A Process-Algebraic Approach to Workflow Specification and Refinement

Peter Y.H. Wong and Jeremy Gibbons

Oxford University Computing Laboratory, United Kingdom
{peter.wong,jeremy.gibbons}@comlab.ox.ac.uk

Abstract. This paper describes a process-algebraic approach to specification and refinement of workflow processes. In particular, we model both specification and implementation of workflows as CSP processes. CSP's behavioural models and their respective refinement relations not only enable us to prove correctness properties of an individual workflow process against its behavioural specification but also allows us to design and develop workflow processes compositionally. Moreover, coupled with CSP is an industrial strength automated model checker FDR, which allows behavioural properties of workflow models to be proved automatically. This paper details some CSP models of van der Aalst et al.'s control flow workflow patterns, and illustrates behavioural specification and refinement of workflow systems with a business process scenario.

1 Introduction

Since van der Aalst published his short note [18] comparing Petri nets and π-calculus with respect to his workflow patterns [19], some attempts have been made to express these patterns in π-calculus [12, 13] and its variants [11, 15]. In this paper, we demonstrate how the process algebra CSP also can be applied to model complex workflow systems; more importantly, we can exploit CSP's notion of *process refinement* [6, 14] to specify and compare these workflow systems. Furthermore, CSP is supported by the automated model checker FDR [4], which has been used extensively in industrial applications [10, 2]. The combination of the mathematics of refinement and the model checker is crucial in the development process of workflow systems, especially when designers do not want to be concerned with the underlying mathematics. To complement the work described in this paper, we have given a formal semantics for BPMN in CSP [23], allowing workflow process designers to construct specifications using BPMN, and to formally compare BPMN diagrams.

We first detail CSP models of some of van der Aalst et al.'s control flow workflow patterns [19], which serve as "jigsaw" pieces for workflow construction. We then present a case study of a business process to illustrate the composition of some of the workflow pattern models and the use of process refinement in specification and verification.

The rest of this paper is structured as follows. Section 2 gives a brief introduction to CSP, its syntax and semantics. Section 3 describes the CSP models of

workflow patterns. Section 4 details the business process case study. Sections 5 and 6 discuss the related work and directions for future work respectively.

2 Communicating Sequential Processes

In CSP [6], a process is defined as a pattern of possible behaviour; a behaviour consists of events which are atomic and synchronous between the environment and the process. The environment in this case can be another process. Events can be compound, constructed using '.' the dot operator; often these compound events behave as channels communicating data objects synchronously between the process and the environment. For example $a.b$ is a compound event which communicates object b through channel a. Below is the grammar of a simplied version of CSP in BNF.

$$\begin{aligned} P, Q ::= \quad & P \mid\mid\mid Q \quad | \quad P \,[\![A]\!]\, Q \quad | \quad P \setminus A \quad | \quad P \triangle Q \quad | \\ & P \,\Box\, Q \quad | \quad P \sqcap Q \quad | \quad P \fatsemi Q \quad | \quad e \rightarrow P \quad | \quad \mathit{Skip} \quad | \quad \mathit{Stop} \\ e ::= \quad & x \quad | \quad x.e \end{aligned}$$

Process $P \mid\mid\mid Q$ denotes the interleaved parallel composition of processes P and Q. Process $P \,[\![A]\!]\, Q$ denotes the partial interleaving of processes P and Q sharing events in set A. Process $P \setminus A$ is obtained by hiding all occurrences of events in set A from the environment of P. Process $P \triangle Q$ denotes a process initially behaving as P, but which may be interrupted by Q. Process $P \,\Box\, Q$ denotes the external choice between processes P and Q; the process is ready to behave as either P or Q. Process $P \sqcap Q$ denotes the internal choice between processes P or Q, ready to behave at least one of P and Q but not necessarily offer either of them. Process $P \fatsemi Q$ denotes a process ready to behave as P; after P has successfully terminated, the process is ready to behave as Q. Our syntactic notation for process sequential composition follows Davies' style [3]. Process $e \rightarrow P$ denotes a process capable of performing event e, after which it will behave like process P. The process *Stop* is a deadlocked process and the process *Skip* is a successful termination. We write $\Box\, a : \{ x_0 \ldots x_i \} \bullet P(a)$ to denote external choice over the set of processes $\{ P(x_0) \ldots P(x_i) \}$ and similarly for CSP operators $\sqcap$ and $\mid\mid\mid$.

