Model Checking Contracts – A Case Study*

Gordon Pace[1], Cristian Prisacariu[2], and Gerardo Schneider[2]

[1] Dept. of Computer Science and AI, University of Malta, Msida, Malta
[2] Department of Informatics – University of Oslo,
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
gordon.pace@um.edu.mt, {cristi,gerardo}@ifi.uio.no

Abstract. Contracts are agreements between distinct parties that determine rights and obligations on their signatories, and have been introduced in order to reduce risks and to regulate inter-business relationships. In this paper we show how a conventional contract can be written in the contract language $\mathcal{CL}$, model the contract and verify properties of the model using the NuSMV model checking tool.

1 Introduction

Internet-based applications involving one or more entities participating in inter-business collaborations, virtual organisations, and web services, usually communicate through service exchanges. Such exchanges are subject to certain understanding on the different roles the participants play, including assumptions on their correct and incorrect behaviours, and their rights and obligations in order to avoid misunderstanding and ambiguities in such business relationships. This motivates the need of establishing an agreement before any transaction is performed, through a *contract*, guaranteeing the rights and duties of each signatory. Such documents may also contain clauses determining penalties in case of contract violations, and be as unambiguous as possible to avoid conflicting interpretations. *Conventional contracts* are documents written in natural language, as one may find in usual judicial or commercial traditional activities. On the other hand, *electronic contracts* (or e-contracts for short) are machine-oriented and as such they must be "understood" by the software responsible for controlling and monitoring the service exchanges. E-contracts might be seen in two different ways: (1) As the executable version of a conventional contract, obtained from the translation of the "paper" version into the electronic one; (2) As contracts by themselves obtained directly from certain software applications, like web services and virtual organisations. For our current purposes, the difference above is irrelevant, though our case study is based on a conventional contract.

Ideally, e-contracts should be shown to be contradiction-free both internally, and with respect to the governing policies under which the contract is enacted.

* Partially supported by the Nordunet3 project "Contract-Oriented Software Development for Internet Services".

K.S. Namjoshi et al. (Eds.): ATVA 2007, LNCS 4762, pp. 82–97, 2007.

Moreover, there must be a run-time system ensuring that the contract is respected. In other words, contracts should be amenable to formal analysis allowing both static and dynamic verification, and thus written in a formal language. In this paper we are interested only in the analysis of the contract itself (statically), and we are not concerned with its relation with policies nor with its enforcement at run-time.

A formal language for writing contracts should be designed as to avoid most of the philosophical problems of deontic logic [11]. Moreover, it should be possible to represent conditional obligations, permissions and prohibitions, as well as *contrary-to-duty obligations* (CTD) and *contrary-to-prohibitions* (CTP). CTDs are statements representing obligations that might not be respected, whereas CTPs are similar statements dealing with prohibitions that might be violated. Both constructions specify the obligation/prohibition to be fulfilled and which is the *reparation/penalty* to be applied in case of violation.

A formal language for writing (untimed) contracts is $\mathcal{CL}$ [13]. The language is tailored to e-contracts, following an action-based approach, and having the following properties: (1) The language avoids most of the classical paradoxes of deontic logic; (2) It is possible to express in the language (conditional) obligations, permissions and prohibitions over concurrent actions keeping their intuitive meaning; (3) It is possible to express CTDs and CTPs; (4) The language has a formal semantics given in a variant of the modal μ-calculus.

The main contribution of this paper is to show how model checking techniques can be applied in the context of contract-oriented software development, in order to determine whether a given contract stipulates what it is supposed to. $\mathcal{CL}$ is used as an intermediary between the contract clauses in plain English and the system specification required by the model checking tool. This use of $\mathcal{CL}$ increases the confidence in the initial formulation of the contract clauses. The model checking method that we present requires to pursue the following steps:

1. Model the conventional contract written in English into the formal language $\mathcal{CL}$;
2. Translate syntactically the $\mathcal{CL}$ specification into the extended μ-calculus $\mathcal{C}\mu$;
3. Obtain a Kripke-like model (a labelled transition system with state propositions — LTS) of the $\mathcal{C}\mu$ formulae;
4. Translate the LTS into the input language of NuSMV;
5. Perform model checking using NuSMV;
6. In case of a counter-example given by NuSMV, interpret it as a $\mathcal{CL}$ clause and repeat the model checking process until the property is satisfied;
7. Finally, repair the original contract by adding a corresponding clause, if applicable.

The paper is organised as follows. In Section 2 we start by presenting the language $\mathcal{CL}$, including an example of the kind of contracts we are dealing with, from which we will extract our case study. Section 3 is the main part of the paper where we first formalise the case study in $\mathcal{CL}$, and afterwards we show how to use model checking and the NuSMV tool to determine whether the contract is

correct with respect to certain desired properties, and how to get feedback as to write the "correct" contract. In Section 4 we analyse related works and conclude by discussing our choice of the model checking tool as well as future work.

2 A Formal Language for Contracts

We present in Fig. 1 a part of a conventional contract between a service provider and a client, where the provider gives access to Internet to the client. We analyse part of this contract in the following section. First we recall the contract language $\mathcal{CL}$; for a more detailed presentation see [13].

