given in the class diagram of Figure 4. Observe that an organization is an actor that has other actors, possibly sub-organizations, as members.

We will specify a policy held by the PO for the purpose of protecting its core assets. The asset 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

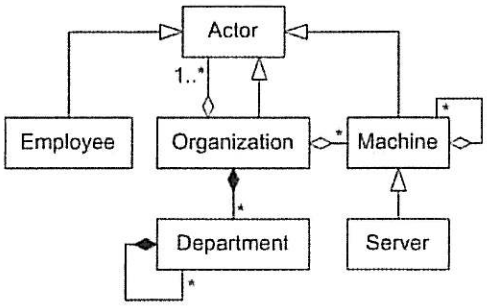

Figure 4: Organization

the ontology.

The TC and the ontology are crucial both for the establishment and the ongoing conduct of the eLearning services, and therefore they constitute highly sensitive information that is very valuable to the PO. Protective measures must be established and maintained to ensure the integrity and confidentiality of this information, but at the same time the collaborating LCPs must be provided access so that they can present their services and resources in a meaningful way. The LCPs must also ensure that the selection of LCPs in the tailoring of learning paths is executed in a fair manner. Figure 5 shows a class diagram capturing the elements of the PO relevant for the policies that will be specified in the sequel.

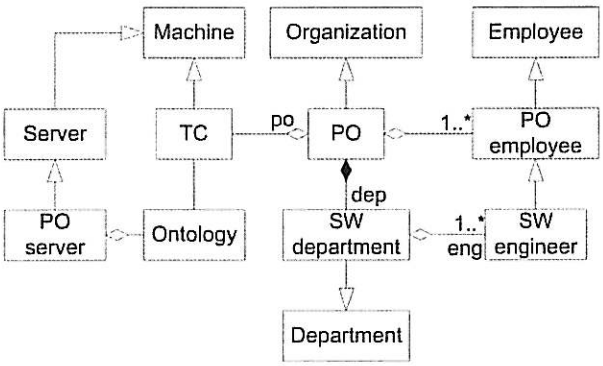

Figure 5: The portal operator

The class diagram of Figure 6 shows the relevant elements of NoContent. The Boolean attribute nda of NoContent employee indicates for each employee whether he or she has signed a non disclosure agreement with the PO concerning information on the TC and the ontology.

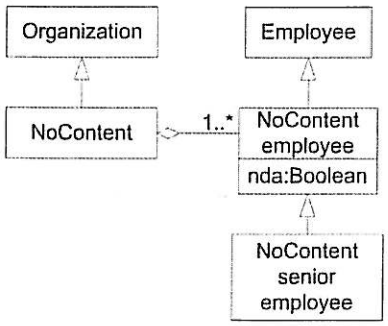

Figure 6: NoContent

7 Specifying State Triggered Policies

A state triggered policy rule applies in a given set of states, viz. the triggering states. In this section we suggest how sequence diagrams and OCL invariants can be utilized to express state triggered policies.

Figure 7 shows in a generic way our approach to specify state triggered policies. The interaction use refers to the behavior relevant to the policy rule, and the lifelines show the entities involved in the behavior. The addressee is always involved in the behavior in addition to any number of other entities, here represented by one lifeline for illustrative purposes. An OCL invariant placed in an attached note imposes constraints on the interaction in accordance with some policy rule.

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

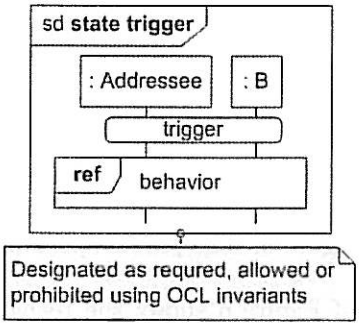

Figure 7: State triggered policy rule

7.1 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 in this subsection specify permissions that constrain the access to the TC and the ontology.

The first concern is the access to the learning content ontology. The following informal statement specifies to whom and under what condition the access is granted: *Access to read the ontology is granted to senior NoContent employees in a non-disclosure agreement with the PO and to PO employees.* The purpose of this rule is to prevent the ontology from being publicly known.

The ontology is stored on a server, see Figure 5, and the access to the ontology will be restricted by restricting the access to this server. Since there are two different addressees, one of which is granted a conditional access, we will split the statement into two as shown in Table 1.