CSP has three denotational semantic models: traces ($\mathcal{T}$), stable failures ($\mathcal{F}$) and failures-divergences ($\mathcal{N}$) models, in order of increasing precision. In this paper our process definitions are divergence-free, so we will concentrate on the stable failures model. The traces model is insufficient because it does not record the availability of events and hence only models what a process *can* do and nothing about what it *must* do [14]. Notable is the semantic equivalence of processes $P \,\Box\, Q$ and $P \sqcap Q$ under the traces model. Their trace semantics are defined below where $\mathcal{T}[\![\cdot]\!]$ is a semantic function which maps a CSP expression to its set of possible traces $\mathbb{P}(\mathrm{seq}\, \Sigma)$ and Σ is the set of all possible events.

$$\mathcal{T}[\![P \,\Box\, Q]\!] = \mathcal{T}[\![P \sqcap Q]\!] = \mathcal{T}[\![P]\!] \cup \mathcal{T}[\![Q]\!]$$

In order to distinguish these processes, it is necessary to record not only what a process can do, but also what it can *refuse* to do. This information is preserved in *refusal sets*, sets of events from which a process in a stable state can refuse to communicate anything no matter how long it is offered. A (stable) *failure* is a pair in which the first element is the trace of a process and the second is a refusal set of the process after the given trace. Below is the stable failures semantics of both choice operators where $\mathcal{F}[\cdot]$ is a semantic function that maps a CSP expression to its set of failures $\mathbb{P}(\text{seq}\,\Sigma \times \mathbb{P}\,\Sigma)$.

$$\begin{aligned}\mathcal{F}[P \,\Box\, Q] &= \{ref : \mathbb{P}\,\Sigma \mid (\langle\rangle, ref) \in \mathcal{F}[P] \cap \mathcal{F}[Q] \bullet (\langle\rangle, ref)\} \\ &\quad \cup \{tr : \text{seq}\,\Sigma;\ ref : \mathbb{P}\,\Sigma \mid tr \neq \langle\rangle \wedge (tr, ref) \in \mathcal{F}[P] \cup \mathcal{F}[Q] \bullet (tr, ref)\} \\ \mathcal{F}[P \sqcap Q] &= \mathcal{F}[P] \cup \mathcal{F}[Q]\end{aligned}$$

Each CSP process hence is characterised by its pattern of behaviour; the type of specification we are concerned with is termed *behavioural specification*. In CSP's behavioural models ($\mathcal{T}$,$\mathcal{F}$ and $\mathcal{N}$) a specification R is expressed by constructing the most non-deterministic process satisfying it, called the *characteristic process* P_R. Any process Q that satisfies specification R has to refine P_R; this is denoted by $P_R \sqsubseteq Q$. In the stable failures model, we say process Q *failure-refines* process P if and only if every failure of Q is also a failure of P.

$$P \sqsubseteq_F Q \Leftrightarrow \mathcal{F}[Q] \subseteq \mathcal{F}[P]$$

Similarly, we say P is *failure-equivalent* to Q if and only if they have the same set of failures.

$$P \equiv_F Q \Leftrightarrow P \sqsubseteq_F Q \wedge Q \sqsubseteq_F P \Leftrightarrow \mathcal{F}[P] = \mathcal{F}[Q]$$

While traces only carries information about *safety* conditions, refinement under the stable failures model allows one to make assertions about a system's *safety* and *availability* properties. These assertions can be automatically proved using CSP's model checker FDR [4]. FDR stands for "Failures-Divergence Refinement"; Model checkers exhaustively explore the state space of a system, either returning one or more counterexamples to a stated property or guaranteeing that no counterexample exists. FDR is among the most powerful explicit exhaustive finite-state exploration tools and has been used extensively in industrial applications [10, 2].