Definition 1 (Contract Language Syntax). *A contract is defined by:*

$$\begin{aligned} \mathcal{C}ontract &:= \mathcal{D}\,;\,\mathcal{C} \\ \mathcal{C} &:= \phi \mid \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid [\alpha]\mathcal{C} \mid \langle\alpha\rangle\mathcal{C} \mid \mathcal{C}\,\mathcal{U}\,\mathcal{C} \mid \bigcirc\mathcal{C} \mid \Box\mathcal{C} \\ \mathcal{C}_O &:= O(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \\ \mathcal{C}_P &:= P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \\ \mathcal{C}_F &:= F(\delta) \mid \mathcal{C}_F \vee [\delta]\mathcal{C}_F \end{aligned}$$

The syntax of $\mathcal{CL}$ closely resembles the syntax of a modal (deontic) logic. Though this similarity is clearly intentional since we are driven by a logic-based approach, $\mathcal{CL}$ is *not* a logic. The semantics of $\mathcal{CL}$ are given in an extension of μ-calculus [8] which we call $\mathcal{C}\mu$. In what follows we provide an intuitive explanation of the $\mathcal{CL}$ syntax.

A contract consists of two parts: *definitions* ($\mathcal{D}$) and *clauses* ($\mathcal{C}$). We deliberately let the definitions part underspecified in the syntax above. $\mathcal{D}$ specifies the *assertions* (or conditions) and the atomic actions present in the clauses. ϕ denotes assertions and ranges over boolean expressions including the usual boolean connectives, and arithmetic comparisons like "the budget is more than 200$". We let the atomic actions underspecified, which for our purposes can be understood as consisting of three parts: the proper action, the subject performing the action, and the target of (or, the object receiving) such an action. Note that, in this way, the parties involved in a contract are encoded in the actions.

$\mathcal{C}$ is the general *contract clause*. $\mathcal{C}_O$, $\mathcal{C}_P$, and $\mathcal{C}_F$ denote respectively *obligation*, *permission*, and *prohibition* clauses. $O(\cdot)$, $P(\cdot)$, and $F(\cdot)$, represents the obligation, permission or prohibition of performing a given action. $\wedge$ and $\oplus$ may be thought as the classical conjunction and exclusive disjunction, which may be used to combine obligations and permissions. For prohibition $\mathcal{C}_F$ we have $\vee$, again with the classical meaning of the corresponding operator. α is a compound action (i.e., an expression containing one or more of the following operators: choice "+"; sequence "$\cdot$"; concurrency "&", and test "?" —see [13]), while δ denotes a compound action not containing any occurrence of $+$. Note that syntactically $\oplus$ cannot appear between prohibitions and $+$ cannot occur under the scope of F.

This deed of **Agreement** is made between:
1. **[name]**, from now on referred to as **Provider** and
2. **[name]**, from now on referred to as the **Client**.

INTRODUCTION

3. The **Provider** is obliged to provide the **Internet Services** as stipulated in this **Agreement**.
5. DEFINITIONS
   5.1. j) **Internet traffic** may be measured by both **Client** and **Provider** by means of Equipment and may take the two values **high** and **normal**.

OPERATIVE PART

7. CLIENT'S RESPONSIBILITIES AND DUTIES
   7.1. The **Client** shall not:
   a) supply false information to the Client Relations Department of the **Provider**.
   7.2. Whenever the Internet Traffic is **high** then the **Client** must pay [*price*] immediately, or the **Client** must notify the **Provider** by sending an e-mail specifying that he will pay later.
   7.3. If the **Client** delays the payment as stipulated in 7.2, after notification he must immediately lower the Internet traffic to the **normal** level, and pay later twice $(2 * [price])$.
   7.4. If the **Client** does not lower the Internet traffic immediately, then the **Client** will have to pay $3 * [price]$.
   7.5. The **Client** shall, as soon as the Internet Service becomes operative, submit within seven (7) days the Personal Data Form from his account on the **Provider**'s web page to the Client Relations Department of the **Provider**.
8. CLIENT'S RIGHTS
   8.1. The **Client** may choose to pay either:
   a) each month; b) each three (3) months; c) each six (6) months;
9. PROVIDER'S SERVICE
   9.2. As part of the Service offered by the **Provider** the **Client** has the right to an e-mail and an user account.
   9.3. **Provider** is obliged to offer with no limitation and within a period of seven (7) days a password and any other Equipment Specific to Client, necessary for the correct usage of the user account, upon receiving of all the necessary data about the client from the Client Relations Department of the **Provider**.
   9.4. Each month the **Client** pays the *bill* the **Provider** is obliged to send a Report of Internet Usage to the Client.
10. PROVIDER'S DUTIES
   10.1. The **Provider** takes the obligation to return the personal data of the client to the original status upon termination of the present **Agreement**, and afterwards to delete and not use for any purpose any whole or part of it.
   10.2. The **Provider** guarantees that the Client Relations Department, as part of his administrative organisation, will be responsive to requests from the **Client** or any other Department of the **Provider**, or the **Provider** itself within a period less than two (2) hours during *working hours* or the day after.
11. PROVIDER'S RIGHTS
   11.1. The **Provider** takes the right to alter, delete, or use the *personal data* of the **Client** only for statistics, monitoring and internal usage in the confidence of the **Provider**.
   11.2. **Provider** may, at its sole discretion, without notice or giving any reason or incurring any liability for doing so:
   b) Suspend Internet Services immediately if **Client** is in breach of Clause 7.1;
13. TERMINATION
   13.1. Without limiting the generality of any other *Clause* in this *Agreement* the **Client** may terminate this *Agreement* immediately without any notice and being vindicated of any of the Clause of the present Agreement if:
   a) the **Provider** does not provide the Internet Service for seven (7) days consecutively.
   13.2. The **Provider** is forbidden to terminate the present Agreement without previous written notification by normal post and by e-mail.
   13.3. The **Provider** may terminate the present Agreement if:
   a) any payment due from **Client** to **Provider** pursuant to this **Agreement** remains unpaid for a period of fourteen (14) days;
16. GOVERNING LAW
   16.1. The **Provider** and the present **Agreement** are governed by and construed according to the Law Regulating Internet Services and to the Law of the State.
   a) The Law of the State stipulates that any **ISP Provider** is obliged, upon request to seize any activity until further notice from the State representatives.

