Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 253 (2009) 39–55

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

On the Specification of Full Contracts[1]

Stephen Fenech, Gordon J. Pace[2]

Dept. of Computer Science and AI
University of Malta, Malta

Department of Computer Science
Aalborg University, Denmark

Gerardo Schneider[4]

Department of Informatics
University of Oslo, Norway

Abstract

Contracts specify properties of an interface to a software component. We consider the problem of defining a full contract that specifies not only the normal behaviour, but also special cases and tolerated exceptions. In this paper we focus on the behavioural properties of use cases taken from the Common Component Modelling Example (CoCoME), proposed as a benchmark to compare different component models. We first give the full specification of the use cases in the deontic-based specification language $\mathcal{CL}$, and then we concentrate on three particular properties in order to compare deontic and operational specifications. We conjecture that operational specifications are well suited for normal cases, but are less easily extended for exceptional cases. This hypothesis is investigated by comparing specifications in CSP (operational) with specifications in $\mathcal{CL}$. The outcome of the experiment supports the conjecture and demonstrates clear differences in the basic descriptive power of the formalisms.

Keywords: Contracts, CoCoME, deontic specifications, operational specifications

1 Introduction

Modern software applications are built from components that are connected either statically or dynamically, for instance using a service oriented architecture for Internet-based applications. Components are developed by different teams that

[1] Partially supported by the Nordunet3 project COSoDIS.

[2] Email:sfen002,gordon.pace@um.edu.mt

[3] Email:ojc,apr@cs.aau.dk

[4] Email:gerardo@ifi.uio.no

doi:10.1016/j.entcs.2009.09.027

may be distributed across countries and organisations. With this reality, it becomes important that the interfaces and protocols used between components are well specified, that there are some *contracts* that regulate these issues. Here the concepts and techniques developed in the formal methods community attract attention. One example is contracts as functional specifications in terms of invariants, pre- and postconditions which are predicates over state variables and parameters that define an input, pre-state, output, post-state relation. The behaviour of components, i.e. the acceptable sequences of method calls or signals that can be exchanged among components, is also important to understand the overall result of connecting distributed, concurrently executing components for an application, as it is done in a service oriented architecture. Here, operational specifications are quite popular; they include both automata based approaches and language oriented process algebras. Deontic logics have not been used to the same extent, although they would offer greater potentials for abstraction from the actual implementations and give a constraint oriented specification style. To some extent this is understandable, because logic formulae are more abstract and not so easy to understand as models. However, they may have an advantage when it comes to providing a full specification of a contract which includes not only the normal use cases, but also special cases with compensations, tolerance of deviations or faults, or exception handling. Here operational models quickly become complex, because they have to specify the compensations and alternatives by branching to different paths.

In this paper, we start by giving the specification in $\mathcal{CL}$, a deontic-based formal language for contracts [12], of a large case study which was developed to compare different formal approaches for the specification and analysis of a component based system of a realistic complexity — the CoCoME (*The Common Component Modelling Example*) experiment [13]. This case study involves all the usual aspects of functionality and behaviour; but also aspects like performance, timing constraints and even dependability. We then investigate contract specifications using logic ($\mathcal{CL}$) and operational models (CSP [5]) by looking at a fragment of CoCoMe; and we also contrast these with specifications in LTL and CTL [11]. Since we want to examine the particular hypothesis about operational versus logic specifications, we limit ourselves to behaviours, where the distinction will come out. We have furthermore isolated a particular component where interaction with humans and external organisations come to the surface. This is where handling exceptions and exceptional cases becomes important to capture the total behaviour so as to avoid unexpected cases. For example, let us consider part of the informal specification of a supermarket cash desk: "While in express mode (allowing only clients with less than 8 items), if no sale is currently taking place, the cashier can choose to disable the express mode". From the behavioural point of view a sequence of events consisting of clients with more than 8 items coming into an express cashier and the subsequent payment, seems to be acceptable given that the cashier can make an exception. Any specification language whose semantics would accept such a sequence would in principle be considered a suitable formalism. This is, however, only partially correct, since it will depend on which kind of properties we are interested in. Just the sequence of events does not keep the original informal specification which uses expressions like “can choose”. This kind of modalities add extra information which may be lost by simply observing the sequence of events.

The contributions of this paper are twofold. First, we formalise the specification of the behavioural aspect of CoCoMe in $\mathcal{CL}$. Second, we take 2 use cases from CoCoME to compare deontic ($\mathcal{CL}$) and operational (CSP) specifications.

The paper is organised as follows. In next section we provide a general description of CoCoME. In section 3 we present the language $\mathcal{CL}$ and we give the CoCoME specification. In section 4 we present in detail the three properties to be specified in section 5 using $\mathcal{CL}$ and CSP, and we briefly comment on the suitability of LTL and CTL as specification languages in this context. We compare the specifications in section 6, to conclude in the last section.

2 CoCoME

The Common Component Modelling Example (CoCoME) [13] is based on a Trading System that handles the sales and inventory of a Store chain. The case study is defined using 8 use cases that describe the main processes. The use cases span from selling products at a cash desk to the exchange of product between stores. The use cases are described as a sequence of actions that must occur followed by a list of exceptional behaviour if the use case allows such behaviour.