Modality	Behavior	Trigger	Addressee
Permission	Read from PO server	State: All	PO employee
Permission	Read from PO server	State: Accessor in NDA with PO	NoContent senior employee

Table 1: Permissions

The categorization of the elements of the policy rules is done according to the policy classification in Section 2. In both these cases, the rules are state triggered. Since the PO employee access is granted in all states, the state trigger is in that case equivalent to true.

Figure 8 shows the UML specification. The sequence diagram specifies the ReadServer interaction which involves the PO server and some anonymous actor. The attached OCL invariant ensures that the actors are of the type that are permitted to read from the PO server.

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.

This sequence diagram characterizes the traces produced by the ReadServer interaction as positive whenever the client is of the required type. By placing the ReadServer within the combined fragment opt, we ensure that the execution of these traces are optional since the trace in which the ReadServer is skipped is also 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 optional.

In a policy specification, the prohibitions and permissions together characterize the allowed behavior. There are three approaches to capture the

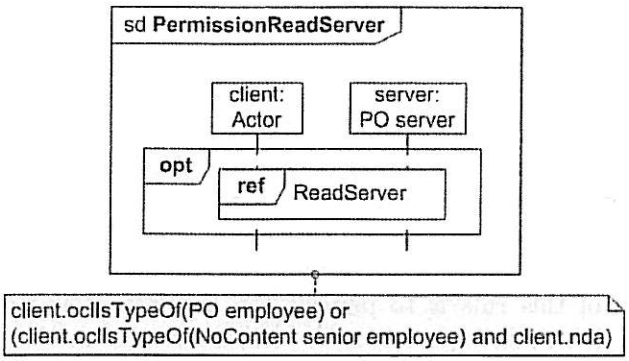

Figure 8: Access to read the ontology

allowed behavior. We need to consider all three separately, since the selection of approach 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 selection of which approach to adopt usually depends on the type of system in question. However, this is an issue we will not pursue in this report. The problem of detecting and resolving policy conflicts and inconsistencies that may arise by using the third approach is also outside the scope of this report.

What we need to address is that if the approach is not prohibition by default, the policy specification must ensure that *only* the actors explicitly allowed to access the PO server are granted access. In this case the constraint is given as an OCL invariant, i.e. a Boolean expression that must evaluate to true for the interaction to execute. The traces in which the client is not a PO employee or a NoContent senior employee in NDA are thus negative. This means that the permission specification we suggest in Figure 8 implicitly is also a prohibition.

Notice finally that class diagrams that specify the system to which a policy applies can be used to capture the inheritance of permissions, obligations and prohibitions: The specialization relation serves as an inheritance relation by the fact that all instances of a class inherits from all of its parent classes. This is closely related to role hierarchies that describe aggregation of access rights within role-based access control (RBAC) [5]. The permission we have specified in Figure 8 thus as well applies to SW engineer.

The next permission rule concerns the access to do configurations on the

TC software. The functionality of the TC is critical, both in terms of ensuring that the learning paths suggested to the end-user match the end-users needs and requirements, as well as to ensure that the selection of LCPs is executed fairly, i.e. that no LCP is discriminated.

Informally, the permission policy rule 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 made during office hours. The elements of the policy is given in Table 2.

Modality	Behavior	Trigger	Addressee
Permission	Configure TC	State: Time between 10 p.m. and 4 a.m.	PO employee that is SW engineer

Table 2: Permission

Figure 9 shows the sequence diagram in which this policy rule is expressed. The state trigger is expressed as a time constraint. Time constraints is a part of the notation for sequence diagrams, and an equivalent to this is not available within OCL. In this case, OCL is used only to capture the addressee of the policy rule.

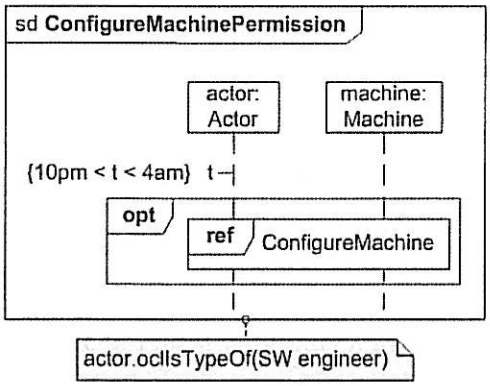

Figure 9: Access to configure the TC

By placing the ConfigureMachine interaction within an opt, the trace in which the actor skips this interaction is positive. The other positive traces are as intended, viz. those in which ConfigureMachine is executed when actor is of type SW engineer and the time is within the required interval.