Fig. 1. Part of a contract between an Internet provider and a client

We borrow from propositional dynamic logic [6] the syntax $[\alpha]\phi$ to represent that after performing α (if it is possible to do so), ϕ must hold. The $[\cdot]$ notation allows having a *test*, where $[\phi?]\mathcal{C}$ must be understood as $\phi \Rightarrow \mathcal{C}$. $\langle\alpha\rangle\phi$ captures the idea that it exists the possibility of executing α, in which case ϕ must hold afterwards. Following temporal logic (TL) notation we have $\mathcal{U}$ (*until*), $\bigcirc$ (*next*), and $\Box$ (*always*), with intuitive semantics as in TL [12]. Thus $\mathcal{C}_1 \mathcal{U} \mathcal{C}_2$ states that $\mathcal{C}_1$ holds until $\mathcal{C}_2$ holds. $\bigcirc\mathcal{C}$ intuitively states that $\mathcal{C}$ holds in the next moment, usually after something happens, and $\Box\mathcal{C}$ expressing that $\mathcal{C}$ holds in every moment. We can define $\Diamond\mathcal{C}$ (*eventually*) for expressing that $\mathcal{C}$ holds sometimes in a future moment.

To express CTDs we provide the following notation, $O_\varphi(\alpha)$, which is syntactic sugar for $O(\alpha) \wedge [\overline{\alpha}]\varphi$ stating the obligation to execute α, and the reparation φ in case the obligation is violated, i.e. whenever α is not performed. The reparation may be any contract clause. Similarly, CTP statements $F_\varphi(\alpha)$ can be defined as $F_\varphi(\alpha) = F(\alpha) \wedge [\alpha]\varphi$, where φ is the penalty in case the prohibition is violated. Notice that it is possible to express nested CTDs and CTPs.

In $\mathcal{CL}$ we can write *conditional* obligations, permissions and prohibitions in two different ways. Just as an example let us consider conditional obligations. The first kind is represented as $[\alpha]O(\beta)$, which may be read as "after performing α, one is obliged to do β". The second kind is modelled using the test operator ?: $[\varphi?]O(\alpha)$, representing "If φ holds then one is obliged to perform α". Similarly for permission and prohibition. For convenience, in what follows we use the notation $\phi \Rightarrow \mathcal{C}$ instead of the $\mathcal{CL}$ syntax $[\phi?]\mathcal{C}$.

3 A Contract Case Study

In what follows we consider part 7 of the contract given in Fig. 1 between a service provider and a client, where the provider gives access to the Internet to the client. We consider two parameters of the service: *high* and *normal*, which denote the client's Internet traffic. We will consider only the following clauses of the contract.

7.1. The **Client** shall not:
  a) supply false information to the Client Relations Department of the **Provider**.
7.2. Whenever the Internet Traffic is **high** then the **Client** must pay [*price*] immediately, or the **Client** must notify the **Provider** by sending an e-mail specifying that he will pay later.
7.3. If the **Client** delays the payment as stipulated in 7.2, after notification he must immediately lower the Internet traffic to the **normal** level, and pay later twice $(2 * [price])$.
7.4. If the **Client** does not lower the Internet traffic immediately, then the **Client** will have to pay $3 * [price]$.
7.5. The **Client** shall, as soon as the Internet Service becomes operative, submit within seven (7) days the Personal Data Form from his account on the **Provider**'s web page to the Client Relations Department of the **Provider**.

We also add clause 11.2 as it is strongly related to clause 7.1 and the two should be taken together:

11.2. **Provider** may, at its sole discretion, without notice or giving any reason or incurring any liability for doing so:
  b) Suspend Internet Services immediately if **Client** is in breach of Clause 7.1;

In what follows we formalise the above contract clauses. As part of the formalisation of a contract in $\mathcal{CL}$ we first have to define the assertions and actions:

ϕ = the Internet traffic is high
fi = client supplies false information to Client Relations Department
h = client increases Internet traffic to *high* level
p = client pays [price]
d = client delays payment
n = client notifies by e-mail
l = client lowers the Internet traffic
sfD = client sends the Personal Data Form to Client Relations Department
o = provider activates the Internet Service (it becomes operative)
s = provider suspends service

Note that we have the action h which does not appear explicitly in the example clauses. Action h is implicit as it makes the proposition ϕ valid (the Internet becomes *high* only if the client increases it). Action h can be considered as the complement of action l which makes ϕ false (lowers the Internet traffic). The six clauses above are written in $\mathcal{CL}$ as follows:

1. $\Box F_{P(s)}(fi)$
2. $\Box[h](\phi \Rightarrow O(p + (d\&n)))$
3. $\Box([d\&n](O(l) \wedge [l]\Diamond O(p\&p)))$
4. $\Box([d\&n \cdot \overline{l}\,]\Diamond O(p\&p\&p))$
5. $\Box([o]O(sfD))$

Clause 1 has a concise syntax and represents a *contrary-to-prohibition*. More precisely, the CTP represents the prohibition $F(fi)$ (clause 7.1) and the reparation which should be enforced in case the prohibition is violated (in this case $P(s)$; the right of the provider to suspend the Internet service, clause 11.2).

Note that all the clauses are supposed to hold throughout the whole contract because of the $\Box$. Clause 2 models clause 7.2 of the contract example and it represents the fact that whenever the assertion ϕ holds (the Internet traffic of the client is at the *high* level) then it must be the case that the client is obliged to choose $(+)$ between either paying immediately (p) or delaying the payment by sending the notification $(d\&n)$.