Use Case 1 describes how a sale is processed, from the scanning of the items to the payment, either by cash or card. In the exceptional situation that a card validation fails, the cashier should retry the validation process or require that the customer pays in cash.

Use Case 2 describes how a cash desk switches to express mode which restricts the total number of items the customer should have.

Use Case 3 describes how products, which are running low, are ordered.

Use Case 4 describes how to receive these orders once the suppliers have delivered the items. In the exceptional situation where the delivery is not correct or complete, the products are sent back to the supplier.

Use Case 5 describes how the system generates stock-related reports.

Use Case 6 describe how the system generates delivery reports.

Use Case 7 describes how the price of a product may be altered.

Use Case 8 describes how products can be exchanged from one store to another when the product is running low in one of the stores. The store running low on a certain product will inform the enterprise server, which will send an update stock request to all ‘nearby’ stores. With the fresh stock information the enterprise server will decide on which store should exchange the goods and sends the request to send the goods. In the exceptional situation that the enterprise server is unreachable, the request is queued to be retried later. In the exceptional situation that not all the ‘nearby’ stores reply to the update stock request the enterprise server will wait for 15 min after which it will continue the process assuming that stores that have not responded to the request do not have the required products.

3 Specification of CoCoME using $\mathcal{CL}$ —Use Cases 3-8

In this section we first present $\mathcal{CL}$ [12], a language to express contracts as terms over obligations, permissions and prohibitions, and then we show how to specify CoCoME in $\mathcal{CL}$. $\mathcal{CL}$ has the following syntax:

$$\begin{aligned}
\mathcal{C} &:= \mathcal{C}_O | \mathcal{C}_P | \mathcal{C}_F | \mathcal{C} \wedge \mathcal{C} | [\beta]\mathcal{C} | \langle\beta\rangle\mathcal{C} | \top | \bot \\
\mathcal{C}_O &:= O_C(\alpha) | \mathcal{C}_O \oplus \mathcal{C}_O \\
\mathcal{C}_P &:= P(\alpha) | \mathcal{C}_P \oplus \mathcal{C}_P \\
\mathcal{C}_F &:= F_C(\delta) | \mathcal{C}_F \vee [\alpha]\mathcal{C}_F \\
\alpha &:= 0|1|a|\alpha\&\alpha|\alpha \cdot \alpha|\alpha + \alpha \\
\beta &:= 0|1|a|\beta\&\beta|\beta \cdot \beta|\beta + \beta|\beta^*
\end{aligned}$$

This syntax is an extension of that given in [8] where here we add the angle brackets. The semantics of $\mathcal{CL}$ have been given in an extension of μ-calculus, an intuitive explanation of which is given below.

A contract typically consists of two parts: *definitions* ($\mathcal{D}$) and *clauses* ($\mathcal{C}$). We deliberately leave the definitions part underspecified in the syntax above. $\mathcal{D}$ specifies the *assertions* (or conditions) and the atomic actions present in the clauses. In this case, the vocabulary of Table 1. Atomic actions are underspecified, but consist of (at least) three parts: the proper action, the subject performing the action, and the target of (or, the object receiving) the action. Note that, in this way, the parties involved in a contract are directly 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 $\oplus$ correspond to the classical conjunction and exclusive disjunction, which may be used to combine obligations and permissions. For prohibition clauses $\mathcal{C}_F$, the operator $\vee$ corresponding to disjunction is used. The constraints on which operators may be used to compose which types of clauses are introduced to avoid expressing paradoxical contracts.

The α is a compound action (i.e., an expression containing one or more of the following operators: choice "+"; sequence "·", and concurrency "&" — see [8]), while β is a compound action which can also be made up of the Kleene star "*". Note that $\oplus$ cannot appear between prohibitions and + cannot occur under the scope of F.

$\mathcal{CL}$ borrows from propositional dynamic logic [3] the syntax $[\alpha]\mathcal{C}$ to represent that after performing α (if it is possible to do so), $\mathcal{C}$ must be satisfied. $\langle\alpha\rangle\mathcal{C}$ captures the idea that the possibility exists of executing α, in which case $\mathcal{C}$ must hold afterwards.

$\mathcal{CL}$ can be extended with the temporal operators $\Diamond$ (*eventually*) and $\Box$ (*always*), with standard semantics [11]. Thus $\Box\mathcal{C}$ can be defined as $[1^*]C$. Similarly, we can define $\Diamond\mathcal{C}$ (*eventually*) for expressing that $\mathcal{C}$ holds sometime in a future moment, as well as the $\mathcal{U}$ (*until*) and $\bigcirc$ (*next*) operators.

Contrary-to-duty (CTD) contracts, which specify an obligation and reparation contract in case the obligation is not met, is expressed in $\mathcal{CL}$ as $O_C(\alpha)$: obliging action α, but defaulting to contract C if it not satisfied. Similarly, contrary-to-prohibition (CTP) contracts, specifying a prohibited action α and its reparation clause C in case of violation, can also be expressed: $F_C(\alpha)$.

In what follows we specify CoCoME using the contract language $\mathcal{CL}$. CoCoME specifies both behavioural and functional requirements. $\mathcal{CL}$ does not yet support the specification of timing constraints natively; however, one could encode these constraints in the definition of the actions. We have only done this in cases where the timing constraint affected the behaviour of the system since we are focusing on the behavioural specification. Though $\mathcal{CL}$ is limited when it comes to timing constraints, it will allow us to describe exceptional behaviour easily and concisely.

