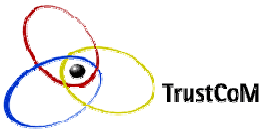



Deliverable

15

# D 15 Appendix E

## User-Level Language for the Analysis of Legal Risks

**WP9 Legal Issues**

A.J. Jones, KCL
M. S. Lund, SINTEF
X. Parent, KCL
B. Solhaug, SINTEF
K. Stølen, SINTEF

**TrustCoM**

*A trust and Contract Management framework enabling secure collaborative business processing in on-demand created, self-managed, scalable, and highly dynamic Virtual Organisations*

**SIXTH FRAMEWORK PROGRAMME**

**PRIORITY IST-2002-2.3.1.9**

*Prepared for the European Commission under FP6 Contract No. 01945 as a deliverable form*

## LEGAL NOTICE

The following organisations are members of the Trustcom Consortium:

Atos Origin,
Council of the Central Laboratory of the Research Councils,
BAE Systems,
British Telecommunications PLC,
Universitaet Stuttgart,
SAP AktienGesellschaft Systeme Anwendungen Produkte in der Datenverarbeitung,
Swedish Institute of Computer Scence AB,
Europaeisches Microsoft Innovations Center GMBH,
Eidgenoessische Technische Hoschschule Zuerich,
Imperial College of Science Technology and Medicine,
King's College London,
Universitetet I Oslo,
Stiftelsen for industriell og Teknisk Forskning ved Norges Tekniske Hoegskole,
Universita degli studi di Milano,
The University of Salford,
International Business Machines Belgium SA .

**Deliverable datasheet**

**Project acronym:** TrustCoM

**Project full title**: *A trust and Contract Management framework enabling secure collaborative business processing in on-demand created, self-managed, scalable, and highly dynamic Virtual Organisations*

---

| | |
|---|---|
| **Action Line:** | **6** |
| **Activity:** | **6.2** |
| **Work Package:** | **9** |
| **Task:** | |

---

| | |
|---|---|
| **Document title**: | **Appendix E – A User-level Language for the Analysis of Legal Risks** |
| **Version:** | |
| **Document reference:** | |
| **Official delivery date:** | **31 July 2005** |
| **Actual publication date:** | |
| **File name:** | |
| **Type of document:** | Report |
| **Nature:** | |

---

| | |
|---|---|
| **Authors:** | A. J.I. Jones[2], M. S. Lund[1], X. Parent[2], B. Solhaug[1], K. Stølen[1] |
| **Reviewers:** | CCLRC, Atos Origin |
| **Approved by:** | |

---

| Version | Date | Sections Affected |
|---|---|---|
| Final Version | 31/07/05 | All |

---

[1] SINTEF

[2] KCL

# Table of Content

# 1 INTRODUCTION

This fifth appendix to D15 is devoted to the user-level language we have developed for legal risks analysis.

We are able to use the CORAS language to document and analyse some aspects of legal risks, and the current language seems to be a good starting point for legal risk analysis. However, we wish to be able to model and analyse more legal aspects, for example whether the act of disclosing certain information is forbidden by contract, e.g., by a non-disclosure agreement. To enable this, we need facilities for:

- specifying ownership, which is highly relevant when determining e.g. the rights and obligations of an actor,
- specifying legal effects on different roles and activities, and
- correlating these effects with the relevant legal sources, e.g. which contract clause is the source of the legal effect in question.

We thus see the need to incorporate more information relevant to legal aspects into the graphical language. Furthermore, to facilitate the use of the graphical modelling language for documentation and communication of legal risk analysis results, the users of the language need a clear understanding of what the graphical models express. A graphical language for legal risk analysis should on the one hand be easily understandable for practitioners and on the other hand be sufficiently precise to allow in-depth analysis. We must be able to explain the meaning of the diagrams as well as how they can be combined and refined. Furthermore, to support automated analysis of the graphical models, tools must be able to extract and process relevant information. To enable this, the semantics of the graphical language needs to be defined, with particular emphasis on the notions of trust, security, privacy, data protection and intellectual property rights.

To meet the requirements stated above, we are extending the CORAS language with concepts and relationships relevant to legal analysis. A central conjecture is that modal logic, in particular deontic logic, may be an important source of inspiration with respect to the kind of language constructs required. These will enable us to specify e.g. which activities are permitted, obligatory and forbidden.

This appendix falls into four main sections: requirements, abstract syntax, concrete syntax and relations to the TrustCoM conceptual modelling:

- The requirements of the language are based on the experiences from the legal risk analyses of the TrustCoM scenarios.
- The abstract syntax defines the elements of the language. The conceptual model defined in Appendix D forms the basis for the abstract syntax of the language. Formalisation of this conceptual model in traditional first-order logic provides a tool for eliminating potential inconsistencies in the model,

and can help reduce the complexity of the model by showing that a given relation is in fact an implicit consequence of those already in the diagram.

- The concrete syntax defines the graphical appearance of the elements of the language (e.g., icons in the UML terminology). We here show how to extend the CORAS graphical language with the concepts relevant to legal analysis. A formalisation based on STAIRS[3] is provided in which use cases are interpreted in terms of sequence diagrams. We have started to test this formalisation by applying it to two TrustCoM scenarios.
- The work presented in this appendix aims to provide bridges between the conceptual modelling of AL1 in terms of UML diagrams and the legal issues of the TrustCoM project. We will show how the developments outlined above contribute to the project.

[3] See Ø. Haugen, K.E. Husa, R.K. Runde, K. Stølen, Why timed sequence diagrams require three-event semantics, Post-proc. of Dagstuhl seminar, Scenarios: Models, Algorithms and Tools, LNCS 3466, pages 1-25, Springer, 2005, and R.K. Runde, Ø. Haugen, K. Stølen, Refining UML interactions with explicit and implicit non-determinism, to appear in the Nordic Journal of Computing.

# 2 REQUIREMENTS TO THE LANGUAGE

Below is a list of requirements to the language. These are organised into three categories. Those of the first group are general. Those of the second group are methodological. Those of the third group deal with expressiveness.

## 2.1 General

*R1* *The language should support legal risk analysis*

This means that it should be complimentary to conventional legal analysis and conventional risk analysis in the sense that it adds value to the traditional methods of analysis. Furthermore, the language should reflect standard notation and terminology for risk analysis and legal analysis. This is related to what is called *domain appropriateness*;[4] the conceptual basis should be able to express all that is in the domain, and should not be able to express anything outside of the domain

*R2* It should be possible to model both input and output of legal risk analysis

Input includes description of the target of evaluation. Output includes description of identified risks and proposals for treatments.

Risk analyses may be carried out at any level of abstraction, from high-level assessments at enterprise level to low level assessments of technical implementations. For an assessment to be efficient it is important that the participants are able to stay focused on the desired abstraction level, and models describing the target of evaluation should be guiding the participants in this respect.

To support documentation, the language must be able to express the output of risk analyses. Moreover, risk analysis is a costly and time consuming process and cannot be carried out from scratch each time a system is updated or modified, so support for maintaining the analysis documentation and keeping it consistent with the target of evaluation as it changes over time is an important requirement. Risk analysis documentation has strong internal dependencies that must be accounted for when doing maintenance,[5] and these dependencies must be reflected in the language.

[4] Krogstie, J.: Evaluating UML using a generic quality framework. In Favre, L., ed.: UML and the Unified Process. IRM Press (2003) 1-22

[5] Lund, M.S., den Braber, F., Stølen, K.: Maintaining results from security assessments. In: Proc. 7th European Conference on Software Maintenance and Reengineering (CSMR'03), IEEE Computer Society (2003) 341-350

*R3* *The language should be easy to understand and use by all participants in legal risk analysis*

Standard risk analysis techniques, like for example Hazard and Operability analysis (HazOp)[6], are based on structured brainstorming, and depend on the participants being able to communicate. For a modelling language to be useful in such settings, it needs to support (and certainly not obscure) the communication between participants. This is related to *comprehensibility appropriateness*;[7] symbols representing different concepts should be distinguishable, a symbol should represent the same concept in all contexts, etc.

*R4* *The language should be based upon the conceptual model for legal risk analysis (see Section 3 below)*

Indeed, we need to be clear about the main concepts that will be used, and about their relationships.

## 2.2 Methodology

*R5* *The language should be graphical*

One of the principal advantages of graphical language is that it can be understood by non-technical people involved in the legal risk analysis (for instance, by the stakeholder); a main benefit of graphical languages is that most people seem to find them easy to comprehend, even when they have formally defined syntax and semantics.

*R6* *The language should be defined as a UML 2.0 profile*

The UML has become the de facto industry standard for modelling objects and components and is a widely used specification language in the software industry today. It is therefore a natural choice of basis for a language targeting risk analysis of IT systems; the use of UML builds a bridge between risk analysis and system development. The well-definedness of UML is also of great value, since it supports structured and uniform documentation, as well as support for automated consistency

[6] Redmill, F., Chudleigh, M., Catmur, J.: Hazop and software Hazop. Wiley (1999)

[7] Krogstie, J.: Evaluating UML using a generic quality framework. In Favre, L., ed.: UML and the Unified Process. IRM Press (2003) 1-22

checking and analysis. With UML 2.0 as the new standard, the older version of UML (versions 1.3, 1.4 and 1.5) are outdated, and not suited as a basis for the language. [8]

*R7* *The language should be compatible with the UML profile for security assessment (the "CORAS language")*

The CORAS framework meets the two previous requirements, and has proved useful and provided insight within the security domain. Therefore, this framework provides a good starting point. When using the CORAS UML profile, the threads identified during the legal risk analysis are modelled in so-called threat and unwanted incident diagrams, and the safeguards that can be used to counteract these risks are modelled in so-called treatment diagrams.

*R8* *The graphical language should have a formal semantics*

In its present form, the UML (upon which the CORAS language is based) lacks precisely defined semantics. In general, the semantics of both UML and the CORAS language is described in English. This means that it is difficult to determine whether a design is consistent, and whether a program correctly implements a design. Given the intended role of UML as a modelling notation standard, a well-defined semantics is imperative. One obvious solution is to try to map the UML graphics to some existing logical language, which would provide an unambiguous reference point. Indeed, formal methods provide notations for writing precise and rigorous specifications. Such methods also provide mechanisms for checking that the final system corresponds accurately to what was intended in the initial requirements.

However, the UML is rich. This is in contrast with existing logical languages, which in general employ a small number of building blocks. One issue is whether the graphical language should be mapped over one existing logical language or a combination of them. Alternatively, one might also decide to put aside some graphical elements and only provide formal semantics for a subset of the language.

*R9* *The semantics should provide a method for deciding, given an arbitrary diagram, what its meaning is*

This requirement is related to R8. It will be fulfilled if the language complies with another minimal requirement, viz. that the mapping rules should be defined by induction on the complexity of the diagram. By applying (from outside to inside) these rules to an arbitrarily chosen UML diagram, we can mechanically generate its counterpart within the logical framework. Of course, some further requirements might be introduced regarding the computability and complexity of the proposed translation

[8] UML Superstructure 2.0 Draft Adopted Specification, OMG document: ptc/04-10-02

method. A natural next step would be to investigate if the language meets such requirements. Given a UML diagram and a logical formula, we would like to know if there is a computational procedure allowing us to decide whether the formula is a correct translation of the UML diagram. If the answer is “yes”, we would like to know how complex the problem is – in particular, what resources of time are needed to carry out the required translation? These issues will not be addressed in this preliminary study, but are worth mentioning.