Clauses 3 and 4 refer to the clauses 7.3 and 7.4 of the contract example. They both refer to the moment after the client has delayed the payment $([d\&n])$. Clause 3 states that the client has the obligation to lower the Internet traffic $(O(l))$ and that after lowering the client should pay twice the price. On the other hand, clause 4 specifies the obligation of the client to pay three times the price in case he does not lower the Internet traffic $(\overline{l})$. The two formulae may be combined in a single formula using CTDs: $\Box([d\&n](O_{\varphi}(l) \wedge [l]\Diamond(O(p\&p))$ where $\varphi = O(p\&p\&p)$. Clause 5 formally represents clause 7.5 of the contract example. It represents the obligation of the client to submit the form $(O(sfD))$ after the Internet service becomes operative $([o])$.

Table 1. The translation function $f^{\mathcal{T}}$ from $\mathcal{CL}$ to $\mathcal{C}\mu$

$$
\begin{array}{lrcl}
(1) & f^{\mathcal{T}}(O(\&_{i=1}^{n} a_i)) & = & \langle\{a_1,\ldots,a_n\}\rangle(\wedge_{i=1}^{n} O_{a_i}) \\
(2) & f^{\mathcal{T}}(\mathcal{C}_O \oplus \mathcal{C}_O) & = & f^{\mathcal{T}}(\mathcal{C}_O) \wedge f^{\mathcal{T}}(\mathcal{C}_O) \\
(3) & f^{\mathcal{T}}(P(\&_{i=1}^{n} a_i)) & = & \langle\{a_1,\ldots,a_n\}\rangle(\wedge_{i=1}^{n} \neg\mathcal{F}_{a_i}) \\
(4) & f^{\mathcal{T}}(\mathcal{C}_P \oplus \mathcal{C}_P) & = & f^{\mathcal{T}}(\mathcal{C}_P) \wedge f^{\mathcal{T}}(\mathcal{C}_P) \\
(5) & f^{\mathcal{T}}(F(\&_{i=1}^{n} a_i)) & = & [\{a_1,\ldots,a_n\}](\wedge_{i=1}^{n} \mathcal{F}_{a_i}) \\
(6) & f^{\mathcal{T}}(F(\delta) \vee [\beta]F(\delta)) & = & f^{\mathcal{T}}(F(\delta)) \vee f^{\mathcal{T}}([\beta]F(\delta)) \\
(7) & f^{\mathcal{T}}(\mathcal{C}_1 \wedge \mathcal{C}_2) & = & f^{\mathcal{T}}(\mathcal{C}_1) \wedge f^{\mathcal{T}}(\mathcal{C}_2) \\
(8) & f^{\mathcal{T}}(\bigcirc\mathcal{C}) & = & [\mathbf{any}] f^{\mathcal{T}}(\mathcal{C}) \\
(9) & f^{\mathcal{T}}(\mathcal{C}_1 \,\mathcal{U}\, \mathcal{C}_2) & = & \mu Z.f^{\mathcal{T}}(\mathcal{C}_2) \vee (f^{\mathcal{T}}(\mathcal{C}_1) \wedge [\mathbf{any}]Z \wedge \langle\mathbf{any}\rangle\top) \\
(10) & f^{\mathcal{T}}(\Box\mathcal{C}) & = & \nu Z.\mathcal{C} \wedge [\mathbf{any}]Z \\
(11) & f^{\mathcal{T}}([\&_{i=1}^{n} a_i]\mathcal{C}) & = & [\{a_1,\ldots,a_n\}] f^{\mathcal{T}}(\mathcal{C}) \\
(12) & f^{\mathcal{T}}([(\&_{i=1}^{n} a_i)\alpha]\mathcal{C}) & = & [\{a_1,\ldots,a_n\}] f^{\mathcal{T}}([\alpha]\mathcal{C}) \\
(13) & f^{\mathcal{T}}([\alpha+\beta]\mathcal{C}) & = & f^{\mathcal{T}}([\alpha]\mathcal{C}) \wedge f^{\mathcal{T}}([\beta]\mathcal{C}) \\
(14) & f^{\mathcal{T}}([\varphi?]\mathcal{C}) & = & f^{\mathcal{T}}(\varphi) \Rightarrow f^{\mathcal{T}}(\mathcal{C})
\end{array}
$$

3.1 Translating the $\mathcal{CL}$ Specification into $\mathcal{C}\mu$

We extract a model from the $\mathcal{CL}$ clauses by first translating the language specification into the extended μ-calculus $\mathcal{C}\mu$ where the semantics is given as a special labelled transition system. The translation function $f^{\mathcal{T}}$ which takes a $\mathcal{CL}$ formula and returns a formula in the $\mathcal{C}\mu$ is shown in Table 1. The special syntax $[\mathbf{any}]$ (or the dual $\langle\mathbf{any}\rangle$) represents the fact that any action can be executed. To represent obligations and prohibitions of a given action a we need the special propositional constants O_a and $\mathcal{F}_a$.

We briefly mention here the semantics of $\mathcal{C}\mu$, see [13] for more details. The formulae are interpreted over a labelled transition system (LTS). The labels of the transitions are represented by multi-sets of actions (e.g. $\{p,p,p\}$ is a label corresponding to the $\mathcal{CL}$ concurrent action term $p\&p\&p$). The formulae are interpreted over states as usual in modal logics with semantics on LTSs. For example the expression $\phi \Rightarrow \langle p\rangle O_p$ is interpreted in a state and should be understood as: if the assertion ϕ holds in the state then $\langle p\rangle O_p$ should hold in the same state. $[p]\mathcal{C}$ and $\langle p\rangle\mathcal{C}$ are interpreted as holding in the current state if and only if in the next state reachable by action p the formula corresponding to the translation of $\mathcal{C}$ holds. In $\mathcal{C}\mu$ the difference between the two operators is that $\langle p\rangle\varphi$ requires the existence of at least one next state reachable by p where φ holds, where $[p]\varphi$ is quantified universally, and thus the formula also holds in case the set of states reachable by p is empty.