Since the policy rule is expressed by invariants, all traces in which the invariants evaluate to false are negative. Thus, the permission specification implicitly is a prohibition as well. It is therefore irrelevant whether the

adopted approach is prohibition by default or not.

7.2 State Triggered Obligations

A state triggered obligation expresses that in a given set of states, the addressee is required to conduct the given behavior. This subsection suggests how a state triggered obligation can be specified using sequence diagrams and OCL.

Since the functionality of the TC is critical to the PO, a requirement on software testing is imposed. Informally, the statement is as follows: *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 5. Their responsibility is to assign the task to one or several of their software engineers. Table 3 shows the structure of this obligation rule.

Modality	Behavior	Trigger	Addressee
Obligation	Testing of TC	State: TC configuration finalized	SW department

Table 3: Obligation

Essentially, this obligation states that one particular behavior, viz. software configuration, requires another behavior to follow, viz. software testing, i.e. the obligation imposes a relationship between behaviors. We will specify the given obligation as an OCL invariant, and in order to do so, we assume the TC has a Boolean attribute *configured* that evaluates to true when the configuration is finalized.

Figure 10 suggests how this obligation can be specified. The OCL expression refers to navigations over associations in the class diagram of Figure 5. Navigations over associations result in collections like sets, bags and sequences, and a property of a collection is accessed by using an arrow '->' followed by the name of the property. In Figure 10, the exists operation is used. Given that 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.

The OCL invariant is defined according to what Castejón and Bræk [3] refer to as a role binding policy. It states the conditions under which there must be at least one software employee that is bound to the role of tester in the TestMachine interaction and therefore conducts the required behavior. Although it is not clear from the UML semantics, we assume that the instantiation of a role or a lifeline implies the participation in the interaction, i.e. it is not possible to be bound to a role in a sequence diagram without conducting the specified activities. The same assumption is made in [1]. By

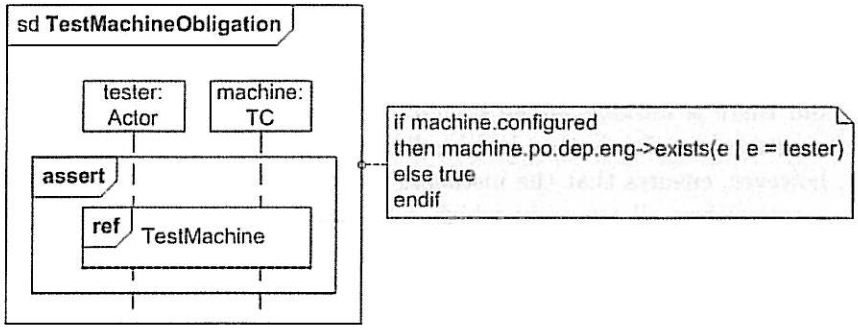

Figure 10: Obligation

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 traces, except those corresponding to TestMachine, as negative at this point of the execution.

Notice that the else clause is set to always evaluate to true. In this case this means that there are no constraints on the TestMachine interaction when configurations have not been conducted; the traces corresponding to TestMachine are positive when machine.configured evaluates to true.

7.3 State Triggered Prohibitions

A state triggered prohibition expresses that in a given set of states, it is forbidden for the addressee to conduct the given behavior. We will in this subsection address a separation of duty policy rule. Rules for separation of duty define mutually exclusive sets of permissions and can be defined as prohibitions.

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 who is different from the one that conducted the configuration.* The structure of this statement is shown in Table 4.

Modality	Behavior	Trigger	Addressee
Prohibition	Both configure and test TC	State: All	SW engineer

Table 4: Prohibition

The combination of the two behaviors is expressed in the sequence di-

agram of Figure 11. There are two lifelines representing tester and conf, where conf is the entity configurating the TC. Both roles are of type Actor, and there is nothing in the interaction specification that prevents one Actor instance to play both roles simultaneously. The attached OCL invariant, however, ensures that the instances must be different. The interaction hence categorizes all traces in which conf and tester are instanced by the same entity as negative.