In this section we will specify use cases three to eight of the CoCoME case study. In the following section we will focus on the most interesting parts of use cases one and two and use them to compare deontic specification with operational specification. In the rest of the paper we will use the action names shown in Table 1. For a more detailed presentation of the $\mathcal{CL}$ specification presented in what follows, refer to [15].

Specification of Use Case 3 (Order Products)

(i) $\Box[\text{startOrderProcess}]O(\text{listItems}\&\text{listLowItems})$

(ii) $\Box[\text{listItems}\&\text{listLowItems}]P(\text{entersAmount})$

(iii) $\Box[\text{entersAmount}]P(\text{mngOrderButton})$

(iv) $\Box[\text{mngOrderButton}]O(\text{placeOrder}\&\text{displayOrderID})$

Once the manager starts the order products process (startOrderProcess) the system is obliged to show the full list of items and the list of items that are running low (listItems&listLowItems). After this the manager has the permission to enter the amount of items he would like to order (entersAmount) after which he is permitted to press the order button (mngOrderButton) in which case the system is obliged to place the order and display the order id (placeOrder&displayOrderID). This use case does not have any exceptional behaviour specified. Furthermore, the distinction between the system *permitting* the manager to do certain actions (e.g. P(entersAmount)) and the system being *obliged* to respond (e.g. O(placeOrder&displayOrderID)) is not explicitly described in the CoCoME specification but rather assumed from the common expectations.

Specification of Use Case 4 (Receive Ordered Products)

(i) $\Box[\text{deliver}]O_{O(\text{sendBack})}(\text{completeCorrect})$

(ii) $\Box[\text{completeCorrect}]O(\text{mngOrderButton})$

(iii) $\Box[\text{mngOrderButton}]O(\text{updateInventory})$

disableExpress	Go to Normal Mode
enableExpress	Go to Express Mode
conditionMet	Condition to go to express mode has been met
startSale	Start a new sale
enterItem	Enter new item
finishSale	Stop entering items and start payment procedure
cashPay	Pay in Cash
cardPay	Pay with Card
correctPin	Pin entered is correct
incorrectPin	Pin entered is incorrect
sendBack	Send customer to another line
> 8	Customer has more than eight items
< 8	Customer has less than eight items
returnItems	Customer forfeits items
startOrderProcess	Manager initiates the start of the Order Products process
listItems	The System lists all the products
listLowItems	The system lists the products which are running out of stock
entersAmount	The store manager chooses the items to order and enters the corresponding amount
mngOrderButton	The store Manager presses the Order button
placeOrder	The System places the order to the appropriate supplier
displayOrderID	The system displays the order identifier generated to the Store Manager
deliver	Supplier delivers the ordered stock which is identified by an order ID
completeCorrect	Supplier made a complete and correct delivery. This is checked by the Stock Manager
orderReceived	Manager receives the order by pressing the button Roll in received order
updateInventory	The System updates the inventory
sendBack	The Stock Manager sends the products back to the supplier
enterStoreID	Manager enters the store identifier and presses the button Create Report
displayReport	System displaces a report including all the available stock items in the store.
enterEnterpriseID	Manager enters the enterprise identifier and presses the button Create Report
displayEnterpriseReport	The System generates and displays an Enterprise report
requestOverview	The Manager requests a listing of available products in the store
listItems	The System lists all the products
selectItem	The Manager Selects an Item
changePrice	The Manager changes price
pressCommit	The Manager commits by pressing enter
commitPriceChange	The System changes the price according to the amount set by the manager
productRunsOut	A product of a store runs out
lowStock	The store server recognises low stock of the product.
productRequest	The Store Server sends a request to the Enterprise Server
inventoryRequest	The enterprise server sends an Inventory request to nearby stores
inventoryReply	The store replies with the inventory information
inventoryUpdate	The enterprise server updates the database and looks up the product
storeChosen	The enterprise server using an “optimisation criterion” to find a store
productReply	The enterprise server sends a message to the receiving store.
transferRequest	The enterprise server sends a message to the transferring store
queueRequest	Store server queues request to enterprise.
15min	15 minutes have passed
allRequestsReceived	All requests have been received

Table 1
Alphabet

The case study describes that that Manager is required to check that the supplier has sent the correct and complete order. Instead of defining an action MgrChecksOrder we defined the action completeCorrect since the obligation is on the supplier to send the correct information. Thus here we have that once the delivery is made (deliver) the supplier is obliged to have sent the complete and correct delivery (completeCorrect). If however the supplier has violated this obligation, the manager is obliged to send the order back (sendBack), otherwise he is obliged to process the order (mngOrderButton) and the system is obliged to update accordingly (updateInventory).

Specification of Use Case 5 (Show Stock Report)

(i) $\Box[\text{enterStoreID}]O(\text{displayReport})$

Once the manager enters the store id (enterStoreID) the system is obliged to display the report (displayReport).

Specification of Use Case 6 (Show Delivery Report)

(i) $\Box[\text{enterStoreID}]O(\text{displayReport})$

Once the enterprise manager enters the store id (enterStoreID) the system is obliged to display the report (displayReport).