## 2.3 Expressiveness

*R10* *The language should be able to express legal aspects at a level of abstraction suitable for legal risk analysis*

The CORAS framework has proved insightful in the security area, but there are grounds to believe the language is not sufficiently expressive to deal with legal risks. Here are a number of notions that are expressed at neither the input level nor the output level: normative modalities (e.g., obligation and permission); ownership of confidential information; notion of “default reasoning” (norms allow for exceptions); distinction between the logical operators “and” and “or” (when combining norms), and so forth.

# 3 ABSTRACT SYNTAX

By abstract syntax, we mean a model of the elements in the language and the relations between these elements. It is essentially the conceptual model for legal risk analysis presented in Appendix D. This model takes the form of a UML class diagram, and comprises two main parts. The first one centres around the notion of legal norms, and the second one incorporates risk analysis and trust.

In this section, we present our results on an approach to formally define the conceptual model using traditional first-order logic (FOL). This formalisation is directly inspired by the attempt made by Berardi et al.[9] to formalise UML class diagrams. As mentioned, the UML has been widely accepted as the standard object-oriented development methodology in software industry. However, many graphical notations in UML only have informal English definitions and are thus error-prone.

This section falls into three parts. In subsection 3.1, we show how to define the main concepts related to class diagrams using first-order logic. These include the notions of class, association, generalisation and association class. We also illustrate the proposed definitions by applying them to some parts of the conceptual model as presented in Appendix D. In subsection 3.2, we address the problem of consistency checking for UML class diagrams. In subsection 3.3, we show how to analyse some more complex constructs, such as multiplicity, attribute and inheritance.

We stress that UML class diagrams can also be used to describe the target of evaluation.[10] There is, thus, another part of legal risk analysis into which the proposed formalisation may help reveal insights – the context identification. Needless to say, the developments presented below aim ultimately to contribute to the overall TrustCoM project, by providing a bridge (there will be others) between the conceptual modelling done within WP1 and the legal issues part of the project. Indeed, the WP1 partners make an extensive use of UML class diagrams as well.

## 3.1 Basic Constructs

### 3.1.1 Class

A class in an UML class diagram denotes a set of objects with common features. A class is graphically rendered as a rectangle. Formally, a class *C* corresponds to a FOL unary predicate *C* ranging over the individuals of the universe of discourse. Individuals are usually denoted by lower-case letters. It is convenient to divide the

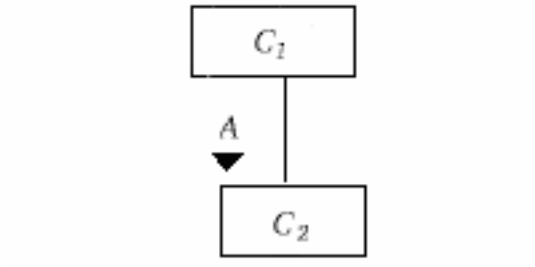
[9] D. Berardi, A. Cal, D. Calvanese and G. De Giacomo, Reasoning on UML class diagrams, Technical Report 11-03, 2003.

[10] See D15 Appendix A.

lower-case letters into two groups: *a*, *b*, *c*,… near the beginning of the alphabet, and *x*, *y*, *z*,… near the end of the alphabet. Letters of the first group are known as *individual constants* or simply *constants*. We think of them as denoting specific instances of the class under discussion (such-and-such actor/person, such-and-such legal norm, and so-forth). Letters of the second group are known as *individual variables* or simply *variables*. For example, *x* is a variable. We think of *x* as denoting not a specific individual but rather an arbitrary or unspecified individual.

### 3.1.2 Association

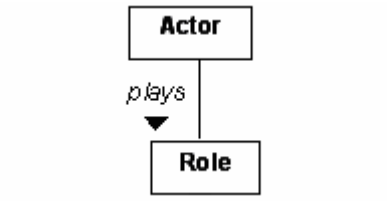
A binary association *A* between two classes $C_1$ and $C_2$ is graphically rendered as in the following figure.




The FOL translation proposed by the above authors is:

$$(1)\ \forall x, y : A(x, y) \rightarrow (C_1(x) \wedge C_2(y))$$

Let us briefly explain the notation. We write, e.g., $C_1(x)$ to denote `*x* is $C_1$' where *x* can be replaced by any object or individual. As just said, the letter *x* is a variable which serves as a place marker, and so is y. The symbol $\forall$ is called the universal quantifier. The quantified variables $\forall x, y$ are read `for every *x* and *y*'. In the conditional proposition (1), the conjunct at the right-hand side of $\rightarrow$ is said to be a necessary condition for the expression at the left-hand side. Therefore, the above proposition says that, for all *x* and *y* in the universe of discourse, a necessary condition for *x* to bear *A* to *y* is that *x* is of type $C_1$ and *y* of type $C_2$. Below is a self-explanatory illustration taken from the conceptual model. In this diagram, the association "plays" connects the classes "Actor" and "Role".




FOL translation:

$$\forall x, y : plays(x, y) \rightarrow (Actor(x) \wedge Role(y))$$

For sake of clarity, we disregard multiplicities at this point. The FOL treatment of this notion is given in subsection 3.3.

The conceptual model contains 17 other association relationships. Their logical rendering follows the same pattern. One of these association relationships involves

four classes. Therefore, equation (1), which applies to binary associations only, needs to be generalised somehow. In the UML, an association *A* among three or more classes $C_1,\ldots, C_n$ is denoted as a large diamond with paths from the diamond to each participating classes. The FOL translation is:

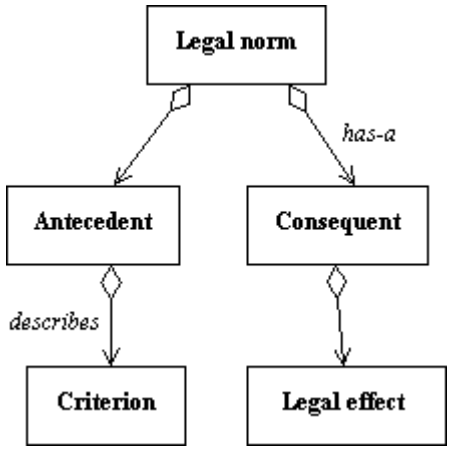
$$(2)\ \forall x_1,\ldots,x_n : A(x_1,\ldots,x_n) \rightarrow (C_1(x_1) \wedge \ldots \wedge C_n(x_n))$$

A particular kind of binary associations is aggregation. An aggregation is a binary relation between the instances of two classes, denoting a part-whole relationship. It is graphically represented using an open diamond shape at the association end of the class that contains the parts. In the study cited above, it is suggested representing aggregation as a binary predicate $G$ for which the following FOL assertion holds:

$$(3)\ \forall x, y : G(x, y) \rightarrow (C_1(x) \wedge C_2(y))$$

where the convention used is that the first argument of $G$ is the containing class. At first sight, nothing in the above formula indicates that an object of the whole (e.g., a car) has objects of the part (e.g., wheels). It might be argued that $G$ is just a placeholder for a binary predicate expressed in words, e.g. `has-a'. So the intended interpretation becomes clear as soon as the appropriate substitution has been done.

The conceptual model involves the following 4 aggregation relationships:




We, then, have

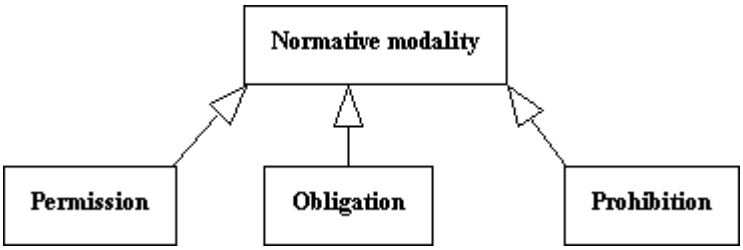
$$\forall x, y : has - a(x, y) \rightarrow (Legalnorm(x) \wedge Consequent(y))$$
$$\forall x, y : describes(x, y) \rightarrow (Antecedent(x) \wedge Criterion(y))$$

and so-on.

Berardi et al. assume (without any question) that aggregation and association can be formalised in the same way. A task for future work is to determine whether such an assumption is justified.

### 3.1.3 Generalisation

In UML one can use a generalisation between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that in general do not hold for the parent class. Graphically, generalisation is rendered as a directed line with a large open arrowhead, pointing to the parent. Of course, several generalisations can be grouped together to form a class hierarchy. The conceptual model for legal risk analysis contains 3 such class hierarchies. The simplest one deals with the normative modalities:




A class hierarchy as in the above figure is formally captured by means of the FOL assertion:

$$\forall x : C_i(x) \rightarrow C(x) \text{ for all i=1,...,n}$$

where $C$ is the parent-class and $C_1,\ldots, C_n$ are the child-classes (located at the same level in the hierarchy).

When needed, disjointness among $C_1,\ldots, C_n$ can be enforced by means of the FOL assertion:

$$\forall x : C_i(x) \rightarrow \neg C_j(x) \text{ for all i} \neq \text{j}$$

We do not force this assumption. For instance, we want to be able to say that if an action is obligatory, it is also permitted (but not the other way around).

When needed, a covering constraint expressing that each instance of C is an instance of at least one of $C_1,\ldots,C_n$ can also be enforced by means of the FOL assertion:

$$\forall x : C(x) \rightarrow \bigvee_{i=1}^{n} C_i(x)$$

where the expression at the right hand side of $\rightarrow$ is a shorthand for the (inclusive) disjunction of the propositions $C_1(x),\ldots, C_n(x)$.

We do not force this assumption either. For instance, we want to be able to talk about empowerment as opposed to permission even though this modality has not been explicitly introduced in the conceptual model.

Sometimes, in UML class diagrams, it is assumed that all classes not in the same hierarchy are a priori disjoint. Here we do not force this assumption; instead we allow

some classes to have common instances. For instance, we allow the class `information' and `permission' to have common instances, as we need to talk and reason about such norms as the permission to get access to information.

### 3.1.4 Association Class

A fundamental building block of the conceptual model is the notion of association class. It is usually defined as a modelling element that has both association and class properties. Graphically, an association class *A* is rendered as a class symbol attached by a dashed line to an association as shown below.