3 Patterns

Van der Aalst et al. introduced workflow patterns as the "gold standard" benchmark of workflow languages [19]. These patterns range from simple constructs to complex routing primitives. Their scope is limited to static control flow.

We model each of these control flow patterns in CSP, adhering to the interpretation of a process instance given by WfMC Reference Model [7] and van der Aalst et al.'s description of workflow activity [19]. We define set $\mathcal{A}$ as the

set of workflow activities, and define set of compound events $\{ n : \mathcal{A} \bullet init.n \}$ as the set of events representing workflow triggers and $\{ n : \mathcal{A} \bullet work.n \}$ as the set of events representing the execution of workflow activities. We can then define the CSP process $P(a, X)$ where a,b,c,... range over $\mathcal{A}$. This process models basic workflow activity a. We use X, Y,... to range over $\mathbb{P}\,\mathcal{A}$.

$$P(a, X) = init.a \rightarrow work.a \rightarrow \Big|\Big|\Big| b : X \bullet init.b \rightarrow Skip$$

The process description $P(a, X)$ first performs the event $init.a$ with the co-operation of the environment. This event represents an external trigger to the start of the activity a; after the trigger has occurred, the event $work.a$, which represents some activity a, will be ready to perform. After $work.a$ has occurred, the process is ready to perform the set of events $\{ b : X \mid init.b \}$ which trigger a set of workflow activities $X \subseteq \mathcal{A}$. A workflow activity which only triggers one subsequent activity can hence be defined.

$$SP(a, b) = P(a, \{b\})$$

Each CSP description of the workflow pattern represents an abstracted view of a workflow process. In this paper we only concern ourselves with the modelling of flow of control between activities and external to them. Each CSP process Q modelling some workflow activities hence has a corresponding process Q' which has the execution of its workflow activities internalised via the hiding operation.

$$Q' = Q \setminus \{\!|work|\!\}$$

The hiding operation reflects independent execution of individual workflow activities and allows us to model workflow processes with different levels of abstraction. As these workflow models are refined, more implementation details about individual activities might be added such as their internal data flow information. We use the event *init.acts* to denote a general trigger for the workflow activity *acts* which is outside the scope of the workflow process in which *init.acts* occurs. It represents the completion of the relevant activities defined with the process. We use the event *init.null* to denote a general trigger to some workflow activity *null* that is outside the scope of the workflow process in which *init.null* occurs and *null* is ignored.

The rest of this section is devoted to a detailed description of the CSP model of these patterns based on the definitions above and the semantics of CSP. Due to page restriction, we have only included in this paper the CSP model of the workflow patterns which are relevant in the subsequent case study. A complete presentation of the CSP models of workflow patterns can be sought elsewhere [22].

3.1 Basic Control Flow Patterns

In this section the workflow patterns capture the basic control flows of workflow activities. They form the basis of more advanced patterns.

Sequence - An activity b in the workflow process is triggered after the completion of the activity a. This pattern is modelled by the CSP process $SEQ(S)$ where S is a non empty sequence of activities to be executed sequentially. For example, in the example given, S would be $\langle a, b\rangle$. (The symbol $\frown$ denotes sequence concatenation.)

$$\begin{aligned} SEQ(\langle\rangle) &= Skip \\ SEQ(\langle s\rangle) &= SP(x, acts) \\ SEQ(\langle s, t\rangle \frown S) &= SP(s, t) \,[\![\, \{init.t\} \,]\!]\, SEQ(\langle t\rangle \frown S) \end{aligned}$$