We will now translate the five $\mathcal{CL}$ clauses corresponding to the contract given above, into $\mathcal{C}\mu$. Note that we use the $\Box$ and $\Diamond$ with their classical interpretation from temporal logics; the last not being included in the Table 1. It is known [2] that $f^{\mathcal{T}}(\Diamond\mathcal{C}) = f^{\mathcal{T}}(\top\mathcal{U}\mathcal{C}) = \mu Z.\mathcal{C} \vee ([\mathbf{any}]Z \wedge \langle\mathbf{any}\rangle\top)$. In order to translate the first clause of the $\mathcal{CL}$ representation above we can proceed as follows:

$$
\begin{array}{rc}
 & f^{\mathcal{T}}(\Box F_{P(s)}(fi)) = \nu Z.f^{\mathcal{T}}(F_{P(s)}(fi)) \wedge [\mathbf{any}]Z, \\
\text{where:} & f^{\mathcal{T}}(F_{P(s)}(fi)) = f^{\mathcal{T}}(F(fi) \wedge [fi]P(s)) = [fi]\mathcal{F}_{fi} \wedge [fi]\langle s\rangle\neg F_s.
\end{array}
$$

In this manner, we use $\Box$ operator in the clauses below simply as syntactic sugar which is reduced to the ν operator in μ-calculus.

1. $\Box[fi]\mathcal{F}_{fi} \wedge [fi]\langle s\rangle\neg F_s$
2. $\Box[h](\phi \Rightarrow (\langle p\rangle O_p \wedge \langle\{d,n\}\rangle(O_d \wedge O_n)))$
3. $\Box[\{d,n\}](\langle l\rangle O_l \wedge [l](\mu Z.\langle\{p,p\}\rangle O_p \vee ([\mathbf{any}]Z \wedge \langle\mathbf{any}\rangle\top)))$
4. $\Box[\{d,n\}][\overline{l}](\mu Z.\langle\{p,p,p\}\rangle O_p \vee ([\mathbf{any}]Z \wedge \langle\mathbf{any}\rangle\top))$
5. $\Box[o]\langle sfD\rangle O_{sfD}$

3.2 From $\mathcal{C}\mu$ to the LTS

In Fig. 2 we have pictured one model of the above clauses where we denote by *else* all other actions different than the ones from the current node (e.g. for the state s_7 in the picture $else = \mathbf{any} \setminus \{fi\}$).

Note that because of the semantics of the prohibition $F(fi)$ (i.e., $[fi]\mathcal{F}_{fi}$), we would not need to explicitly add a transition from each state labelled with fi to a state with the propositional constant $\mathcal{F}_{fi}$. However, in the presence of a CTP, as it is the case with clause 1, we need to do so in order to represent the reparation $P(s)$.

We attempt to build a model in the form of an LTS — in a certain sense an implementation of the contract as specified. The process is done manually and prone to error — to ensure correctness of the automata we build, we model check them against the contract specification. Furthermore, multiple models satisfying the contract specification exist, ranging from the weakest being equivalent to the specification itself, to stronger and more concrete implementations. In this paper we are not concerned with achieving the weakest model.

Although the weakest model is desirable to have, we can still reason about our contract based on a (correct) model we build. Given a model $\mathcal{M}$ and contract specification $\mathcal{C}$, we start off by proving that the model really implements the contract: $\mathcal{M} \models \mathcal{C}$. We note that when the model does not satisfy a property π: $\mathcal{M} \not\models \pi$, it immediately follows that neither does the contract: $\mathcal{C} \not\models \pi$, thus enabling us to discover bugs in our specification as translated from the natural language, or in the original natural language contract itself. On the other hand, using this approach we cannot prove the correctness of the original contract. Were we able to obtain the weakest model, we would have been able to reason directly about the contract specification itself.

In what follows, we will specify this model using the input language of NuSMV, and prove that it is indeed a model of the $\mathcal{CL}$ formulae.

3.3 From the LTS to the NuSMV Input Syntax

In NuSMV [4], a model can be specified in two ways: either using *assignments* or by *direct specification*. We choose to use the direct specification technique as it enables us to translate our system more directly into NuSMV.

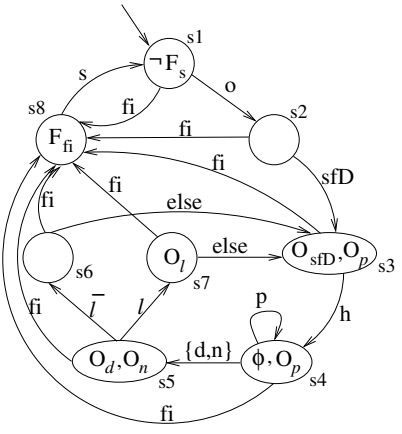

Fig. 2. Example of a model for the five clauses written in $\mathcal{CL}$

NuSMV uses *state variables* to identify states; the number of states is determined by the product of the number of different values each state variable can take. There is also a second kind of variables, *input variables* which are meant to specify labels of a labelled transition system. Since we have actions as labels, we make substantial use of the input variables in our application.

We have defined an input variable for each atomic action of the $\mathcal{CL}$ specification. The type of the input variables is `boolean` so that if the value of `d = false` then `d` is not an active label of the transition. Whenever a variable is left unspecified then NuSMV interprets it as having any value so it creates a transition (or a state in case of state variables) for each value of the variable.