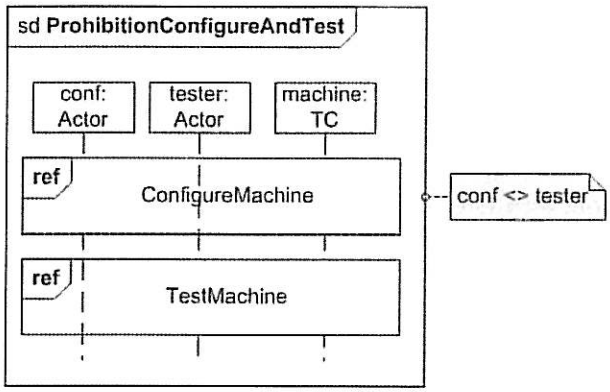

Figure 11: Prohibition

Implicitly, this diagram specifies a permission also, since the interactions are allowed to execute should the OCL expression evaluate to true. It is therefore 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 11 with references to ConfigureMachinePermission, cf. Figure 9, and TestMachineObligation, cf. Figure 10, respectively.

8 Specifying Event Triggered Policies

An event trigger can be represented as a receive event on the addressee lifeline. Figure 12 suggests a generic specification of event triggered policies. The challenge is to find suitable ways of relating the receive trigger event and the given behavior.

In addition to the addressee, there may be one or several other roles involved in the interaction. For illustrative purposes, we have in Figure 12 shown one additional lifeline, although there can be arbitrarily many.

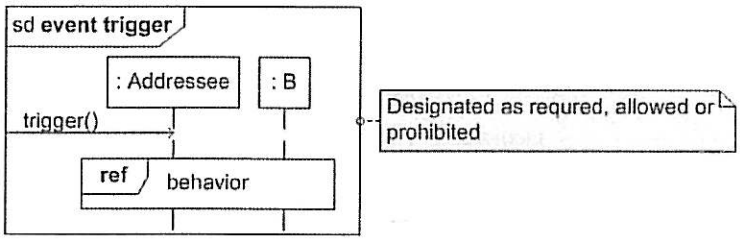

Figure 12: Event triggered policy

8.1 Event Triggered Permissions

Event triggered permissions can be expressed by placing the relevant behavior immediately after the triggering event as illustrated in Figure 13.

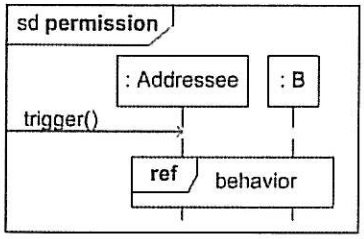

Figure 13: Event triggered permission

The interaction 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 is sufficient when the approach is prohibition by 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 triggering the permission. This can be done by simply using the neg operator with the behavior as the operand.

8.2 Event Triggered Obligations

Event triggered obligations can be expressed by placing the required behavior within an assert fragment. This will specifically designate all other behavior than the prescribed one as prohibited when the execution reaches the beginning of the assert construct. This means that the traces corresponding to the concatenation of the triggering event and the traces representing the execution of the behavior are positive, while all other traces are negative. Figure 14 suggests how an obligation can be specified using assert.

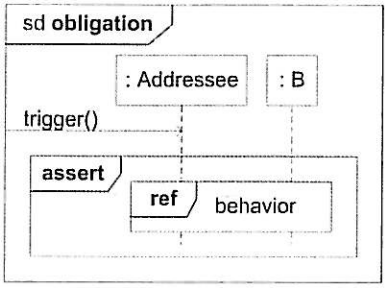

Figure 14: Event triggered obligation

As a concrete example of an event triggered obligation, we specify the obligation expressed in Table 3 as shown in Figure 15. In the section 7.2, this obligation was specified as state triggered and referred to a Boolean attribute of the TC. Here we rather assume that the SW department is alerted by the TC whenever a configuration has been conducted.