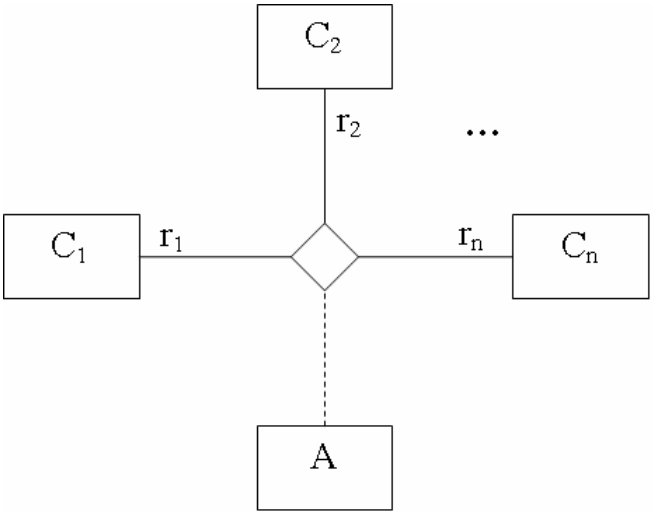
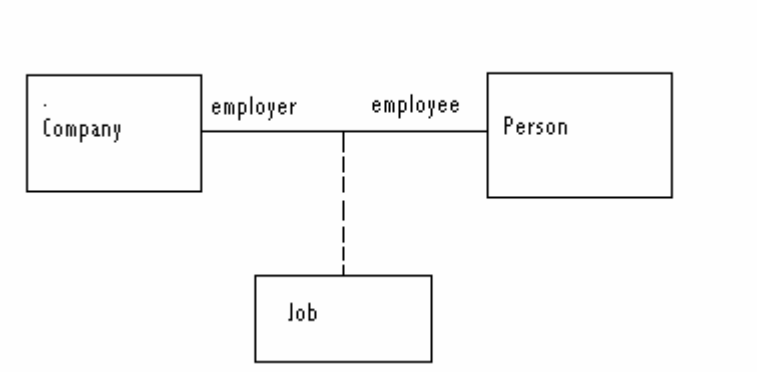



Figure 1 Association class

Note the presence of the labels $r_1,\ldots,r_n$. These denote so-called role names. A concrete illustration is given below. In this example, there exists an association class "Job" between the classes "Person" and "Company". Each instance of the first class plays the role "Employee", and each instance of the second class plays the role "Employer".



In first-order logic, an association between *n* classes $C_1,\ldots,C_n$ that has a related association class will be represented by a unary predicate *A* and *n* binary predicates $r_1,\ldots,r_n$ - one for each role name. These are not explicitly used in the conceptual model presented in Appendix D. According to Berardi et al., the concept of role name is central to the formalisation of the notion of association class. On this account, the interplay between the unary predicate *A* and the *n* binary predicates $r_1,\ldots,r_n$ is governed by the following specific laws:

$$(4)\ \forall x, y : (r_i(x, y) \wedge A(x)) \rightarrow C_i(y) \text{ for all i=1,...,n}$$

$$(5)\ \forall x : (A(x) \rightarrow \exists y.r_i(x, y)) \text{ for all i=1,...,n}$$

$$(6)\ \forall x, y, y' : A(x) \wedge r_i(x, y) \wedge r_i(x, y') \rightarrow y = y'$$

$$(7)\ \forall y_1,\ldots, y_n, x, x' : A(x) \wedge A(x') \wedge \bigwedge_{i=1}^{n}(r_i(x, y_i) \wedge r_i(x', y_i)) \rightarrow x = x'$$

Formula (4) types the association. Formula (5) and (6) say that there is exactly one element playing role $r_i$ for each component of *A.* Formula (7) says that two instances of *A* cannot represent the same tuple.

## 3.2 Consistency Checking

The construction of a class diagram is not constrained by the UML standards. Therefore, there is no guarantee that the graphic does not contain an inconsistency. We can avoid this pitfall by using formal methods.

In order to check consistency, proceed as follows: Let $A_1,\ldots,A_n$ be the set of FOL assertions corresponding to the entire diagram (e.g., the conceptual model for legal risk analysis as described in Appendix D). Intuitively, saying that the latter diagram is consistent amounts to saying that its classes can be populated without violating any of the requirements imposed by the diagram. Formally, this can be captured by means of the following definition, where $\phi$ denotes the conjunction of $A_1,\ldots,A_n$:

> A UML class diagram is said consistent if and only if $\phi$ is satisfiable (in the sense that there is a model for $\phi$).

Such a consistency check can realistically be carried out. In fact, we can settle the matter by determining whether or not $\neg\phi$ is valid. Should $\neg\phi$ be valid, then $\phi$ is not satisfiable, and the diagram is inconsistent. If $\neg\phi$ is not valid, then $\phi$ is satisfiable, and the diagram is consistent. The Beth-Smullyan method of semantic tableaux[11] provides an easy algorithm for deciding whether any FOL formula is valid – it, thus,

[11] See, e.g., Smullyan, First-Order Logic, Dover Pub., New York, 1968.

suffices to apply it to $\neg\phi$. Carrying out in detail the verification will be time-consuming. Nevertheless, if the conceptual model can be shown to be free of inconsistency, our effort will be well repaid.

It is worth mentioning that the formalisation not only provides tool support for consistency checking, but also can help reduce the complexity of the diagram by detecting possible redundancies For instance, determining equivalences of two classes allows for their merging. Likewise, if a given relation can be shown to be a logical consequence of other relationships, then the first one can be dropped, since it is implicit.

## 3.3 More Complex Constructs

Section 3.1 reports a first attempt to give a semantic foundation to the abstract syntax by mapping the conceptual model for legal risk analysis into traditional first-order logic (FOL). For ease of presentation, we only dealt with the most basic constructs. In this section, we give the FOL treatment of some more complex constructs, which were put to one side. These include:

- The multiplicity construct
- The attribute construct
- The inheritance mechanism

### 3.3.1 Multiplicity

For sake of clarity, we will consider the case of a binary association only. Suppose we have the following:

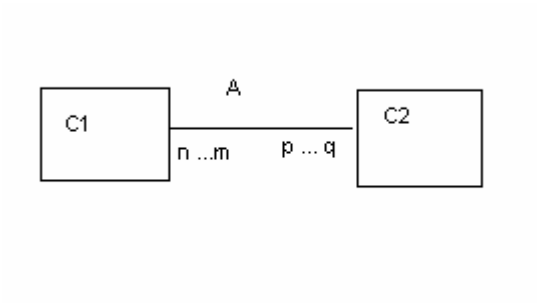



Figure 2 Multiplicity

As explained in section 3.1 the FOL translation of the solid line between the two rectangles is

$$\forall x, y : A(x, y) \rightarrow (C_1(x) \wedge C_2(y))$$

The multiplicity *p*…*q* on the binary association specifies that, for each object at the opposite (i.e., for each instance of the class $C_1$), there are at least p objects and at most q objects at the near end (i.e., of the class $C_2$). *n*…*m* has an analogous meaning for the class $C_2$.

The counterpart of multiplicity within FOL is rather complicated, for which reason we will introduce it in stages.

We here concentrate on the multiplicity *p*…*q* (the FOL treatment of *n*…*m* follows the same pattern). Let $\exists^{\leq q}\, y.A(x, y)$ be an abbreviation of the FOL formula expressing the proposition that, for a given *x*, there are at most *q* objects *y* such that $A(x, y)$. (Notice that *x* in the following occasionally will appear as a free variable.) It is natural to define $\exists^{\leq q}\, y.A(x, y)$ as follows:[12]

$$\begin{aligned} &\exists^{\leq q}\, y.A(x, y) =_{def} \\ &\forall y_1, \ldots, y_q, y_{q+1} : (A(x, y_1) \wedge \ldots \wedge A(x, y_{q+1})) \rightarrow \\ &((y_1 = y_2) \vee \ldots \vee (y_1 = y_q) \vee (y_1 = y_{q+1}) \vee \\ &(y_2 = y_3) \vee \ldots \vee (y_2 = y_q) \vee (y_2 = y_{q+1}) \vee \ldots \\ &\vee (y_q = y_{q+1})) \end{aligned} \tag{8}$$

Intuitively, the formula states the following: If one might find q+1 entities *y* that bear A to x, then at least two of them are identical. This ensures us that no more than q entities y are related to x by A.

Now, let $\exists^{\geq p}\, y.A(x, y)$ be an abbreviation of the FOL formula expressing the proposition that there are at least *p* objects *y* such that $A(x, y)$. It is tempting to put:

$$\begin{aligned} &\exists^{\geq p}\, y.A(x, y) =_{def} \\ &\exists y_1, \ldots, y_p : A(x, y_1) \wedge \ldots \wedge A(x, y_p) \wedge \\ &(y_1 \neq y_2) \wedge \ldots \wedge (y_1 \neq y_p) \wedge \\ &(y_2 \neq y_3) \wedge \ldots \wedge (y_2 \neq y_p) \wedge \ldots \\ &\wedge (y_{p-1} \neq y_p) \end{aligned} \tag{9}$$

This formula states that there are p or more objects *y* that bear *A* to *x*, and that none are identical with any of the others. Note that, in the particular case where p=1, then

[12] Berardi et al. do not say what this FOL formula is like. The definition proposed here generalises one given by L.T.F. Gamut, Logic, *Language and Meaning, Vol. 1, Introduction to Logic,* The University of Chicago Press, Chicago and London, 1991. (These authors consider the case of a unary predicate *A*.)

$\exists^{\geq p}$ collapses to the familiar existential quantifier $\exists$ (as in the example below). The FOL formula expressing the proposition that there is one or more *y* such that $A(x, y)$ is simply $\exists\, y.A(x, y)$. Of course, one might also express the proposition that there is zero or more *y* such that $A(x, y)$. In this case, the UML uses the icon $*$ alone, rather than the string *p…q*. The FOL translation is straightforward.

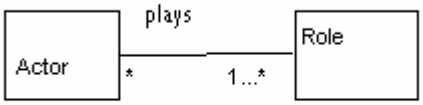
We are now in a position to translate the multiplicity constraint put on the binary association relationship *A* introduced at the beginning of this section. The proposition that each instance of the class $C_1$ can (so to say) participate at least *p* times and at most *q* times to relation *A*, can be enforced by means of the following two FOL sentences:

$$\forall x : C_1(x) \rightarrow \exists^{\geq p}\, y.A(x, y)$$

$$\forall x : C_1(x) \rightarrow \exists^{\leq q}\, y.A(x, y)$$

We here consider the case of $C_1$, but the multiplicity associated with $C_2$ will be expressed in much the same way.

We end this section with an example, taken from the conceptual model (see Appendix D). Consider the following:




In subsection 3.1.2, this binary association was rendered as

$$\forall x, y : plays(x, y) \rightarrow (actor(x) \wedge role(y))$$

Intuitively, the UML icon 1…* means that an actor can have one or more roles. This can be rendered as

$$\forall x : actor(x) \rightarrow \exists y.plays(x, y)$$

Intuitively, the UML icon * means that a role can be played by zero or more actors. This can be rendered as

$$\forall y : role(y) \rightarrow (\neg\exists\, x.plays(x, y) \vee \exists\, x.plays(x, y))$$

Notice that the latter formula is equivalent to the Boolean constant *true*. This is as it should be since the statement that a role is played by no one or someone clearly is a tautology.

### 3.3.2 Attribute

As already mentioned, in UML a class is in fact divided into three parts. The first part contains the *name* of the class, which has to be unique in the whole diagram. The second part contains the *attributes* of the class, with an associated type. The third part contains the *operations* of the class, i.e., the operations associated with the objects of the class. There is, then, the issue of modelling so-called inheritance mechanism, by which more specific elements incorporate the structure and behaviour of more general elements.

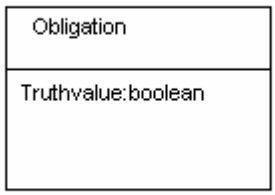
Such more complex notions are not employed in the conceptual model for legal risk analysis. However, it might be useful to know how these should be translated. In this subsection, we analyse the attribute construct. In the next subsection, we briefly explain how inheritance is dealt with. For simplicity's sake, we leave the modelling of the operation construct for later.