In NuSMV it is easy to simulate the concurrent labels $\{d, n\}$ of $\mathcal{C}\mu$ which mean that the transition is taken if both actions d and n are executed concurrently: we activate both input variables `d = true ; n = true`. We can also represent the resource-awareness of the labels (i.e. the $p\&p$ of $\mathcal{CL}$, or the $\{p, p\}$ of $\mathcal{C}\mu$) by defining the input variable with the type *range of integers*. If `p = 0` then the transition is not labelled with the action p; if `p = 1` then the transition is labelled with one normal action p (like in the case of `boolean` type); but if `p = 2` then we take the transition if two copies of the action p are executed concurrently. We have then the following declaration of variables:

```
IVAR
  d : boolean ;
  n : boolean ;
  p : 0 .. 3 ;
```

Note that we may have *empty transitions* (with no label) by giving to all the input variables the value `false` (or `p = 0`). Moreover, we may represent the special action **any** of $\mathcal{C}\mu$ by leaving all input variables unspecified.

We have defined a state variable named `state` of enumeration type so it can take only eight values, corresponding to the eight states depicted in Fig. 2.

```
VAR
  state : {s1,s2,s3,s4,s5,s6,s7,s8} ;
```

Other variables are declared accordingly (e.g., `high : boolean`). Moreover, we define a state variable of type `boolean` for each input variable. This is required by the $\mathcal{C}\mu$ where we have a propositional constant O_a or $\mathcal{F}_a$ associated to each atomic action a which enters under the scope of an obligation or of a prohibition respectively:

```
  F_s : boolean ;  F_fi : boolean ;
  O_p : boolean ;  O_d : boolean ;  O_n : boolean ;
  O_l : boolean ;  O_sfD : boolean ;
```

As an example, we show below the encoding of the initial state, and one of its outgoing transitions, of the automaton in Fig. 2. We call the initial state `s1`.

```
INIT
  (state = s1) & !high &
  !F_fi & !O_p & !O_d & !O_n & !O_l & !O_sfD & !F_s ;
```

The transitions are specified using the `TRANS` keyword followed by a propositional formula which determines the pairs of states that form the transition relation. The propositional formula contains names of state variable (which are tested in the current state) and `next` expressions which refer to the value of the state variables in the next state. It also contains the input variables to model the labels of the transitions. Remember that any variable that is missing from the formula is interpreted as having any value and will give rise to a number of different transitions equal to the number of values it can take.

```
TRANS
--state variables of the current state
  ((state = s1) & !high &
   !F_fi & !O_p & !O_d & !O_n & !O_l & !O_sfD & !F_s &
--input variables as the labels
   (!fi & p = 0 & !d & !n & !l & !negl & !sfD & o & !s) &
--the values of the state variables in the next states
   (next(state) = s6) & !next(high) &
   next(!F_fi & !O_p & !O_d & !O_n & !O_l & !O_sfD & !F_s))
```

3.4 Model Checking the Contract

We propose to combine the contract specification and the model we build in different ways with model checking techniques to help us improve the contract and increase our confidence in our model.

Proving that the model satisfies the original clauses: Clearly, to have confidence that we are reasoning using a correct model, we need to prove that the automaton

of Fig. 2, specified in NuSMV[1] respects the five $\mathcal{CL}$ clauses representing the statements from the contract example. For this we have specified each clause as a special LTL specification in NuSMV:

```
G ((fi -> X F_fi) & (fi -> X (s & X !F_s)))
G (h -> X (high -> ((p = 1 -> X O_p) &
                    ((d & n) -> X (O_d & O_n)))))
G ((d & n) -> X ((l -> X O_l) & l -> X F (p = 2 -> X O_p)))
G ( (d & n) -> X (l -> X F (p = 3 -> X O_p)))
G (o -> X (sfD -> X O_sfD))
```

The first, second and fourth properties go through immediately. The third fails, but upon investigation, it turns out that the actual contract wording gave a dependency between the second and third properties — the $d\&n$ action in the third property only refers to ones produced in the context of the second property (just after the Internet traffic going high and the user paying once). This indicates that the two ought to be combined together either by adding extra logic to indicate the dependency, or by merging then into a single property. We choose the latter, obtaining:

```
G (h -> X (high -> ((p = 1 -> X O_p) &  ((d & n) ->
                  X (O_d & O_n & (l -> X O_l) &
                     l -> X F (p = 2 -> X O_p))))))
```

This new property can be verified of our model.

Finally, the fifth property fails, suggesting that our model is incorrect. However, upon inspection it was realised that nothing in the contract specifies that the activation of the service happens once, or that the user's obligation is only valid the first time the activation occurs. We choose to revise the original contract to state that: "The first time the service becomes operative, the client is obliged to send the Personal Data Form to Client Relations Department". This is formulated as the following property, which model checks:

```
(!o) U (o -> X(can_sfD & (sfD -> X O_sfD)))
```

An alternative solution is to ensure that the contract is only in force once the Internet Service becomes operative, and simplify the property accordingly.

Verifying a property about client obligations: The first desirable property we want to check on the contract model can be expressed in English as: "It is always the case that whenever the Internet traffic is high, if the clients pays immediately, then the client is *not* obliged to pay again immediately afterwards". The property is expressed in $\mathcal{CL}$-like syntax[2] as: $\Box\neg(\phi \Rightarrow [p]\neg O(p))$. The property proves to be false, as can be seen in the transcript below, which includes a counter-example:

[1] The NuSMV code we have used is available on Nordunet3 project homepage: `http://www.ifi.uio.no/~gerardo/nordunet3/software.shtml`

[2] Notice that formally in $\mathcal{CL}$ there is no negation at the clause level.