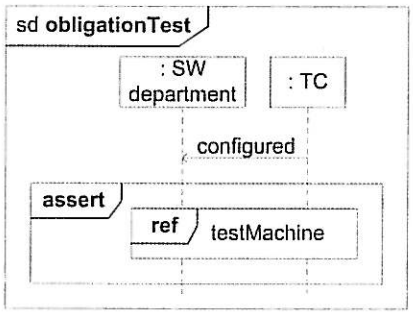

Figure 15: Event triggered obligation

8.3 Event Triggered Prohibitions

The neg operator is the obvious construct for expressing event triggered prohibitions, and shown in Figure 16. The semantics for the interaction is a set of negative traces corresponding to the trigger event followed by the traces produced by executing the behavior interaction. The only allowed trace in this case is one consisting of the reception of the trigger event only.

If behavior is permitted by default it is sufficient to specify prohibitions alone. If all behavior is to be explicitly categorized as one or the other, the behavior must be specified as permitted when the event triggering the prohibition does not occur.

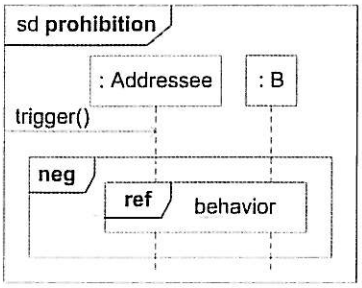

Figure 16: Event triggered prohibition

9 Discussion

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

9.1 Permissions

The first success criterion states that permissions may be expressed by sequence diagrams. Section 7.1 and Section 8.1 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, as exemplified in Figure 9 and Figure 13.

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 represening the behavior. A sequence diagram characterizing the set B as positive, as do our permission specifications suggested above, 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 of B. The latter, however, characterizes B as negative, which is inconsistent with the permission. Thus. the implication holds.

Observe, importantly, that for the state triggered permission we have suggested, the specifications are stronger than what is required. An OCL invariant on an interaction must evaluate to true for the interaction to execute. In cases in which it evaluates to false, the traces corresponding to the interaction are characterized as negative, which means that a state triggered permission implicitly also specifies a prohibition. In Figure 9, 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 $\mathbf{PR}p$ is triggered 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 independently of prohibitions and vice versa.

9.2 Obligations

Criterion C2 states that obligations may be expressed by sequence diagrams. Our suggestions to specify state and event triggered obligations are given in Section 7.2 and Section 8.2, respectively. As exemplified in Figure 10 and Figure 15, obligations are captured by applying the assert operator on the interaction use representing the relevant behavior. 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 obligatory in world w, then $\forall v \in A^w : \exists b \in B : v = w^\frown b$, where B is the set of traces representing the behavior. Since the assert operator ensures that the traces corresponding to the behavior are the only continuations allowed at that point in the execution, our obligation specifications match the desired semantics.

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

9.3 Prohibitions

The third success criterion states that prohibitions may be expressed by sequence diagrams. Figure 11 in Section 7.3 suggest how a state triggered prohibition can be specified by an OCL invariant on a sequence diagram where the latter expresses the behavior relevant for the prohibition. This specification characterizes the traces corresponding to the behavior as negative when the invariant evaluates to false. Figure 16 in Section 7.3 specifies an event triggered prohibition in which the traces corresponding to the behavior are 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 $\forall v \in A^w : \neg\exists b \in B : v = w^\frown b$, where B is the set of 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 of B. Thus the implication $\mathbf{PR}p \rightarrow \mathbf{OB}\neg p$ also holds.

A state triggered prohibition implies a permission in the same way that a state triggered permission implicitly specifies a prohibition, as observed in Section 9.1 above: If the OCL invariant evaluates to true, the traces corresponding to the behavior are characterized as positive. Interpreted over Kripke structures, this means that if a state triggered prohibition $\mathbf{PR}p$ holds in the set of worlds $W_T \subseteq W$, the implicit permission $\mathbf{PE}p$ is triggered in the complementary set of worlds $W \setminus W_T$.

9.4 SDL Axioms and definitions

Criterion C4 states that the formalization under criteria C1 through C3 respects the axioms and definitions of SDL.

In Section 2.1 we emphasized one axiom and two definitions. As argued in Section 9.2, the axiom schema $\mathbf{OB}p \rightarrow \mathbf{PE}p$ is preserved by our policy notation. The fact that the definitions $\mathbf{PE}p \leftrightarrow \neg\mathbf{OB}\neg p$ and $\mathbf{PR}p \leftrightarrow \mathbf{OB}\neg p$ are preserved follows as a corollary to the six other implications proven in Sections 9.1 through 9.3.