Specification of Use Case 7 (Change Price)

(i) $\Box[\text{requestOverview}]O(\text{listItems})$

(ii) $\Box[\text{listItems}]P(\text{selectItem})$

(iii) $\Box[\text{selectItem}]P(\text{changePrice})$

(iv) $\Box[\text{changePrice}]P(\text{pressCommit})$

(v) $\Box[\text{pressCommit}]O(\text{commitPriceChange})$

This use case shows the process of how a manager may change a price of an item. The manager starts this process by requesting a list of available products (requestOverview). The system is obliged to list all the items (listItems) and give permission to the manager to choose items (selectItem). If the manager does select an item, the system should give permission to the manager to change the price (changePrice) after which it should give permission for the manager to commit the price change (pressCommit). If the manager commits the changes, the system is obliged to make these changes permanent (commitPriceChange).

Specification of Use Case 8 (Product Exchange Among Stores)

(i) $\Box[\text{productRunsOut}]O(\text{lowStock})$

(ii) $\Box[\text{lowStock}]O_{O(\text{queueRequest})}(\text{productRequest})$

(iii) $\Box[\text{productRequest}]O(\text{inventoryRequest})$

(iv) $\Box[\text{inventoryRequest}]O(\text{inventoryReply})$

(v) $\Box[\text{inventoryReply}]O(\text{inventoryUpdate})$

(vi) $\Box[15\text{min} + \text{allRequestsReceived}]O(\text{storeChosen})$

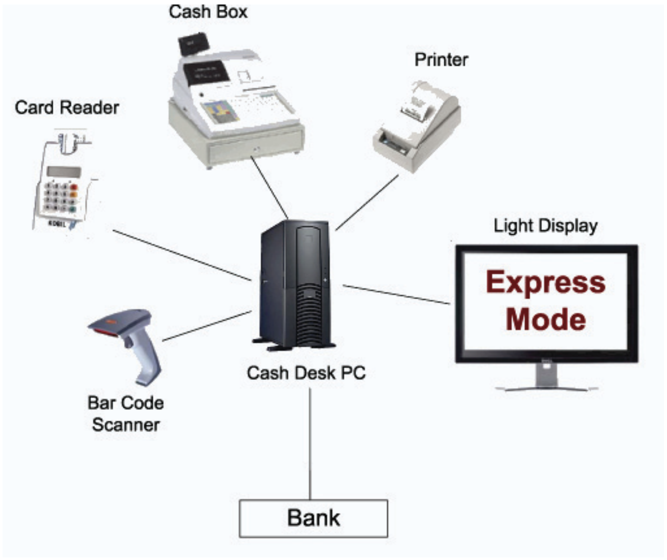

Fig. 1. Cash desk and its constituents

(vii) $\Box$[storeChosen]O(productReply&transferRequest)

If a product runs out (productRunsOut) the local store server should recognise that this has occurred (lowStock) and is obliged to send a request to the enterprise server(productRequest). If this is not successful (for example the connection is down) then the request should be queued (queueRequest). Once such a request is received by the enterprise server, it is obliged to send an inventory request to all nearby stores (inventoryRequest). Every store that receives this request is obliged to reply with the inventory information (inventoryReply). After every reply the enterprise server updates the local databases (inventoryUpdate). Once the enterprise server receives all the replies from the stores or 15 minutes have passed since the requests were sent (15min + allRequestsReceived) it chooses from where the items should be taken and sends a reply to the original store requesting the items and a message to the store that is going to supply the items (productReply&transferRequest).

4 An Example of a Full Contract –Use Cases 1-2

We shall concentrate on the cash desk part of the example shown in Fig. 1 which have the following constituents:

(1) Each cash desk has a Cash Box for starting and finishing a sale, and entering received money. (2) In order to identify the products to sell, each cash desk is equipped with a Bar Code Scanner. (3) A Card Reader is installed at each cash desk for handling card payment. Paying by cash can be handled by the Cash Box. (4) In addition there is a Printer for printing the bill which is handed out to the customer at the end of the sale process. (5) To realise the express checkout mentioned above, each cash desk is equipped with a Light Display which signals the customers if the Cash Desk is currently operating in an express mode. If so, the

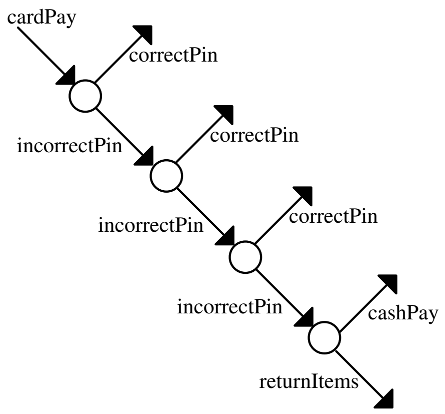

Fig. 2. Full transition diagram for cardPay (F1)

customers are only allowed to buy a small amount of goods and must pay cash in order to keep each transaction short. (6) Each Cash Desk has its own Cash Desk PC where the software handles the sale process, and takes care of the communication with the Bank. Furthermore, it integrates all devices at the Cash Desk.

We focus on the behavioural aspect of the use case, and in particular the following 3 clauses of the contract which includes expected and exceptional behaviours, fairness, permissions and obligations:

F1 If the customer chooses to pay by cash he is obliged to swipe the card followed by entering the correct pin number. If the pin number is incorrect the customer has two more attempts at entering the correct pin after which the client is obliged to pay with cash. If the client refrains to pay with cash the client has to give up the goods. See transition diagram in Fig. 2.