In UML an attribute $a$ of type $T$ for class $C$ associates with each instance of $C$ an instance $a$ of $T$. Attributes are unique within a class, but two classes may have the same attribute, possibly of different types. Formally, an attribute $a$ of type $T$ for class $C$ corresponds to a binary predicate $a$ for which the following FOL assertion holds:

$$(10) \quad \forall x, y : (C(x) \wedge a(x, y)) \rightarrow T(y)$$

The assertion states that, for each instance $x$ of class $C$, an object $y$ related to $x$ by $a$ is an instance of $T$.

Graphically, attributes are shown in the second compartment of the class box, as in the following example.




Roughly, this diagram says that any obligation (class) has a truth-value (attribute), this one being Boolean (type). By the latter, we mean that an obligation is in general either true (1) or false (0). The FOL translation is straightforward:

$$\forall x, y : (obligation(x) \wedge truth - value(x, y)) \rightarrow boolean(y)$$

### 3.3.3 Inheritance

Given the above, the inheritance mechanism is easily dealt with. Such a mechanism is modelled as follows. As just said, the fact that class $C$ has some attribute $a$ of type $T$ means that we have:

$$(11) \quad \forall x, y : (C(x) \wedge a(x, y)) \rightarrow T(y)$$

Now, suppose class $C$ generalises another class $C_1$. This means that the following also holds:

$$(12) \forall x : C_1(x) \rightarrow C(x)$$

In first-order logic, one might immediately infer that the following implication holds as well:

$$(13) \forall x : (C_1(x) \wedge a(x, y)) \rightarrow T(y)$$

Therefore, the attributes of the more-general class each are correctly inherited by the more-specific class.

The above mechanism is usually referred to as “single inheritance”. Although this is not mentioned in subsection 3.1, we can also create classes that have more than one parent. The child class will, then, have all the attributes of its parents. This is called “multiple inheritance”. Multiple inheritance can be dealt with along similar lines.

# 4 OUTLINE OF A CONCRETE SYNTAX

By a concrete syntax, we mean the graphical appearance of the elements of the language (icons in the UML terminology). We here show how to extend the CORAS concrete syntax with concepts relevant to legal analysis. Much attention is paid to the normative modalities.

In legal risk analysis, parts of the target of evaluation (ToE) will originate from legal issues formulated in legal texts such as laws and contracts. This motivates that ToE descriptions not only cover technical and organisational issues but also the legal issues. Legal texts have a tendency to be complex and also hard to read by laymen. Since participants of legal risk analyses will include people that are not legal experts, we claim that standard modelling techniques can be applied to give these participants a better grip of the legal issues.

As stated in the requirements, UML does not have a formal semantics. This means that a formal semantics of our language, which is based on UML use case diagrams, also must provide formalisation of use case diagrams. In the following we present a formalisation of our language based on the STAIRS semantics of UML sequence diagrams. The motivation for this is the observation that a use case represents a set of scenarios. A scenario may be viewed as a set of traces. In STAIRS the semantics of a sequence diagram is defined by means of traces. This should mean that the semantic foundation of sequence diagrams in STAIRS could be used as a semantic foundation of use cases as well. On a syntactic level this should mean that use cases can be viewed as references to sequence diagrams. It is, however, important to notice that sequence diagrams contain a lot more information than a use case diagram. What we are aiming for is not a translation or transformation from use cases to sequence diagrams but a characterisation of the constraints a use case model places on a sequence diagram specification, or, in other words, how a sequence diagram specification may fulfil a use case model.

## 4.1 STAIRS[13]

A UML sequence diagram describes the interaction between communicating *lifelines*. Let $L$ be the set of lifelines and let $M$ be the set of all messages such that $(s,t,r) \in M$ is message with name $s$, transmitter $t$ and receiver $r$ where $t,r \in L$. Let $E$ be the set of events. An event is either the output $(!,m) \in E$ or the input $(?,m) \in E$ of a message $m \in M$. We use $!m$ and $?m$ as shorthand notation for output and input events respectively, or $!m@i$ and $?m@j$ if we want to explicitly show the lifeline on

[13] This presentation is based on Ø. Haugen, K.E. Husa, R.K. Runde, K. Stølen, Why timed sequence diagrams require three-event semantics, to appear in LNCS, 2005, and R.K. Runde, Ø. Haugen, K. Stølen, Refining UML interactions with explicit and implicit nondeterminism, working draft, 2005.

which the events occur on. $l.e$ denotes the lifeline of event $e$, and $e.l$ the set of events occurring on lifeline $l$. Let **sd** $d$ denote a sequence diagram. $ll.(\textbf{sd}\ d)$ denotes the set of lifelines in **sd** $d$. The set $E^*$ denotes the set of finite sequences of events, while the set $E^\infty$ denotes the set of infinite sequences of events. A trace $t$ is a finite or infinite sequence of events $t \in E^\omega = E^* \cup E^\infty$. We let $H$ denote the set of all traces. We let $\langle\ \rangle$ represent the empty trace and $h_1 \hat{}\, h_2$ the concatenation of traces $h_1$ and $h_2$

### 4.1.1 Syntax

Sequence diagrams are composed with the following operators.[14]

- **skip**: the empty sequence diagram
- $d_1$ **seq** $d_2$: weak sequential composition
- $d_1$ **par** $d_2$: parallel composition
- $d_1$ **alt** $d_2$: non-deterministic choice
- $d_1$ **xalt** $d_2$: explicit non-deterministic choice
- **refuse** $d$: all behaviour than include $d$ are negative
- **assert** $d$: noting except the behaviour described by $d$ is allowed

$D$ denotes the set of all syntactically correct sequence diagrams. Syntactically we may view a sequence diagram as a term. We let $(\textbf{sd}\ d)|_p$ denote the sub-term of **sd** $d$ at position $p$. If we, for example, let

$$\textbf{sd}\ d = (!m\ \textbf{seq}\ ?m)\ \textbf{alt}\ (!n\ \textbf{seq}\ ?n)$$

we have

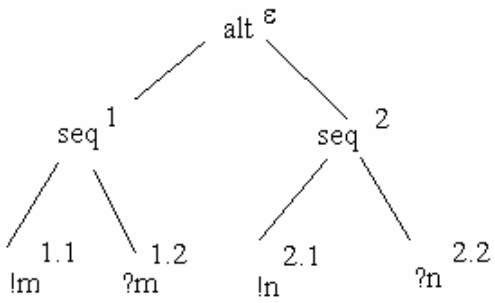
$$(\textbf{sd}\ d)|_\varepsilon = \textbf{sd}\ d = (!m\ \textbf{seq}\ ?m)\ \textbf{alt}\ (!n\ \textbf{seq}\ ?n)$$

$$(\textbf{sd}\ d)|_1 = !m\ \textbf{seq}\ ?m$$

$$(\textbf{sd}\ d)|_{2.1} = !n$$

[14] There exist other operators as well, but they will not be considered in this presentation.

This is shown in the following syntax tree:




We define a sub-diagram relation

$$(\mathbf{sd}\ d_2) \lhd (\mathbf{sd}\ d_1) = \exists p : (\mathbf{sd}\ d_1)\,|_p = (\mathbf{sd}\ d_2)$$

From a formal point of view, this means that $d_2$ is a sub-formula of $d_1$. In terms of syntax trees, $d_2$ is a sub-tree of $d_1$.

### 4.1.2 Semantics

In the standard (informal) UML semantics a trace may be categorised as positive, negative or inconclusive relative to a sequence diagram. Given a sequence diagram the trace universe $H$ is partitioned into three sets of traces, the set of positive traces $P$, the set of negative traces $N$ and the set of inconclusive traces $I$ such that $I = H \setminus (P \cup N)$. The semantics of a sequence diagram is given by the pair ($P$, $N$).

STAIRS provides a formalisation and generalisation of this semantics. A sequence diagram defines a set of obligation alternatives, where an obligation alternative $o$ is a pair of sets of traces ($P$,$N$) and each obligation alternative provide a separate partitioning of $H$. Let $O = 2^H \times 2^H$ be the set of all interaction obligations. $[\![\mathbf{sd}\ d]\!] \subseteq O$ denotes the set of obligation alternatives described by $d$, and is defined recursively as follows

$$[\![(k,m)]\!] = \{(\{\langle (k,m) \rangle\}, \varnothing)\}\text{, for } (k,m) \in E$$

$$[\![\mathbf{skip}]\!] = \{(\{\langle\ \rangle\}, \varnothing)\}$$

$$[\![\mathbf{refuse}\ d]\!] = \{(\varnothing, p \cup n) \mid (p,n) \in [\![d]\!]\}$$

$$[\![\mathbf{assert}\ d]\!] = \{(p, n \cup (H \setminus p)) \mid (p,n) \in [\![d]\!]\}$$

$$[\![d_1\ \mathbf{alt}\ d_2]\!] = [\![d_1]\!] \oplus [\![d_2]\!]$$

$$[\![d_1\ \mathbf{xalt}\ d_2]\!] = [\![d_1]\!] \cup [\![d_2]\!]$$

$$[\![d_1\ \mathbf{par}\ d_2]\!] = [\![d_1]\!] \,||\, [\![d_2]\!]$$

$$[\![d_1 \textbf{ seq } d_2]\!] = [\![d_1]\!] \succ [\![d_2]\!]$$

where

$$O_1 \oplus O_2 = \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\}$$

$$O_1 \parallel O_2 = \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\}$$

$$(p_1, n_1) \parallel (p_2, n_2) = (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1))$$

$$O_1 \succ O_2 = \{o_1 \succ o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\}$$

$$(p_1, n_1) \succ (p_2, n_2) = (p_1 \succ p_2, (n_1 \succ (p_2 \cup n_2)) \cup (n_2 \succ p_1))$$

and $s_1 \parallel s_2$ and $s_1 \succ s_2$ denotes respectively the parallel composition (merging) and weak sequential composition of the traces in the sets $s_1$ and $s_2$. We define a trace filtering operator

$$A\Theta\langle\ \rangle = \langle\ \rangle$$

$$A\Theta(\langle e \rangle^\frown h) = \langle e \rangle^\frown (A\Theta h) \Leftarrow e \in A$$

$$A\Theta(\langle e \rangle^\frown h) = A\Theta h \Leftarrow e \notin A$$

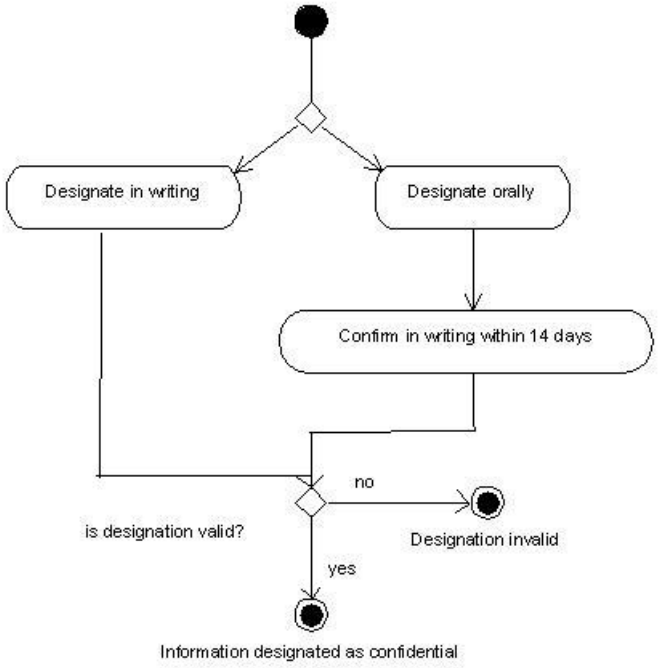
and let $\overline{\Theta}$ be an operator that filters pairs of sequences with respect to a set of pairs. We then define

$$s_1 \parallel s_2 = \{h \in H \mid \exists p \in \{1,2\}^\infty :$$
$$\pi_2((\{1\} \times E)\overline{\Theta}(p,h)) \in s_1 \wedge$$
$$\pi_2((\{2\} \times E)\overline{\Theta}(p,h)) \in s_2\}$$

$$s_1 \succ s_2 = \{h \in H \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in L :$$
$$e.l\Theta h = e.l\Theta h_1 \,^\frown e.l\Theta h_2\}$$

where $\pi_2(p,h) = h$ is the projection on the second element of a pair.