Parallel Split (AND-split) - Both activities b and c are triggered after the completion of the activity a. The execution of b and c is concurrent. This pattern can be modelled by the CSP process $ASP(a, X)$ where a is some activity; set $X \subseteq \mathcal{A}$ is a non empty set of activities to be triggered in parallel after a has completed; in the example given, X would be $\{b, c\}$.

$$ASP(a, X) = P(a, X) \,[\![\, \{\, k : X \bullet init.k \,\} \,]\!]\, |||\, k : X \bullet SP(k, acts)$$

Synchronisation (AND-join) - An activity a is triggered after *both* activities b and c have completed execution, The execution of b and c is concurrent. This pattern may be modelled by the CSP process $AJP(X, y)$ where $X \subseteq \mathcal{A}$ is a non empty set of concurrent activities. In the example given, X would be $\{b, c\}$.

$$AJP(X, a) = |||\, k : X \bullet SP(k, a) \,[\![\, \{init.a\} \,]\!]\, SP(a, acts)$$

Exclusive Choice (XOR-split) - Either activities b or c is triggered after the completion of the activity a. The choice between b and c is internally (demonically) nondeterministic since such decision is part of the implementation detail. This pattern is modelled by the CSP process $XS(a, X)$ where in the example given, X is $\{b, c\}$.

$$\begin{aligned} &XS(a, X) = \\ &\quad \mathsf{let} \quad XSP(a, X) = init.a \rightarrow work.a \rightarrow \sqcap\, k : X \bullet init.k \rightarrow Skip \\ &\quad \mathsf{within} \quad XSP(a, X) \,[\![\, \{\, k : X \bullet init.k \,\} \,]\!]\, \Box\, k : X \bullet SP(k, acts) \end{aligned}$$

3.2 Multiple Instance Patterns

These patterns allow an activity in a workflow process to have more than one running, active instance at the same time. In our process descriptions we model a maximum of N instances of an activity running in any workflow process where N ranges over the strictly positive naturals $\mathbb{N}_1$. We define events *trig*, *done* and *ntrig* to denote the triggering, the completion and the cancelling of activity instances.

In this section we first define some CSP processes common to all multiple instance patterns described this paper. Each multiple instance pattern triggers

multiple instances of some activity. We define process $SR(a, n)$ to model the triggering of n out of N instances of activity a.

$$SR(a,n) = (|||\ k : \{1\,..\,n\} \bullet init.a \rightarrow done \rightarrow Skip) \fatsemi (|||\ k : \{1\,..\,N-n\} \bullet end \rightarrow Skip)$$

We define process $RP1(a)$ to model the N instances of activity a; standard CSP does not allow unbounded nondeterminism, as its semantics raises deep issues.

$$RP1(a) = |||\ k : \{1\,..\,N\} \bullet (SP(a, null)\,[init.null \leftarrow done] \,\Box\, end \rightarrow Skip)$$

Multiple Instances with a priori Design Time Knowledge - In this pattern multiple instances of activity b are triggered after activity a has completed execution. The number of instances is known at design time which means static within the model. Once all instances are completed, activity c is triggered. This pattern is modelled by the CSP process $DES(a, n, b, c)$ where n is the number of instances of activity b determined at design time. Process $DP(a, n, b, c)$ sets the number of instances of activity b before execution.

$$\begin{array}{l} DES(a,n,b,c) = \\ \quad \mathsf{let}\ \ DP(a,n,b,c) = init.a \rightarrow work.a \rightarrow SR(b,n) \fatsemi init.c \rightarrow Skip \\ \quad \mathsf{within}\ \ DP(a,n,b,c) \,[\![\, \{init.b, end, done\} \,]\!]\, RP1(b)) \setminus \{end, done\} \\ \qquad\qquad [\![\{init.c\} \,]\!]\, SP(c, acts) \end{array}$$

Multiple Instances with a priori Runtime Knowledge - The semantics of this pattern is somewhat ambiguous as it offers two patterns of behaviour. According to van der Aalst et al.'s original work [19], multiple instances of activities may be triggered in parallel with the correct synchronisation or the execution of these activities may be routed sequentially. In this paper, both cases are considered. Note in CSP $b \,\&\, P$ denotes the conditional expression **if** b **then** P **else** $Stop$.