F2 While in normal mode, the cashier may choose to switch to express mode if in the last hour 50% of the sales had less than eight items (conditionMet). Once in express mode the cashier is obliged to eventually go back to normal mode. If conditionMet holds infinitely often, then the cashier should change to express mode infinitely often. See transition diagram in Fig. 3.

F3 In express mode, once a sale has commenced, the cashier is obliged to service customers with less than eight items. To service a customer, the items need to be entered in the system, and then finish the sale. If a customer has more than eight items then it is up to the cashier's discretion whether to service the client or send him back to the end of another line. See Fig. 4.

Clause F2 includes interesting aspects as permissions, obligations and fairness constraints. In Fig. 3 the leftmost state decorated with a black circle indicates that the state should be visited infinitely often. This models the part of the clause which states that the cashier is obliged to always eventually go back to normal mode. From the normal state we can only exit when the express condition is met, after which the cashier has the choice of going back to normal mode or express mode. The dashed transition signifies that if this transition is taken infinitely often then the dotted transition needs to be also taken infinitely often, modelling the part of

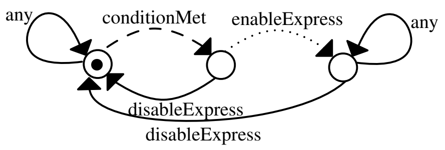

Fig. 3. Transition diagram for Express mode (F2)

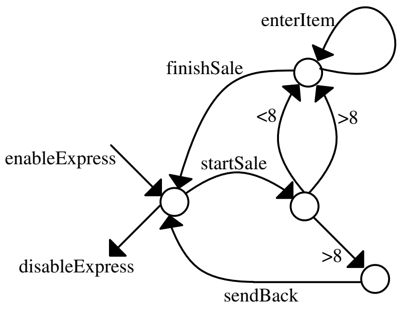

Fig. 4. Transition diagram for sales process (F3)

the clause stating that if the condition is met infinitely often then the cash desk needs to infinitely often go into express mode.

In clause F3 the choice to serve a client with more than 8 items is up to the cashier's judgement, This 'permission' to the cashier to 'violate' the rule can be seen as an allowed explicit exception.

5 Formal Specifications of Use Cases 1-2

Our first formal specification is operational, using CSP; it includes the normal operations for the three clauses. Then follows specifications using temporal logics, and finally the deontic logic based specifications. We use the action names shown in Table 1.

5.1 Operational Specification

The Relational Calculus of Object and Component Systems (rCOS) is a method for developing component based systems. Syntactically, it is rooted in Unified Theory of Programming (UTP) [6] which has been adapted for object and component based use [4]. Behavioural aspects are syntactically expressed by UML diagrams. Semantically and for verification purposes, they are translated to CSP [5].

CSP terms define processes:

$$P \ ::= \ Stop \mid a \rightarrow P \mid P[]P \mid P \sqcap P \mid X$$

where $Stop$ denotes the deadlocked process; action prefix $a \rightarrow P$ means do a then act as P; external choice ($[]$) between processes, whichever is able to proceed is executed; non-deterministic or internal choice ($\sqcap$), one is chosen; and finally X denotes a process name for a process defined in a set of mutually recursive definitions: $X = P$.

$$\begin{array}{lcl}
CashDesk & = & \quad disableExpress \to NormalDesk \\
 & & [\,]\ enableExpress \to ExpressDesk \\
ExpressDesk & = & startSale \to EnterExp(0) \\
NormalDesk & = & startSale \to EnterNormal \\
EnterExp(i) & = & i < 8 \quad \wedge \\
 & & enterItem \to EnterExp(i+1) \\
 & & [\,]\ finishSale \to cashPay \to CashDesk \\
EnterNormal & = & \quad enterItem \to EnterNormal \\
 & & [\,]\ finishSale \to Finish \\
Finish & = & \quad cashPay \to CashDesk \\
 & & [\,]\ cardPay \to CashDesk
\end{array}$$

Table 2
Normal case specification

The trace semantics of CSP defines a set of finite traces. For the refusal semantics, which distinguishes the two choice operators, refer to [5,14].

The Normal Case Specification

The example scenario of sale processing which forms the basis for the example contract is rendered as the CSP processes shown in Table 2. In this specification, we use a bounded integer counter i which ranges from 0 to 8; thus the specification stays within the fragment that can be analysed with a model checker.

Specification of F1

Here we need to modify the $Finish$ process only:

$$\begin{array}{lcl}
Finish & = & \quad cashPay \to CashDesk \ \ [\,]\ \ cardPay \to Card \\
Card & = & sendPin \to Check(0) \\
Check(i) & = & \quad correctPin \to CashDesk \\
 & & [\,]\ \ i \geq 3 \wedge incorrectPin \to Nocard \\
 & & [\,]\ \ i < 3 \wedge incorrectPin \to Check(i+1) \\
Nocard & = & \quad cashPay \to CashDesk \ \ [\,]\ \ returnItems \to CashDesk
\end{array}$$

This can be proved to be a refinement of the $Finish$ process in the normal behaviour; but note the intricate branching logic.