Here is an example. According to article 8.1 of the Consortium Agreement template for use by an SME cluster, the parties agree to advise and notify the other as to which part of the disclosed information constitutes confidential information. The following activity diagram models the procedure that, according to article 8.2, the parties must follow:




Suppose we focus on the upper branch. In an interaction diagram the process may look like Figure 3 below, with the obvious definition:

des_w **alt** (des_o **seq** conf).

The "alt" operator (*alias* "or" exclusive) defines a choice between two sets of messages, which here are separated by a line. The other is the "seq" operator; it denotes sequential composition. We will not calculate the explicit traces, but it is a straightforward, mechanical task that only requires patience. Note that the semantics says nothing about the period within which the confirmation must have been made. Extending STAIRS to handle time constraints is a straightforward matter. The details can be found in the paper on which this presentation is based.[15]

[15] See footnote 13 above.

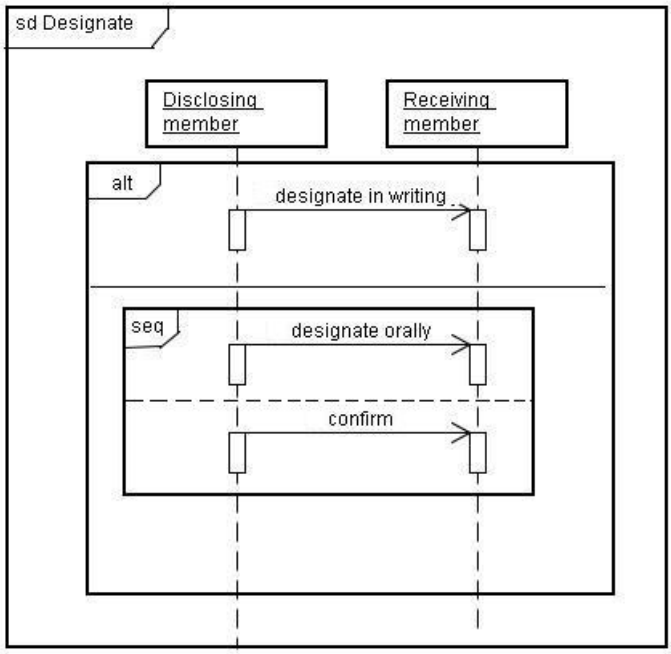



Figure 3 Designate information as confidential

### 4.1.3 Refinement

Refinement is an essential part of STAIRS. Informally refinement is making a specification more concrete by classifying positive traces as negative or making it more detailed by classifying inconclusive traces as either positive or negative. An obligation alternative $(p_2,n_2)$ refines the obligation alternative $(p_1,n_1)$, denoted $(p_1,n_1) \rightsquigarrow (p_2,n_2)$ if and only if

$$n_1 \subseteq n_2 \wedge p_1 \subseteq p_2 \cup n_2$$

A sequence diagram **sd** $d_2$ refines a sequence **sd** $d_1$, written $(\mathbf{sd}\ d_1) \rightsquigarrow (\mathbf{sd}\ d_2)$, iff

$$\forall o \in [\![\mathbf{sd}\ d_1]\!] : \exists o' \in [\![\mathbf{sd}\ d_2]\!] : o \rightsquigarrow o'$$

## 4.2 Use Case Diagrams

### 4.2.1 Basic idea

We write **uc** $a$ to name a use case, and **ud** $A$ to name a use case diagram. The function $uc$(**ud** $A$) returns the set of names of use cases in **ud** $A$. The function $act$(**uc** $a$) returns the names of the actors associated with **uc** $a$, and $act$(**ud** $A$) returns the names of the actors in the diagram. In the following example we have

$uc$(**ud** $A$) = {**uc** $a$, **uc** $b$}

$act$(**ud** $A$) = {$I$,$J$}

$act$(**uc** $a$) = {$I$,$J$}

$act$(**uc** $b$) = {$I$}

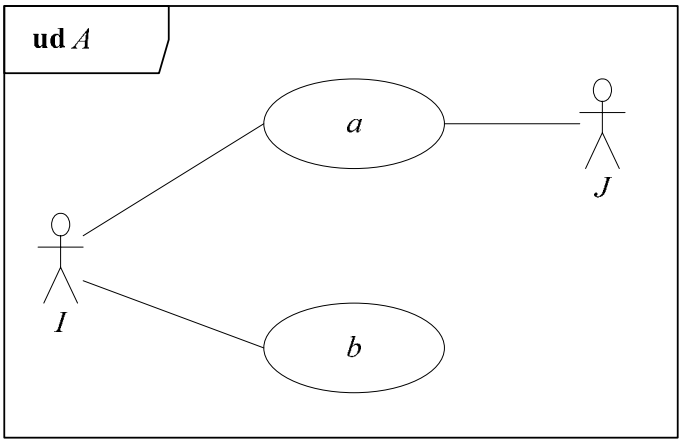



Figure 4 Use case diagram

For each use case **uc** $a$ we associate a sequence diagram **sd** $a$ such that

$$[\![\text{uc } a]\!] = [\![\text{sd } a]\!]$$

Since sequence diagrams may be composed, a use case diagram (containing possibly several unrelated use cases) may be seen as one composite sequence diagram. For a use case diagram **ud** $A$ we associate a sequence diagram **sd** $A$ such that

$$(\textbf{uc } a) \in uc(\textbf{ud } A) \Rightarrow (\textbf{sd } a) \triangleleft (\textbf{sd } A)$$

$$[\![\textbf{ud } A]\!] = [\![\textbf{sd } A]\!]$$

Intuitively, this means that any use case diagram is mapped onto an interaction diagram. Figure 5 below illustrates the basic idea.

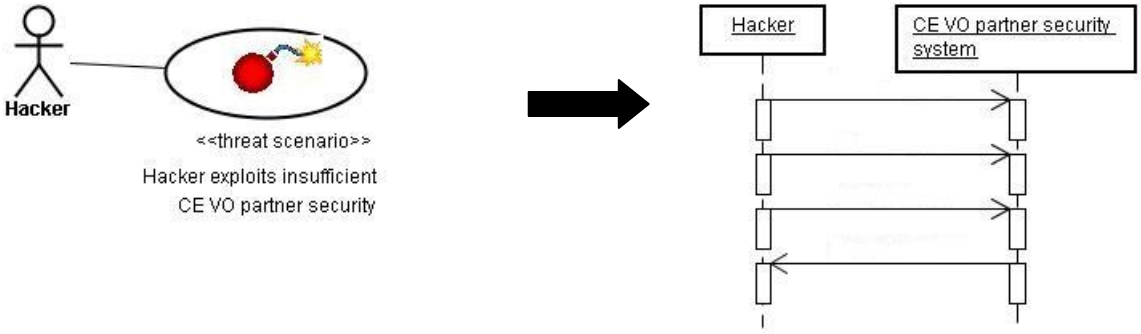



Figure 5 The hacker example (CE scenario)

The left-hand side shows a CORAS UML use case diagram. It describes one of the threat scenarios that were identified during the legal risk analysis of the CE scenario; the unwanted incident is that SI confidential information is disclosed.[16] The right-hand side of Figure 5 shows the corresponding sequence diagram. Objects are arranged along the horizontal axis and messages, ordered in increasing time, along the vertical axis. This simple diagram shows the sequence of messages that needs to pass between objects so as to complete a use case for exploiting insufficient CE VO partner security. The arrows from left to right represent the hackers' attempts to break into the system. The arrow from right to left indicates that the last attempt was successful – the system is now giving information to the hacker.

This diagram is itself part of a bigger diagram involving many different use cases organised by specifying relationships among them, such as <<include>> and <<initiate>>. There is, then, the task of analysing the latter relationships – task to which we now turn.

### 4.2.2 Include

We suggest analysing the <<include>> relationship in terms of the sub-diagram relationship. Thus, Figure 6 below is interpreted as

$$(\mathbf{sd}\ a) \lhd (\mathbf{sd}\ b)$$

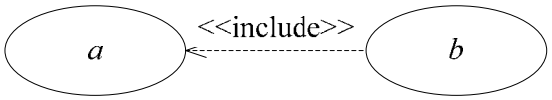



Figure 6 <<include>>

[16] See D15 Report on Legal Issues (page 18) and D15 Appendix A.

From a logic-based viewpoint, this means that the temporal formula associated with use case *a* is a sub-formula of the temporal formula onto which use case *b* has been mapped.

Here is an example. Consider, first, the use case labelled with “unfaithful employee discloses confidential information”:

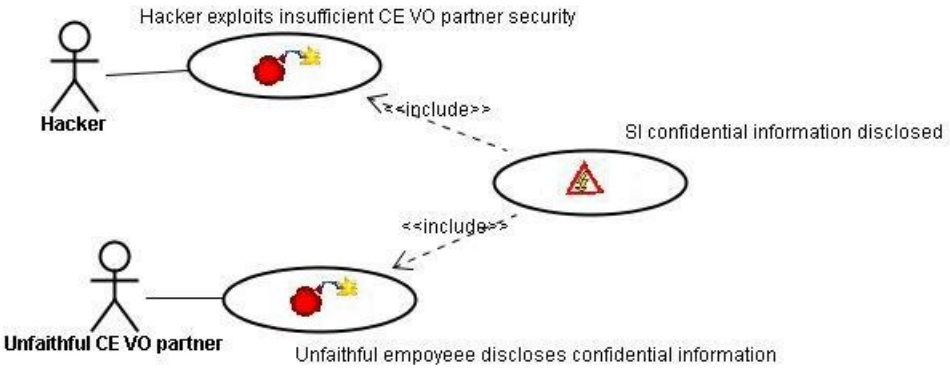



Figure 7 SI confidential information disclosed – use case diagram (CE scenario)

The interaction diagram to be associated with this use case is obvious:

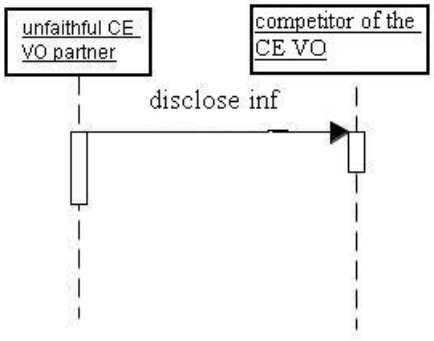



Figure 8 The unfaithful partner (CE scenario)