9.5 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 interaction alternatives. An example of sequential composition is given in Figure 11 in Section 7.3. As explained there, the sequential composition of ConfigureMachinePermission, cf. Figure 9, and TestMachineObligation, cf. Figure 10, gives a set of traces where each trace is the sequential composition of one trace from the first interaction and one trace from the second. The set of composed traces adheres to both of the separate policy rules, and further deontic constraints can be imposed on the composition as shown in Figure 11.

Composition of policy rules can also be done at the level of OCL expressions by combining to different constraints. Assume that personnel apart from SW engineers, e.g. SW consultants, were allowed to configure the TC, cf. Figure 9. This can be specified by extending the OCL expression to actor.oclIsTypeOf(SW engineer) or actor.oclIsTypeOf(SW consultant).

We have in our case study suggested two ways of composing policy rules. 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 separately refer to the same behavior and lifelines, care must be taken. Consider again the permission to configure the TC given in Figure 9. If SW engineer access and SW consultant access 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 deontic 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 implemented. The xalt operator capture 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 available.

9.6 Policy triggers

The sixth criterion states that sequence diagrams allow the specification of both event and state triggers, as well as the combination of the two. Our suggested solution in Section 7 and Section 8 was to refer to state triggers by means of OCL invariants and represent event triggers by a 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, for example that a given set of sub-traces are present in some specific order. 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.

9.7 Policy specification in the spirit of UML

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 which is an occurrence in time that can be represented by the sending or reception of a message on a lifeline. As shown in Figure 12, 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.

9.8 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, shown in Figure 1. In some cases it is not obvious which type of deontic modality is represented, cf. Figure 9 and Figure 11. 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 to specify permissions independent of prohibitions and vice versa.

As to the policy triggers, we concluded in Section 9.6 above 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 interactions. However, there is nothing in the event per se that distinguishes it as a policy trigger, which clearly is a disadvantage for a policy specification language that is supposed to capture policy rules as we have defined them. State triggers as represented in OCL are even less recog-

nizable. In the permissions specified in Figure 10, 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 always 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 9, 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 policies 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 10, which make them tedious and hard to read. The specification of OCL expressions is moreover error-prone.

9.9 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. Decision makers must give their approval for the policy, and to do that they obviously need to understand what is presented to them. The same is the case for human addressees that are supposed to comply with the policy.

In both cases, those who are presented the policy specification generally have no technical background. Nevertheless, they should be able to read and understand the specification with little or no guidance.

UML sequence diagrams have many facilities that can be utilized for the purpose of making easily understandable representations of information. As discussed in the previous subsection, however, the fact that the specifications of policy rules do not have explicit constructs that correspond to the various elements of a policy rule is an obvious weakness. At points where an engineer has difficulties in interpreting a policy rule, as discussed for criterion C8, 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) [7] 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.

The last decade has experienced increased focus on policy based management, and various languages has been proposed for the specification of policies. Requirements to a language vary depending on the pragmatics of the language, i.e. who are supposed to use the language and for what purposes. Most of the existing languages, see e.g. [21], are developed either for the purpose of specifying policies for being implemented on computerized systems or in order to do formal analysis of specifications, and requirements to readability for human are not core issues.

Koch and Parisi-Presicce [12] evaluate 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 4 of this report. 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 [5] although they refer to policies as a means to manage the general behavior of complex systems. The scope of their analysis is hence more narrow than our.

The reference model for open distributed processing [10] (RM-ODP) establishes concepts for the specification of distributed systems, including a notion of policy that operates with the tripartition of policy rules into permissions, obligations and prohibitions. Efforts have been made for the purpose of analyzing and formalizing these concepts [1, 2, 13, 14].

Blanc et al. [2] propose ways of using the UML to specify RM-ODP concepts, but the specification of policies is left for future work. They suggest OCL, but in their paper the policies are represented informally by notes in natural language.

Aagedal and Milošević [1] also propose representations of RM-ODP concepts in UML. With respect to policy specification, they suggest OCL preconditions 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 precondition pre: Boolen-expression, so the behavior is allowed when the ex-

pression 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 no difference between permissions and prohibitions by using pre-conditions like this; in both cases the specification allows a certain behavior when the guard evaluates to true and prohibits the behavior otherwise. This is a shortcoming to the specification that corresponds to the state triggered permissions and prohibitions we suggested in Section 7, 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 10 in this report.