Specification of F2

Concerning F2, a non-deterministic switching could be added. It can be specified as follows:

$$Switch \ = \ (enableExpress \to Switch) \ \sqcap \ (disableExpress \to Switch)$$

However, there is no guarantee of fairness or liveness, so it is left underspecified.

Specification of F3

Here we have to modify the process $EnterExp$:

$$\begin{array}{rcl} \mathit{EnterExp}(i) & = & (i < 8 \rightarrow \mathit{enterItem} \rightarrow \mathit{EnterExp}(i+1) \\ & & \;\; [] \; \mathit{finishSale} \rightarrow \mathit{CashDesk}) \\ & & [] \;\; i \geq 8 \rightarrow \mathit{Finalise}(i) \\ \mathit{Finalise}(i) & = & (\mathit{finishSale} \rightarrow \mathit{cashPay} \rightarrow \mathit{CashDesk} \\ & & \;\; [] \;\; \mathit{enterItem} \rightarrow \mathit{EnterExp}(i+1)) \\ & & \sqcap \; \mathit{finishSale} \rightarrow \mathit{CashDesk} \end{array}$$

where *Finalise* gives the non-deterministic choice of the cashier. Note, however, that in this case we get a process that is no longer a refinement of the previous defined one because it allows same behaviours that were prohibited before.

5.2 Temporal Logics Specification

Two widely used temporal logics are LTL and CTL. LTL is a linear temporal logic which allows us to specify properties over paths. Given a set P of atomic prepositions, the syntax of an LTL formula is

$$\varphi \;::=\; p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

The LTL formula $\mathbf{G}\varphi$ means that φ always hold, $\mathbf{F}\varphi$ that φ will eventually hold, $\mathbf{X}\varphi$ that φ will hold in the next step and $\varphi\mathbf{U}\psi$ that φ holds until ψ holds.

CTL is a branching time temporal logic which makes use of the same LTL temporal operators but each temporal operator is preceded by a path quantifier, either $\mathbf{E}$ or $\mathbf{A}$:

$$\varphi \;::=\; p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{AG}\varphi \mid \mathbf{AF}\varphi \mid \mathbf{AX}\varphi \mid \varphi\mathbf{AU}\varphi \mid \mathbf{EG}\varphi \mid \mathbf{EF}\varphi \mid \mathbf{EX}\varphi \mid \varphi\mathbf{EU}\varphi$$

$\mathbf{E}$ is the existential path quantifier meaning that there exists at least one path starting from this state, which satisfies the quantified formula. $\mathbf{A}$ is the universal path quantifier meaning that all the paths starting from this state must satisfy the quantified formula.

Specification of F1

The first clause can be seen as a list of conditional statements where it is always the case that after the card is swiped then there is a choice of either entering the correct pin, in which case it would satisfy the formula or else it could be satisfied in the next step. In the next step we repeat the possibility of satisfying the formula by entering the correct pin and if not we again check the next step. This formula can be described in both CTL and LTL:

$$\begin{array}{l} \mathbf{AG}(\text{cardPay} \rightarrow \;\mathbf{AX}\;(\text{correctPin} \;\vee\; \mathbf{AX}(\text{correctPin} \vee \\ \qquad\qquad\qquad\qquad \mathbf{AX}(\text{correctPin} \;\vee\; \mathbf{AX}(\text{cashPay} \;\vee\; \text{returnItems}))))) \end{array}$$

$$\begin{array}{l} \mathbf{G}(\text{cardPay} \rightarrow \;\mathbf{X}(\text{correctPin} \;\vee\; \mathbf{X}(\text{correctPin} \vee \\ \qquad\qquad\qquad\qquad \mathbf{X}(\text{correctPin} \;\vee\; \mathbf{X}(\text{cashPay} \vee \text{returnItems}))))) \end{array}$$

Specification of F2

The second clause cannot be described using CTL due to the fairness, unless the logic is extended with fairness constraints. Moreover, it is not clear how the permissions and obligations of the clause could faithfully be represented in CTL. Fairness is expressible using LTL, however, the clause also requires the existence of the transition leading to express mode which cannot be represented using LTL.

Specification of F3

For the third clause it is always the case that once we go to express mode then we need to satisfy the express mode behaviour until we go back to normal mode. Once a sale is started the client needs to be serviced until the sale is finished or the client is sent to another line. If the client has less than eight items then that implies that he should be serviced, otherwise the cashier has to choose between either servicing the customer or sending the customer back. We are also ensuring that there exists the possibility of both servicing the customer and sending the customer back since this is required by the clause. It is because of this requirement that the behaviour cannot be expressed using LTL. However, in CTL it is:

$$\begin{array}{l}
\mathbf{AG}\ \text{enableExpress} \rightarrow\ \mathbf{AX}(\text{startSale} \rightarrow \\
\quad \mathbf{AX}((< 8 \rightarrow \mathbf{AX}(\text{enterItem}\mathbf{AU}\text{finishSale})) \wedge \\
\qquad (> 8 \rightarrow\ \mathbf{AX}(\text{enterItem}\ \vee\ \text{sendBack})\ \wedge\ \mathbf{EX}(\text{enterItem})\ \wedge\ \mathbf{EX}(\text{sendBack}) \wedge \\
\qquad\ \ \mathbf{AX}(\text{enterItem} \rightarrow \text{enterItem}\mathbf{AU}\text{finishSale}))) \\
\quad \mathbf{AU}\ \text{disableExpress})
\end{array}$$

5.3 Deontic Specification

In this section we will present a deontic specification of the properties, using $\mathcal{CL}$.