In order to obtain the representation of Figure 7, we need to combine the last two sequence diagrams, by means of the “alt” operator mentioned above. We get:

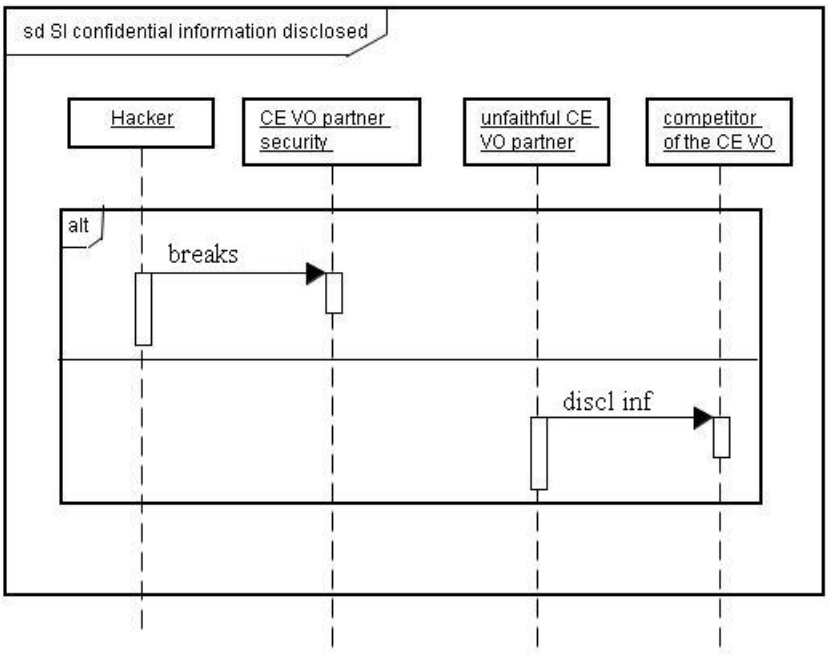



Figure 9 SI confidential information disclosed – sequence diagram (CE scenario)

We have identified the need to, in some situations, restrict the <<include>> construct. Below we present three special cases of <<include>>, namely <<initiate>> which originate from the CORAS language, an AND-construct and an OR-construct.

### 4.2.3 Initiate

In addition to <<include>> we use an arrow labelled <<initiate>>. In the example below we interpret <<initiate>> as

$$\exists(\mathbf{sd}\ d) : (\mathbf{sd}\ b) = a\ \mathbf{seq}\ d$$

or in other words as a kind of prefix. Since this implies that $(\mathbf{sd}\ a) \triangleleft (\mathbf{sd}\ b)$, we see that <<initiate>> is a special case of <<include>>

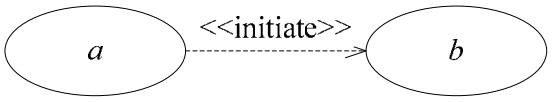



Figure 10 <<initiate>>

Consider the following use case diagram:

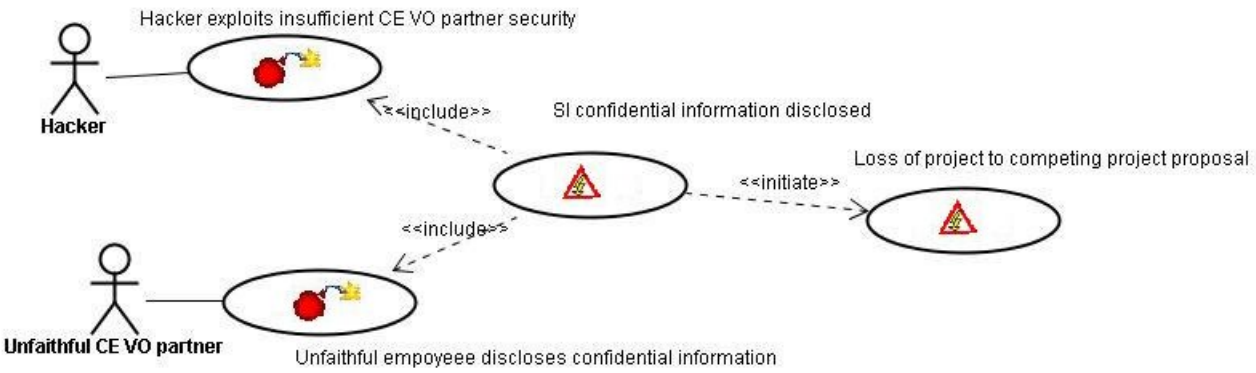



Figure 11 SI confidential information disclosed – use case diagram (continued)

The corresponding sequence of messages is shown below:

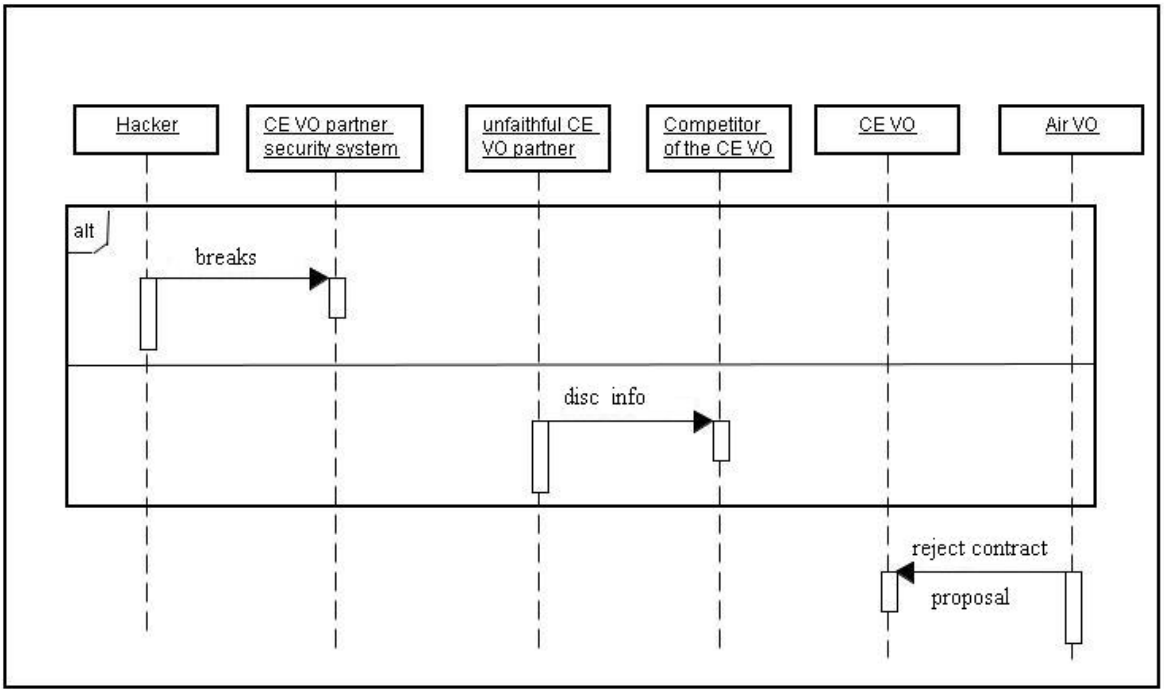



Figure 12 SI confidential information disclosed - sequence diagram (continued)

### 4.2.4 AND vs. OR

Below we show how the AND- and OR-constructs can be analysed. The first one can be graphically rendered as in Figure 13.

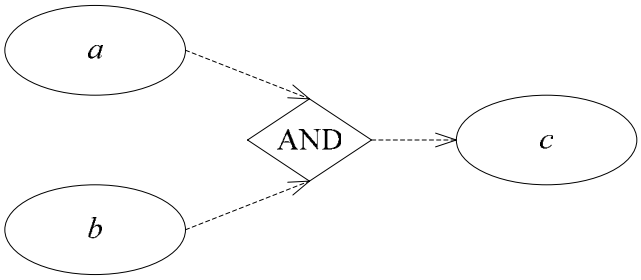



Figure 13 AND

This construct is interpreted as

$$(a \textbf{ par } b) \triangleleft (\textbf{sd } c)$$

An illustration is given in section 4.4.1 below – the example also involves the deontic modalities.

The OR-construct can be graphically rendered as in Figure 14 below.

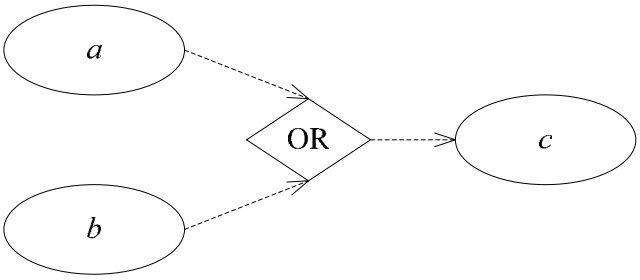



Figure 14 OR

The OR is interpreted as

$$(a \textbf{ alt } b) \triangleleft (\textbf{sd } c)$$

An illustration of the OR construct is given supra, section 4.2.3, Figure 11. From this example, it emerges that both constructs are special cases of Figure 15:

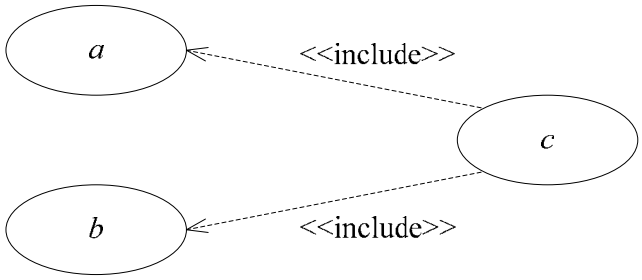



Figure 15 The general case

Figure 15 can be formalised as follows:

$$(\mathbf{sd}\ a) \triangleleft (\mathbf{sd}\ c) \wedge (\mathbf{sd}\ b) \triangleleft (\mathbf{sd}\ c)$$

### 4.2.5 Specialisation

Other useful constructs are specialisation and realisation. In the example shown below, **uc** $b$ is a specialisation of **uc** $a$, and hence we would like **sd** $b$ to be a specialisation of **sd** $a$. We do however find the description of specialisation of behaviour in the UML standard a bit ambiguous. The standard states that "[s]pecializing [a sequence diagram] is simply to add more traces to those of the original. The traces defined by the specialization is [*sic*] combined with those of the inherited [sequence diagram] with a union", but further that "the [use case] owning [a sequence diagram] may be specialized, and in the specialization the [sequence diagram] may be redefined. Redefining [a sequence diagram] simply means to exchange the redefining [sequence diagram] for the redefined one [...]. This is similar to redefinition of other kinds of Behavior."[17]


Figure 16 Specialisation

Since we view sequence diagrams and use cases as essentially the same thing and not as essentially different things (in UML a sequence diagram *is* behaviour while a use case *owns* behaviour), we will not distinguish specialisation of sequence diagrams from specialisation of use cases. With the reasonable restriction on

[17] UML Superstructure 2.0 Draft Adopted Specification, OMG document: ptc/04-10-02, page 526.

specialisation of sequence diagrams (that specialisation is to add traces without removing any) we can define specialisation as the opposite of refinement.[18] Hence, we get that

$$(\mathbf{sd}\ b) \sim> (\mathbf{sd}\ a)$$

when **uc** $b$ is a specialisation of **uc** $a$.