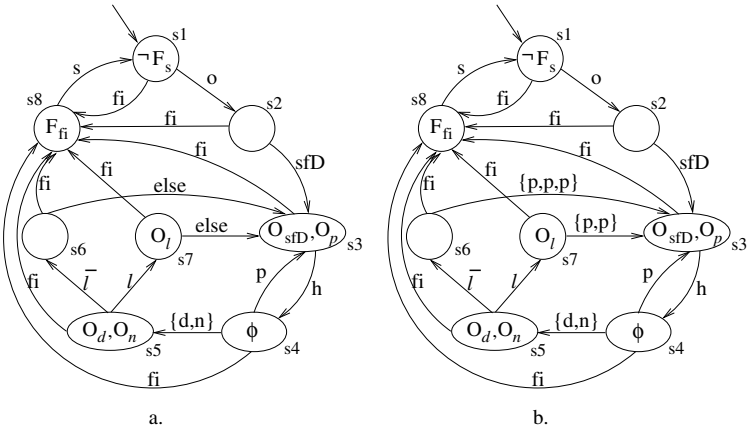

Fig. 3. The model of Fig. 2 corrected.

```
NuSMV > check_ltlspec
-- specification
     G (!high | (p = 1 ->  X (p = 1 ->  X !O_p))) is false
-- as demonstrated by the following execution sequence
-> State: 2.1 <-
  state = s1; o = 1
-> State: 2.2 <-
  state = s2; sfD = 1
-> State: 2.3 <-
  state = s3; O_p = 1; O_sfD = 1; h = 1
-- Loop starts here
-> State: 2.4 <-
  state = s4; high = 1; O_sfD = 0; p = 1
-- Loop starts here
-> State: 2.5 <-
  p = 1
```

The above counter-example shows that in state `s4` of Fig. 2 the client must fulfil one of the following obligations: or to pay (`p`), or to delay payment and notify (`d,n`). However, after paying once, the automaton is still in a state with high traffic (state `s4`), and thus the client is still obliged to pay again.

We give in Fig. 3-a the new model, which is proved correct with respect to the above property. The difference is the transition $s_4 \xrightarrow{p} s_3$ which replaces the one labelled with p from s_4 to itself. From this it is easy now to modify the original contract by introducing the following clause: "The provider guarantees that if the Internet traffic of the Client reaches a high level and the Client pays the [price] then it will not be obliged to pay the [price] again".

Notice that though we have obtained a new model that satisfies the property (and a clause in the original contract solving the above problem), the solution is still not satisfactory, as the contract does not specify what happens after the

client pays but does not decrease the Internet traffic. In the new model shown in Fig. 3-a this is reflected by the fact that after taking the new added transition (from `s4` to `s3`), there is an implicit assumption that the Internet traffic is low. For brevity we do not further analyse the contract in order to obtain the right contract concerning this problem, though it can be done following a similar approach as above.

Verifying a property about payment in case of increasing Internet traffic: The checking of the previous property was done for the benefit of the client. We now perform model checking in order to increase the confidence of the provider of the service.

We are interested in proving that: "It is always the case that whenever Internet traffic is high, if the client delays payment and notifies, and afterwards lowers the Internet traffic, then the client is forbidden to increase Internet traffic unless she/he pays twice". This complicated English clause is specified in $\mathcal{CL}$-like syntax as: $\Box(\phi \Rightarrow [d\&n \cdot l](F(h)\ \mathcal{U}\ \text{done}_{p\&p}))$.

Here $\text{done}_{p\&p}$ is an assertion added to specify that the client has paid twice. Notice that in order to prove the property we need to extend the NuSMV model of the contract with a propositional constant corresponding to $\text{done}_{p\&p}$ which is true only after a transition labelled $\{p, p\}$ is taken.

In Fig. 3-a we show the control structure of the LTS. The additional state variable $\text{done}_{p\&p}$ is added to the NuSMV model, thus effectively introducing two states for every one in Fig. 3-a, with different values for the state variable.

The original property proves to be false, since from state s_4 (where ϕ holds), after $d\&n \cdot l$, it is possible to increase Internet traffic in state s_7 (due to the $else$ label), so neither $F(h)$ nor $\text{done}_{p\&p}$ hold.

Though it was not apparent at first sight, and confirmed by the result given by the tool, the above clause allow the client to go from normal to high Internet traffic many times and pay the penalty $(2 * [price])$ only once. The problem is that after the client lowers the Internet traffic, he might get a high traffic again and postpone the payment till a future moment. This problem comes from the ambiguity of the language. Note that the $\mathcal{CL}$ formalisation in the clauses 3 and 4 use the $\Diamond$ to model the fact that a statement will hold eventually in the future but not necessarily *immediately* (expressions "pay later" in clause 7.3 and "will have to pay" in clause 7.4 are the ambiguities). The *eventually* was translated with the help of the special syntax $else$ that we see in Fig. 3-a. We use the counter-example given by NuSMV to construct the model in Fig. 3-b where the property holds. The difference is at the transition from s_7 to s_3 where we have changed the label to the multi-set label $\{p, p\}$. In $\mathcal{CL}$ the solution is to add a new clause corresponding to the property above, and the original contract should be extended with the English version of the property as expressed above. Note that a similar property can be stated for the clause 4 for which we have given the solution in Fig. 3-b also by replacing the label of the transition from s_6 to s_3 by the multi-set label $\{p, p, p\}$.

4 Final Discussion

In this paper we have shown how model checking techniques and tools can be applied to analyse contracts. In particular, we have used NuSMV [4] to model check conventional contracts specified using the language $\mathcal{CL}$. In this paper, we presented multiple uses of model checking for reasoning about contracts. Firstly, we use model checking to increase our confidence in the correctness of the model with respect to the original natural language contract. Secondly, by finding errors in the model, we can identify problems with the original natural language contract or its interpretation into $\mathcal{CL}$. Finally, we enable the signatories to safeguard their interests by ensuring certain desirable properties (and lack of undesirable ones).