Linington et al. [14] relate 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 [9], 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 [13], 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 discuss policy specification in relation to service oriented architecture. In their approach, services are specified using UML collaborations, and a policy 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

This report investigates the suitability of using sequence diagrams for policy specification. A policy framework is built up of normative statements, and SDL is a well established tool that has been used over decades for the purpose of representing and reasoning about normative statements. By basing our policy definition and semantics upon SDL and interpreting Kripke structures in terms of traces, we have provided a formal explanation of policy rules that allows a precise evaluation of policy specification in sequence diagrams. Since the case study was conducted as a thorough analysis of a realistic and well established scenario, we furthermore ensured that the evaluation addressed real world issues.

The trace semantics can be used for the interpretation of various UML diagrams, including state machines, interaction overview diagrams, sequence

diagrams and communication diagrams. Sequence diagrams generalize all the other diagrams in the sense that for any UML specification with a trace semantics, a sequence diagram with the same semantics can be specified. By this observation, the results of our evaluation with respect to expressivity are valid for all the mentioned UML diagrams.

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 the deontic modalities of our policy rules do not conform with the spirit of UML; sequence diagrams are not designed for the purpose of policy specification.

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.

Composition of policy rules is for feasibility reasons an important concern. We have addressed this issue here, but work remains with respect to identifying suitable and effective ways of specifying and composing policy rules, while avoiding pitfalls such as introducing inconsistencies that can be hard to detect.

The activity of capturing and specifying policies for an organization usually starts at a high organizational level where overall goals, ethical standards, legal constraints, etc. are considered. A policy specification language should allow policies to be expressed at this level of abstraction. Once the high level policy is captured, there is a need to carry this down in the organizational structure and make it more concrete for the various departments, employees and information systems. The notion of refinement refer to this activity. Refinement is conducted stepwise, where each step brings the specification to a more concrete level, closer to realizability and enforcement.

STAIRS formally defines refinement relations, and these should be considered in future work on establishing a notion of policy refinement. The STAIRS notions of refinement furthermore support compositional refinement, and is transitive and monotonic with respect to several composition operators. Such a refinement relation for policy specifications ensures on the one hand that a process of stepwise concretization of a policy ends up in a realization that refines the initial high level specification. On the other hand,

a policy can be specified in a modular fashion, reflecting logical or physical organizational structures such as various departments and platforms.

This report first and foremost provides an evaluation of sequence diagrams for policy specification. At the same time, however, we have raised several questions and put forward matters that should be addressed in the development of a customized policy specification language. Our findings here combined with the STAIRS formalism for interaction specification and its precise definitions of refinement and composition provides a well-founded basis for future endeavors.

Acknowledgments

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] X. Blanc, M. P. Geravis, and R. Le-Delliou. Using the UML Language to Express the ODP Enterprise Concepts. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 50–59. IEEE CS Press, 1999.

[3] 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.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proceedings of POLICY'01, the International Workshop on Policies for Distributed Systems*, volume 1995 of *LNCS*, pages 18–38. Springer, 2001.

[5] D. Ferraiolo and R. Kuhn. Role-based access control. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[6] T. García. Baseline prototype infrastructure for the Aggregated Services scenario. TrustCoM Deliverable 11, 2005. Contract No. 01945 of the Sixth Framework Programme of EU.

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

[8] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 4:355–367, 2005.

[9] 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.

[10] ISO/IEC. *ISO/IEC FCD 15414, Information Technology - Open Distributed Processing - Reference Model - Enterprise Viewpoint*, 2000.

[11] 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*, pages 63–74. IEEE Computer Society, 2003.

[12] 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.

[13] 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.

[14] 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.

[15] T. Mahler. Report on Legal Issues. TrustCoM Deliverable 15, 2005. Contract No. 01945 of the Sixth Framework Programme of EU.

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

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

[18] 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.

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

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

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

[22] 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.

[23] The TrustCoM Project. Contract No. 01945 of the Sixth Framework Programme of EU, http://www.eu-trustcom.com/.

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

[25] F. Vraalsen and T. Mahler. Legal risk management for Virtual Organisations. TrustCoM Deliverable 17, 2006. Contract No. 01945 of the Sixth Framework Programme of EU.

[26] 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.