First case: we define CSP process $PAR(a, b, c)$ to model multiple instances of activity b being triggered in parallel after activity a has completed execution. Activity c is triggered after instances of activity b have completed execution. The number of instances is not determined until runtime.

$$\begin{array}{l} SR1(a,b,c) = \\ \quad \mathsf{let}\ \ IT1(a,n) = exec \rightarrow (n \geq N \,\&\, SR(a,n) \,\Box\, n < N \,\&\, IT1(a,n+1) \sqcap SR(a,n)) \\ \quad \mathsf{within}\ \ init.a \rightarrow work.a \rightarrow (IT1(b,1) \fatsemi init.c \rightarrow Skip \sqcap SR(b,0) \fatsemi init.c \rightarrow Skip) \end{array}$$

$$\begin{array}{l} PAR(a,b,c) = (SR1(a,b,c) \,[\![\, \{init.y, end, done\} \,]\!]\, RP1(y)) \setminus \{exec, done, end\} \\ \qquad\qquad [\![\{init.z\} \,]\!]\, SP(z, acts) \end{array}$$

Second case: we define process $SR21(a, n)$ to model sequential triggering of n instances of activity a. Process $IT21(a, n)$ models a non-deterministic counter deciding upto N instances of a to be triggered.

$$\begin{array}{l} SR21(a,n) = \\ \quad \mathsf{let}\ \ SR(a,n) = (n=0) \,\&\, Skip \,\Box\, n > 0 \,\&\, init.a \rightarrow done \rightarrow SR(a,n-1) \\ \quad \mathsf{within}\ \ (|||\ k : \{1\,..\,N-n\} \bullet end \rightarrow Skip) \fatsemi SR(x,n) \\ IT21(a,n) = \\ \quad exec \rightarrow (n \geq N \,\&\, SR21(a,n) \,\Box\, n < N \,\&\, IT21(a,n+1) \sqcap SR21(a,n)) \end{array}$$

We model the second case by defining the process $MSEQ(a, b, c)$.

$$\begin{aligned} SR2(a, b, c) &= init.a \rightarrow work.a \rightarrow (IT21(b, 1) \fatsemi init.c \rightarrow Skip \\ &\qquad \sqcap SR(b, 0) \fatsemi init.c \rightarrow Skip) \\ MSEQ(a, b, c) &= (SR2(a, b, c) \,[\![\, \{init.b, end, done\} \,]\!]\, RP1(b)) \setminus \{exec, done, end\} \\ &\quad [\![\{init.c\}]\!]\, SP(c, acts) \end{aligned}$$

It is easy to see that the sequential triggering of multiple instances, defined by the CSP model $MSEQ'(a, b, c)$, *failure-refines* the parallel triggering defined by the model $PAR'(a, b, c)$.

$$PAR'(a, b, c) \sqsubseteq_F MSEQ'(a, b, c)$$

Multiple Instances without a priori Runtime Knowledge - This is a generalisation of the pattern "Multiple Instances with a priori Runtime Knowledge". After the completion of activity a, some instances of activity b are triggered. The number of instances is not decided at runtime, rather it is decided during the execution of instances. Activity c will only be triggered after all triggered instances of activity b have completed execution. We define the CSP process $NPAR(a, b, c)$ to model this pattern.