### 4.2.6 Realisation

In the diagram below **uc** $b$ is a realisation of **uc** $a$. The standard states that "[r]ealization can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, etc."[19]

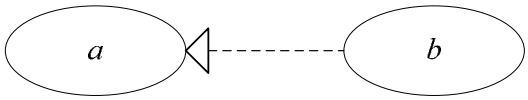



Figure 17 Realisation

We choose to view realisation as refinement, and the formalisation of the diagram above is then

$$(\mathbf{sd}\ a) \sim> (\mathbf{sd}\ b)$$

## 4.3 Actors and Lifelines

Use cases may be associated with actors that participate in the use cases. In the corresponding sequence diagrams we wish to represent these actors by lifelines, but as will be clear below we do not want this to be a one-to-one mapping. If we read the UML standard, it is not clear what the relationship between actors and lifelines is, but it seems that both concepts in some sense refer to roles. The standard states that "[a]n actor specifies a role played by a user or any other system that interacts with the subject"[20], while a lifeline is a name that represents a ConnectibleElement which

[18] See e.g. Wing, J.M. Subtyping for distributed object stores. In, Bowman, H. and Derrick, J. (eds.), Formal Methods for Open Object-based Distributed Systems, vol. 2, pages 305-318, Chapman & Hall, 1997, and Harel, D. and Kupferman, O. On object systems and behavioral inheritance. IEEE Transactions on Software Engineering, 28(9): 889-903, 2002.

[19] UML Superstructure 2.0 Draft Adopted Specification, OMG document: ptc/04-10-02, page 131.

[20] Op. cit., page 634.

again is defined as “an abstract metaclass representing a set of instances that play roles of a classifier”.[21]

We introduce the concept of agents and let agents be the instances referred to in the UML standard. Let $A$ denote the set of all agents. To keep things simple we define a role $R$ as being a nonempty set of agents, $R \subseteq A$, $R \neq \varnothing$, and let an actor be simply a role. In the example below we therefore have that $I \subseteq A$. If, given two roles $R_1$ and $R_2$, we have that $R_1 \subseteq R_2$, we call $R_1$ the sub-role of $R_2$ and $R_2$ the super-role of $R_1$. We use this to define specialisation of actor. In the example below, we have that $I$ is a super-role of $J$ and $J$ is a sub-role of $I$.

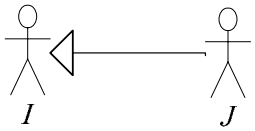



Figure 18 Specialisation of actor

In STAIRS, as well as in the standard, a lifeline is a name. We introduce a mapping $role \in L \to 2^A$ that for each lifeline $l$ returns the role (set of agents) that $l$ represents. If we have that an actor is associated with a use case we say that there exists a lifeline in the associated sequence diagram which role is the super-role of the actor. Since there may be lifelines in the sequence diagram not visible at the use case level, we do not impose a one-to-one correspondence between actors and lifelines.

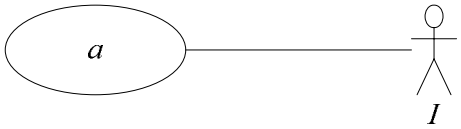



Figure 19 Actor associated with use case

The example above is then interpreted as

$$\exists l \in ll.(\mathbf{sd}\ a) : I \subseteq role(l)$$

Suppose we have the situation described in Figure 20 below. This scenario may be interpreted as either

$$\exists l \in ll.(\mathbf{sd}\ a) : I \subseteq role(l) \wedge J \subseteq role(l)$$

or

$$\exists l_1, l_2 \in ll.(\mathbf{sd}\ a) : l_1 \neq l_2 \wedge I \subseteq role(l_1) \wedge J \subseteq role(l_2)$$

In a given case we may even have that

$$\exists l_1, l_2 \in ll.(\mathbf{sd}\ a) : l_1 \neq l_2 \wedge I \subseteq role(l_1) \wedge I \subseteq role(l_2)$$

[21] Op. cit., page 182.

We will however not prefer any of the interpretations, and allow all cases.

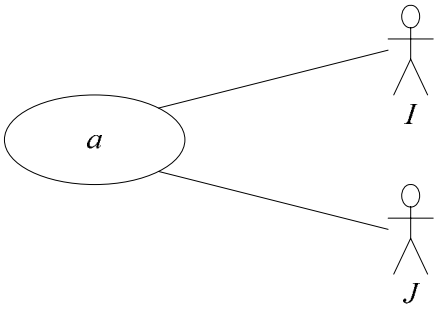



Figure 20 Actors

A task for future work is to give an interpretation of role (and sub-role/super-role) with respect to the behaviour of agents.

## 4.4 Normative Extensions of the UML

Legal norms have effects that bind the roles involved in certain activities by *normative modalities*, such as *it is permitted for* A *to do* X and *it is forbidden for* B *to do* Y. When modelling legal issues, we concentrate on modelling the effect of the relevant legal texts (not the legal texts themselves) and do this by introducing the normative modalities as modelling elements.

Consider a small example where company A and company B decide to cooperate on a shared project to reach some goal. Both companies hold information – respectively "A's information" and "B's information" – that they want to keep confidential (trade secrets) but which they agree to share with each other in the project. In order to protect their interests they make a contract managing the use of this information. An extract of the *effects* of this contract are modelled in the diagram of Figure 21. In this example, one of the effects is that A has permission to access B's information, and another that A is forbidden to distribute B's information. The diagram also shows from where these effects originate (e.g. Contract §3).

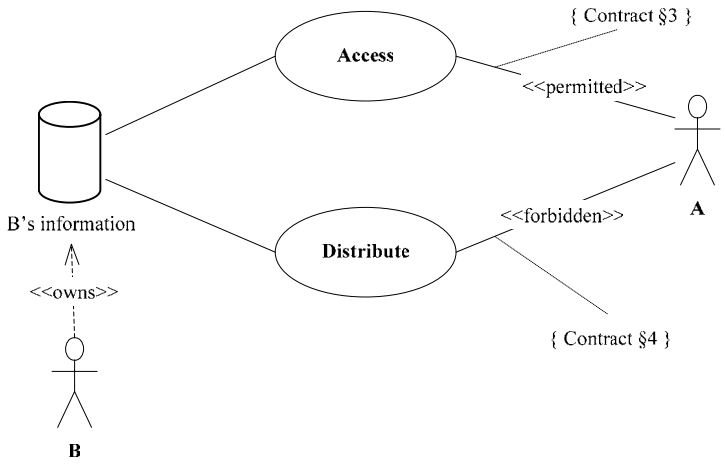



Figure 21 Modelling legal effects

Because we have defined lifelines as representing super-roles of actors, there will sometimes be a need for restricting a sequence diagram to one specific actor. We define a restriction (**sd** $a$)[$J$] meaning that all lifelines $l$ in the sequence diagram with $J \subseteq role(l)$ are replaced by a lifeline $l'$ with $role(l') = J$. This restriction is defined recursively as follows:

(**op** $d$)[$J$] = **op** ($d$[$J$]) for **op** $\in$ {**refuse**, **assert**}

($d_1$ **op** $d_2$)[$J$] = ($d_1$[$J$]) **op** ($d_2$[$J$]) for **op** $\in$ {**seq**, **par**, **alt**, **xalt**}

($k$,$m$)[$J$] = ($k$, $m$[$J$]) for ($k$,$m$) $\in$ $E$

($s$, $t$, $r$)[$J$] = ($s$, $t$[$J$], $r$[$J$]) for ($s$, $t$, $r$) $\in$ $M$

$l$[$J$] = $l$ if $agents(l) \cap J = \varnothing$

$l$[$J$] = $l'$ such that $agents(l') = J$ if $J \subseteq$ role($l$)

### 4.4.1 Towards a Formalisation

The normative modalities can be included in the use case syntax by using the stereotypes <<permitted>>, <<forbidden>> and <<obligated>>. This is shown in Figure 22.

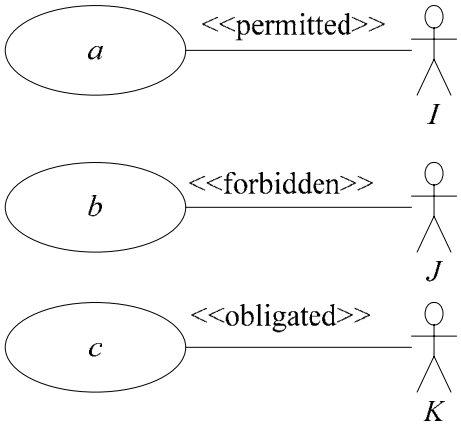



Figure 22 Normative modalities

We aim at a formalisation where the interpretation of the diagram is equivalent to the deontic sentences:

| | | |
|---|---|---|
| $P_I a$ | read as | "it is permitted that $I$ sees to it that $a$" |
| $F_J b$ | | "it is forbidden that $J$ sees to it that $b$" |
| $O_K c$ | | "it is obligatory that $K$ sees to it that $c$". |

The main idea is to make an interpretation of these sentences by means of the sequence diagram operators. A first attempt is shown below:

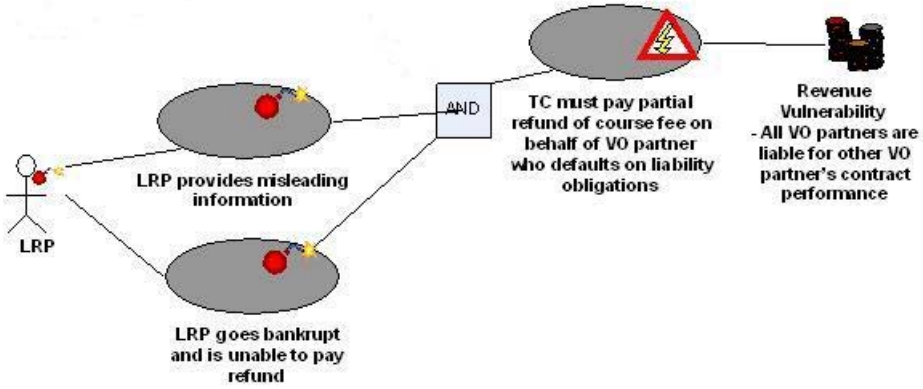
$$P_I a = (\textbf{sd}\ a)[\underline{I}]$$
$$F_J b = \textbf{refuse}\ (\textbf{sd}\ b)[J]$$
$$O_K c = \textbf{assert}\ (\textbf{sd}\ c)[K]$$

The **refuse (**also called **neg)** and the **assert** operators have a distinctly deontic flavour. These operators have been introduced in the UML to make sequence diagrams more suitable to express requirements. As mentioned, the standard defines the semantics as a set of traces. More precisely, "the semantics of an interaction is given as a pair of sets of traces"[22], representing "valid traces and invalid traces"[23]. Roughly speaking these two types of traces are associated with the operators **assert** and **refuse/neg**, respectively. However, it is not obvious what valid and invalid really means, and the standard does not provide any example. As some have suggested, it is possible to interpret them as `necessary vs. forbidden'

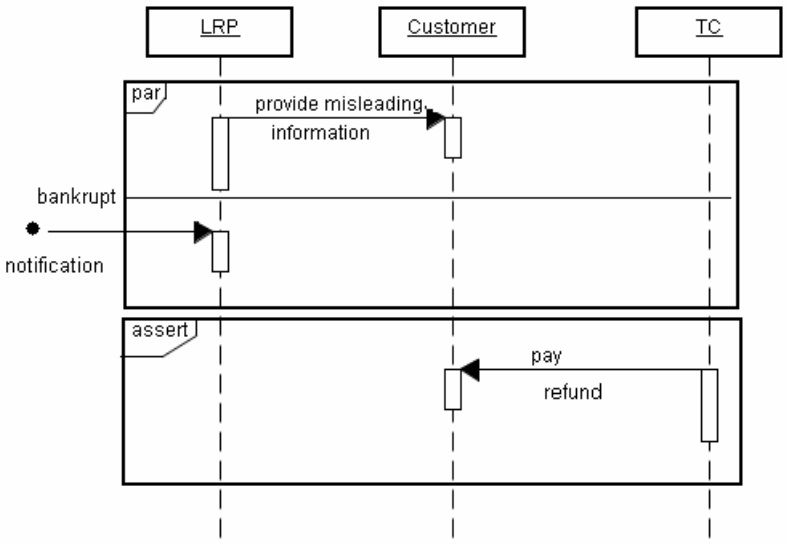
[22] UML Superstructure 2.0 Draft Adopted Specification, OMG document: ptc/04-10-02, page 419.