About NuSMV: NuSMV [4] is the successor of the milestone symbolic model checker SMV [10]. Symbolic model checking [3] is based on the clever encoding of the states using binary decision diagrams or related techniques, but still relies on the classical model checking algorithm. NuSMV allows the checking of properties specified in CTL, LTL, or PSL. More recently NuSMV has included *input variables* with which it is possible to specify directly a labelled transition system. This feature of NuSMV has been very useful in our context.

Related Work: To our knowledge, model checking contracts is quite an unexplored area where only few works can be found [15,5]. The main difference with our approach is that in [15] there is no language for writing contracts, instead automata are used to model the different participants of a contract, i.e. there is no model of the contract itself but only of the behaviour of the contract signatories. Many safety and liveness properties identified as common to e-contracts are then verified in a purchaser/supplier case study using SPIN [7]. Similarly, in [5] Petri nets are used to model the behaviour of the participants of a contractual protocol. Though in [15] it is claimed that modelling the signatories gives modularity, adding clauses to a given contract implies modifying the automata. In our case, adding clauses to a contract is done as in any declarative language, without changing the rest. Though in our current implementation we would also need to rewrite the verification model, this should not be seen as a disadvantage; given that $\mathcal{CL}$ has formal semantics in $\mathcal{C}\mu$ the model could be obtained automatically after the modifications. An advantage of our approach is the possibility of explicitly writing conditional obligations, permissions and prohibitions, as well as CTDs and CTPs. We are not aware of any other work on model checking e-contracts along the same line as ours. See [13] and [15] (and references therein) for further discussions, and other approaches, on formalisations of contracts.

Future Work: The approach we have followed has few drawbacks. First notice that the way we have obtained the model for the least fix-point in the $\mathcal{C}\mu$ formula 3 in Section 3.1 was modelled as the cycle $(s_7, s_3, s_4, s_5)^*$, which may indeed be an infinite loop as we do not have accepting conditions in our labelled Kripke structure nor fairness constraints. This of course would need to be refined in

order to guarantee that the cycle will eventually finish. Moreover, in order to be able to prove properties about actions which must have been performed, we should extend our language with a constructor $done(\cdot)$ to be applied to actions, meaning that the action argument was performed (as with the $\text{done}_{p\&p}$ in the example). This will definitely facilitate specifying properties like the last one of the previous section concerning the prohibition on actions by the client. We are currently working on improving the above aspects in order to make a more precise analysis.

We have presented a manual translation from the $\mathcal{C}\mu$ semantics of the contract written in $\mathcal{CL}$ into the input language of NuSMV. We plan to implement a tool to automatically model check contracts written in $\mathcal{CL}$. We can benefit from the counter-example generation to fix the original contract, as we have briefly shown in Section 3.4. The underlying model checker of such tool could be NuSMV or another existing μ-calculus model checker (e.g., [1,9]).

With such a tool the whole model checking process will be accelerated facilitating its use and thus making it easy to prove other interesting general properties about e-contracts, as suggested in [15]. Besides such classical liveness or safety properties we are also interested in properties more specific to e-contracts, such as: finding the obligations or prohibitions of one of the parties in the contract; listing of all the rights that follow after the fulfilling of an obligation; what are the penalties for whenever violating an obligation or prohibition; determining whether a given participant is obliged to perform contradictory actions.

The generation of the (automata-like) model that we did by hand in Section 3 can be done automatically along the lines of existing LTL-to-Büchi automata translators (like `ltl2smv` or `ltl2ba`). [14] presents a comprehensive overview of the state-of-the-art of such tools.

In the current state of development, the language $\mathcal{CL}$ cannot explicitly express timing constraints. We intend to extend the language with such features in order to be able to specify and verify real-time properties.

Acknowledgements. We would like to thank Martin Steffen for suggestions on an early draft of this paper.

References

1. Biere, A.: mu-cke - efficient mu-calculus model checking. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 468–471. Springer, Heidelberg (1997)
2. Bradfield, J., Stirling, C.: Modal Logics and Mu-Calculi: an Introduction, pp. 293–330. Elsevier, Amsterdam (2001)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: LICS 1990, pp. 428–439 (1990)
4. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
5. Daskalopulu, A.: Model checking contractual protocols. In: JURIX 2000, Frontiers in Artificial Intelligence and Applications Series, pp. 35–47 (2000)

6. Fischer, M.J., Ladner, R.E.: Propositional modal logic of programs. In: STOC 1977, pp. 286–294 (1977)
7. Holzmann, G.: The Spin Model Checker, Primer and Reference Manual. Addison-Wesley, Reading (2003)
8. Kozen, D.: Results on the propositional mu-calculus. Theor. Comput. Sci. 27, 333–354 (1983)
9. Mateescu, R., Sighireanu, M.: Efficient on-the-fly model-checking for regular alternation-free mu-calculus. Sci. Comput. Program. 46, 255–281 (2003)
10. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
11. McNamara, P.: Deontic logic. In: Handbook of the History of Logic, vol. 7, pp. 197–289. North-Holland Publishing, Amsterdam (2006)
12. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57 (1977)
13. Prisacariu, C., Schneider, G.: A formal language for electronic contracts. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 174–189. IFIP (2007)
14. Rozier, K.Y., Vardi, M.Y.: Ltl satisfiability checking. In: Bosnacki, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 182–200 (2007)
15. Solaiman, E., Molina-Jiménez, C., Shrivastava, S.K.: Model checking correctness properties of electronic contracts. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 303–318. Springer, Heidelberg (2003)