Specification of F1

Here we make extensive use of nested CTDs, where we have a number of options of how the client may satisfy the payment by card. Once a card is swiped then the client is obliged to enter the correct pin (primary obligation). However, if the pin entered is incorrect then the client may still try again two times (secondary obligation) and in case of failure the exceptional cases of paying by cash or returning the items must be enforced. If none is satisfied, the contract is violated:

$$\square[\text{cardPay}]\ \ O_{\psi_1}(\mathit{correctPin})$$

where $\psi_1 = O_{\psi_2}(\mathit{correctPin})$, with $\psi_2 = O_{O(\mathit{cashPay}+\mathit{returnItems})}(\mathit{correctPin})$.

Specification of F2

Clause F2 starts by stating that the cashier is infinitely often obliged to go to the normal mode: it can never stay in express mode forever. Then we state that it is always the case that after conditionMet is observed (possibility to enable the express mode) then the cashier is obliged to either choose to stay in normal mode or express:

$$\begin{aligned}
&\Box\Diamond O(\text{disableExpress}) \wedge \\
&\Box([\text{conditionMet}]\ (O(\mathit{disableExpress} + \mathit{enableExpress})\ \wedge\ P(\mathit{enableExpress})\ ))\wedge \\
&\Box\Diamond[\text{conditionMet}]\ \Box\ \Diamond O(\mathit{enableExpress})
\end{aligned}$$

We also enforce that once the condition is met, the cashier has the possibility to go to express mode to avoid a model that only contains a return to normal mode. We do not need to explicitly ensure that there is a possibility to choose to stay in normal mode, similarly to what we have done with the express mode, or that after being in express mode we have the possibility to go back to normal mode because of the first conjunct which states that we have to go infinitely often to normal mode. The fairness requirement is specified in the final part of the clause where we say that if we infinitely often observe conditionMet, then we will infinitely often be obliged to go back into express mode.

Specification of F3

It is always the case that once we go to the express mode a certain behaviour needs to be followed until we go back to normal mode. In the case that the client has less than eight items, then the cashier is obliged to service the customer. However, if the client has more than eight items the cashier is obliged to choose to either service the customer or send back the customer to another cash desk and both possibilities should exist. The last property is thus specified in $\mathcal{CL}$ as follows:

$$\begin{aligned}
&\Box(\ [\mathit{enableExpress}](\ [\mathit{startSale}](\\
&\qquad [<8]O(\mathit{enterItem})\ \mathcal{U}\ \text{finishSale} \wedge \\
&\qquad [>8](O(\mathit{enterItem} + \mathit{sendBack})\ \wedge\ P(\mathit{sendBack})\ \wedge\ P(\mathit{enterItem}) \wedge \\
&\qquad\quad [\mathit{enterItem}]O(\mathit{enterItem})\ \mathcal{U}\ \text{finishSale}\ )) \\
&\quad \mathcal{U}\ \text{disableExpress}\ )
\end{aligned}$$

6 Comparison

In Table 3 we present a summary of which formulae can be expressed by the formalisms we used in the previous section. We elaborate in what follows on the differences between the approaches.

The specification of the example using the different notations shows that CTL and CSP allow the specification of exceptional behaviour aspect of a contract which cannot be specified in other notations such as LTL. Thus making it possible to specify full contracts. However, model based formalisms cannot express global properties such as fairness or liveness of a transition system, because they essentially model the individual transitions.

$\mathcal{CL}$ combines both linear and branching time, with the addition of certain deontic notions. It has not only information of what actions are to be done to satisfy the $\mathcal{CL}$ clause but also prescriptive information about the action, namely whenever the action is observable it is possible to distinguish whether it was required to perform it (as a primary obligation), whether it was a reparation to an obligation, or simply a permitted action.

Moreover, the expression of CTDs and CTPs in terms of basic $\mathcal{CL}$ goes beyond syntactic rewriting, since it still enables a contractual view of when obligations, permissions and prohibitions are active, have been satisfied, or violated. The main advantage of viewing the properties as a deontic contract is that this knowledge is preserved and can be reasoned about.

In summary, F2 seems to be relatively complex property difficult to be captured in specifications using temporal logics and operational approaches. Deontic specifications seem to be appropriate, whenever a right combination of deontic operators with temporal ones is provided.

Analysis

Though our aim is to compare the specification style of temporal logics, operational and deontic specifications, we are also interested in what we can do with those specifications, namely how easy it is to analyse them. It is well known that both LTL and CTL are amenable to model checking [1,7]. In the case of CSP suffices, so one can take advantage of the existing tool FDR2 [9] to do the analysis. It may be used to check CSP refinement as well as other properties such as deadlock freeness, trace refinement, etc. However, it is unclear what refinement should be checked for $F3$ since it contains contrary-to-duty actions, which do not blend well with ordinary refinement.