[23] Ibid.

or `must vs. must not'. Whereas the **assert** operator defines traces that *must* occur at a given point in a scenario, in some sense or other of 'must', the **refuse** operator defines traces that *must not* occur.[24] Of course, the problem is to specify what sense of `must' this is. For the purposes of the present discussion, we need not enter into this issue. It suffices to have identified in the UML the presence of operators that have a deontic flavour. These will be used to represent the normative concepts, broadly conceived. An example taken from the legal risk analysis of the e-learning scenario[25] is given below; it also provides an illustration of the AND-construct:




In the original example, there is an additional use case, saying that the dissatisfied customer demands refund of course fee. To keep things simple, we omit this additional use case. The corresponding sequence of messages is shown below.




[24] Harald Störrle. Assert, Negate and Refinement in UML-2 Interactions. In Jan Jürjens, Bernhard Rumpe, Robert France, and Eduardo B. Fernández, editors, Workshop on Critical Systems Development with UML (CSDUML'03, Proceedings), pages 79-94.

[25] The results of the risk analysis of this scenario will be presented in the next report.

This is not shown in this example, but it might very well be the case that $I$, $J$ and $K$ are related to the same use case with different modalities. We can use **xalt** to compose the different restrictions on the sequence diagram.

$$(\textbf{sd}\ d)[I]\ \textbf{xalt refuse}\ (\textbf{sd}\ d)[J]\ \textbf{xalt assert}\ (\textbf{sd}\ d)[K]$$

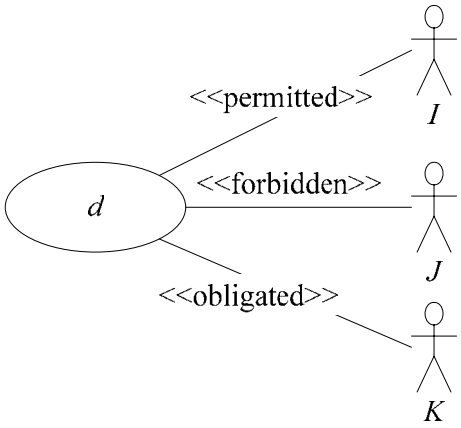



Figure 23 Normative modalities (continued)

In order to carry out this formalisation in a sound manner, we will probably need a model theoretic interpretation of sequence diagrams. Giving this model theoretic interpretation and using it for giving meaning to normative modalities in sequence diagrams is one of the tasks identified for further work.

## 4.5 Threat Modelling

In order to model threats related to legal issues, we need to integrate the approach above with the threat modelling features of the CORAS profile. Figure 24 shows a threat related to one of the legal effects in the above example. Company B acts as stakeholder and company A acts as a threat agent in this example. Further, B's information is viewed as one of A's assets and the threat scenario includes an activity that for A is forbidden.

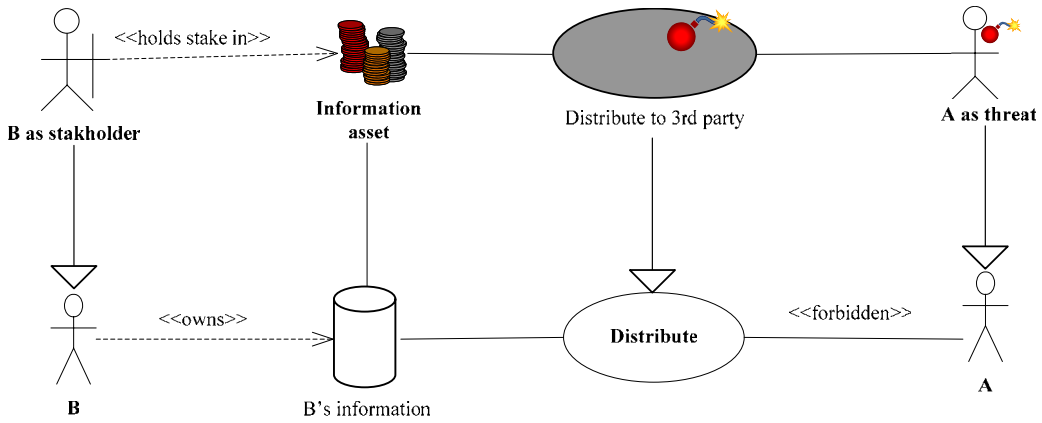



Figure 24 Modelling legal threats

One of the tasks identified for further work is to give an interpretation in the trace-based framework of notions from the CORAS language, such as threat scenario and treatment.

# 5 RELATION TO THE TRUSTCOM CONCEPTUAL MODELLING

The developments presented in this appendix aim ultimately to contribute to the overall TrustCoM project, by providing a bridge between the UML-based conceptual modelling done within AL1 and the legal issues part of the project.

First of all, the conceptual modelling done within WP1 makes an extensive use of UML class diagrams. In particular, ID 1.1.2 “Preliminary Conceptual Model for the TrustCoM Framework” (v 1.0) contains twenty-six UML diagrams, sixteen of which are class diagrams. These are:

- Figure 1 - VO Specification showing business and operational levels (p. 16)
- Figure 2 - VO Specification (Operational Level) (p. 18)
- Figure 7 - Trust: Class Diagram (p. 34)
- Figure 9 - Abstract permission model (p. 45)
- Figure 10 - Abstract role-based user model (p. 46)
- Figure 11 - Abstract user model (p. 47)
- Figure 12 - Service model (p. 48)
- Figure 13 - Service resource type and action model (p. 49)
- Figure 14 - Policy resource type and action model (p. 50)
- Figure 17 - Domains, Policies and Objects (p. 54)
- Figure 18 - Monitoring: structural model (p. 60)
- Figure 19 - Static model of General VO Agreements (p. 76)
- Figure 20 - Static model of Service Level Agreements (p. 79)
- Figure 22 - Negotiation: class diagram (p. 95)
- Figure 23 - Negotiation Policy: class diagram (p. 96)
- Figure 24 - Negotiation: relationships to other subsystems (p. 96)

Of the remaining ten diagrams, only five are UML diagrams. The first two are use case diagrams:

- Figure 25 – Use case for direct negotiation (p. 97)
- Figure 26 – Use case for brokered negotiation (p. 97)

The third one is a sequence diagram:

- Figure 16 – Functions of policy deployment (p. 53)

The last two are activity diagrams:

- Figure 15 – Policy life-cycle (p. 52)

- Figure 21 - Evaluation of a simple obligation (p. 86)

The formalisation in terms of first-order logic proposed in Section 3 can be used to check the consistency of the first kind of diagrams. The construction of a UML class diagram is not constrained by the UML standard. Therefore, there is no guarantee that the model does not contain an inconsistency. Should the model be inconsistent it might not be possible to implement it efficiently. The process of transforming a UML model into code through a mapping to an implementation language is usually called "forward engineering".[26] Formal methods provide a framework for expressing and checking the consistency of the diagrams to be implemented, and thus help verify when forward engineering is, or is not, worth a try. In this respect, our work also provides a bridge between the modelling work in AL1 and the matching implementation in AL2.

Of course, thus far we have only addressed the consistency-checking issue with regard to UML class diagrams - the part of the language based on UML sequence diagrams is still under development. Having at our disposal a similar algorithm for checking the consistency of sequence diagrams would certainly be an advantage. In the coming months, TrustCoM will indeed shift the emphasis from the static aspects of VOs to their dynamic aspects.[27] We believe that our formalisation of sequence diagrams can help study the dynamic aspects of VOs, especially - but not exclusively - in the contract area. One of our main concerns was to extend the UML with facilities for specifying obligations and related deontic concepts. If there is an area (or, to use the terminology employed in the DoW, a subsystem) that might benefit from these normative extensions of the UML, it is the area of contract and SLA in the first place. For instance, it is natural to ask if the approach outlined in this report can shed new light on the behavioural model of contract violations and fulfilment defined in ID 1.1.2 "Preliminary Conceptual Model for the TrustCoM Framework" (v 1.0, p. 85). This is a topic for future research.

The elaboration of a language for legal risk analysis was initially conceived as a subtask of WP9. However, we stress that the resulting framework is domain-independent, in the sense that it can be customised relatively easily to the other areas encompassed by the conceptual modelling done in AL1.

[26] See G. Booch, J. Rumbauch and I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1998, p. 239.

[27] As observed in the second review report "research on dynamic aspects of VOs [] constitutes the core of the planned work in TrustCoM, and represents the bulk of its contribution to innovation" (p. 17).

# 6 CONCLUSION

We have shown how to incorporate more information relevant to legal risks analysis into the UML graphical models, and how to give a precise semantics to the resulting language. Of course, much work remains to be done on both the language itself and its application to the analysis of legal risks. However, we believe that the framework outlined in this appendix is a first step in the right direction.

The concrete syntax is based on the existing CORAS graphical language for security risk analysis. The contribution of this appendix with regard to the CORAS language is on the one hand an extension of the CORAS language to make it suitable for carrying out legal risk analysis, and on the other hand the STAIRS based formalisation in which use cases are interpreted as traces in sequence diagrams.

The concrete syntax was used and tested against the TrustCoM e-learning testbed scenario, and was capable of capturing the aspects of the scenario as well as being easily understandable for the practitioners. We have also started to test the formalisation based on the mapping of use cases upon sequence diagrams. This has been done by applying it to both the TrustCoM scenario on collaborative engineering and the TrustCoM e-learning scenario. In the future, however, the language needs to be used in a more consistent way, and it must be tested against the results obtained from further legal risk analysis.

Two other tasks for future work have been identified. One is to define a mapping between the formalisation of the abstract syntax (conceptual model) and the formalisation of the concrete syntax. The other is to provide an interpretation in the model-theoretic foundations of deontic logic. These are only hints, to which a later report will be devoted.

# Table of figures