In what concerns $\mathcal{CL}$, an *ad-hoc* algorithm for checking deontic inconsistencies has recently been developed. In this way, given a $\mathcal{CL}$ contract, we are able to detect whether the contract contains contradictory obligations, or an obligation and a prohibition to do something at the same time, and other kinds of contradictions (see [2] for more details). A general model checker for $\mathcal{CL}$ is currently under development, though by using a semantic encoding into an extended μ-calculus [12] it is possible to model check contracts written in $\mathcal{CL}$ as presented in [10].

As an example in addition to the three clauses seen in section 4, let us consider the contract $[a]O(c) \wedge [b]F(c)$ which is satisfiable except when the concurrent action $a\&b$ is observed: we end in a state where the contract cannot be satisfied since c is both forbidden and required to happen. We could encode the $\mathcal{CL}$ trace semantics into LTL, however, the correct encoding of the deontic notions as to be able to model check contract inconsistencies would be extremely difficult. Moreover, in order to handle the above small example, CTL and LTL should be extended with concurrent actions, and a priority order among actions (this is built-in in $\mathcal{CL}$ [12]).

Summarising, once the specifications are written in any of the approaches under consideration, one can apply existing tools to further analyse them. However, only $\mathcal{CL}$ can be model checked against properties concerning obligations, permissions, and prohibitions, as well as CTDs and CTPs.

7 Final Remarks

In this paper we have given a specification of the CoCoME benchmark case study using a deontic specification language. We have then presented and examined the use of three specification styles for the description of total contracts, contracts which not only specify normal behaviours, but also exceptional ones. Clauses of the

	LTL	CTL	CSP	$\mathcal{CL}$
F1	✓	✓	✓	✓
F2	–	–	–	✓
F3	–	✓	(✓)	✓

Table 3
Comparisons between specifications

CoCoME example have been used to illustrate different types of contract clauses and how they can be handled using different specification approaches in order to identify their respective strengths and weaknesses.

One prevailing view of contracts is that of properties which the underlying system must satisfy. In the gist of this view, we have shown how they can be expressed in terms of appropriate standard logics, CTL and LTL. One main disadvantage of this approach is that obligations, permissions and prohibitions are encoded in terms of the underlying logic, making it difficult, in some cases practically impossible, to relate behaviour of the system back to these operators. The encoding also leads to loss of compositionality of contracts for exception handling or contract violations, as in the case of CTDs. Reasoning about CTDS and CTPs would be difficult. In particular, the detection of deontic inconsistencies, as explained at the end of the previous section, cannot be done in temporal logics, and quite difficult in many operational models.

Using a process calculus approach to describe contracts enables reasoning about the contracts in a direct manner — for instance comparing contracts up to a simulation relation. Also, more complex composition of contracts can be encoded in a direct manner. On the other hand, one still lacks information about contract violation and satisfaction which would have to be encoded directly (and thus prone to error), making the description of total contracts less direct.

Finally, we explore the use of a deontic logic based language to describe the contract clauses. In this approach, we note that reasoning about the deontic state of the system is possible. Moreover, the possibility to analyse contracts, and to express properties of contracts (such as "Whenever you are obliged to pay, you are forbidden from leaving the store, unless you pay") which may refer to the deontic state of the system, is highly desirable. Furthermore, only the analysis of deontic specification is suitable to detect inconsistencies concerning obligations, permissions and prohibitions in full contracts. An implementation of the inconsistency checker for $\mathcal{CL}$ is described in [2].

Overall, it can be argued that the appropriate specification language depends on the intended use. If the contract is intended to be used simply as a property which should be satisfied by a system, then the use of a standard logic, with adequate expressiveness and tool support, will usually suffice. If the use also includes the composition and comparison of contracts, the process calculus approach gives more flexibility. If it is required to analyse and compose full contracts including exceptional behaviour, a deontic approach would be more appropriate.

References

[1] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* The MIT Press, 1999.

[2] S. Fenech, G. J. Pace, and G. Schneider. Conflict analysis of deontic contracts (extended abstract). In *NWPT'08*, pages 34–36, November 2008.

[3] M. J. Fischer and R. E. Ladner. Propositional modal logic of programs. In *STOC'77*, pages 286–294. ACM, 1977.

[4] J. He, X. Li, and Z. Liu. rCOS: A refinement calculus for object systems. *Theoretical Computer Science*, 365(1-2):109–142, 2006.

[5] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[6] C. Hoare and J. He. *Unifying Theories of Programming.* Prentice-Hall, 1998.

[7] G. J. Holzmann. The model checker spin. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[8] M. Kyas, C. Prisacariu, and G. Schneider. Run-time monitoring of electronic contracts. In *ATVA'08*, LNCS. Springer-Verlag, October 2008. To appear.

[9] F. S. E. Ltd. http://www.fsel.com/software.html.

[10] G. Pace, C. Prisacariu, and G. Schneider. Model Checking Contracts –a case study. In *ATVA'07*, volume 4762 of *LNCS*, pages 82–97. Springer, October 2007.

[11] A. Pnueli. Temporal logic of programs, the. In *FOCS'77*, pages 46–57. IEEE Computer Society, 1977.

[12] C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.

[13] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, editors. *The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]*, volume 5153 of *LNCS*. Springer, 2008.

[14] A. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[15] Stephen Fenech. Full $\mathcal{CL}$ specification of CoCoME. Available from `http://www.cs.um.edu.mt/svrg/Tools/CLTool/Papers/CoCoMEfullCLSpec.pdf`.