$$\begin{aligned} SR31(a, n) &= \Vert\!\Vert\, k : \{1 \,..\, N - n\} \bullet end \rightarrow Skip \\ IT31(a, n) &= n \geq N \;\&\; Skip \\ &\quad \Box\, n < N \;\&\; (init.a \rightarrow done \rightarrow IT31(a, n + 1)) \sqcap SR31(a, n) \\ SR3(a, b, c) &= init.a \rightarrow work.a \rightarrow (IT31(b, 0) \fatsemi init.c \rightarrow Skip \\ &\qquad \sqcap SR31(b, 0) \fatsemi init.c \rightarrow Skip) \\ NPAR(a, b, c) &= (SR3(a, b, c) \,[\![\, \{init.b, end, done\} \,]\!]\, RP1(b)) \setminus \{exec, done, end\} \\ &\quad [\![\{init.c\}]\!]\, SP(c, acts) \end{aligned}$$

3.3 State Based Patterns

This type of pattern captures external decisions at certain "states" within a workflow process. In previous patterns decisions on branching and looping are made a-priori and their semantics has been represented by the CSP internal choice operator. However, it is possible that these decisions are offered to the environment.

Deferred Choice - This is similar to "Multi-choice" pattern formalised above in which *either or both* activities b or c will be triggered after activity a has completed execution. However, in this pattern the choice is made by the environment. The semantics of this behaviour can be expressed by the CSP external choice operator. This pattern is modelled by the CSP process $DEF(a, X)$ where $X = \{b, c\}$.

$$DEF(a, X) = \textsf{let}$$
$$DC(a, X) = init.a \rightarrow work.a \rightarrow \Box\, y : X \bullet init.y \rightarrow Skip$$
$$\textsf{within}$$
$$DC(a, X) \,[\![\, \{\, y : X \bullet init.y \,\} \,]\!]\, \Box\, y : X \bullet SP(y, acts)$$

4 Case Study

In this section we study a realistic complex business process of a traveller reserving and booking airline tickets, adapted from the Web Service Choreography Interface (WSCI) specification [20]. A BPMN (Business Process Modelling Notation) diagram of the airline ticket reservation workflow is shown in Figure 1.

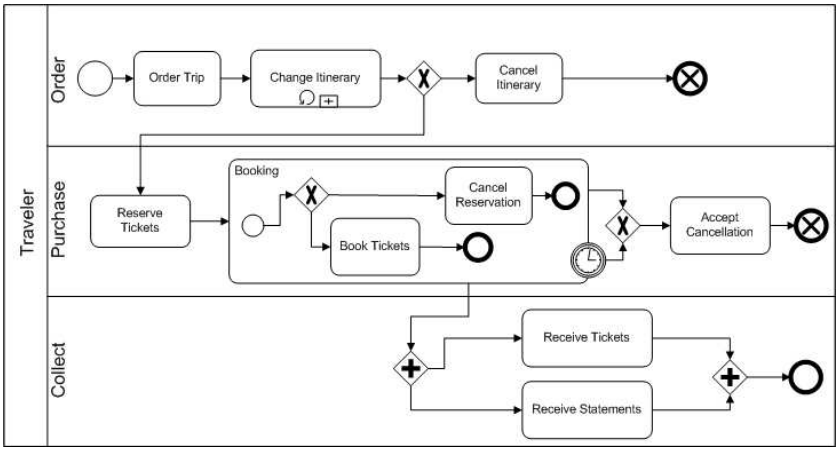

Fig. 1. Making an airline ticket reservation

4.1 Airline Ticket Reservation

We observe that the traveller can initiate the business process by ordering a trip. She may change her travel itinerary or cancel it. She may make a reservation with her chosen itinerary and before she confirms her booking she may at any time cancel her reservation or be notified of a cancellation by the airline due to the reservation period elapsing. After the traveller has confirmed her booking, she will receive the booked tickets and the statement for them.

The textual description given in the previous paragraph is somewhat ambiguous and a graphical representation like Figure 1 becomes difficult to read as

the complexity of the control flow of the business process increases. Furthermore both of these specifications lack a formal semantics and hence do not support checking *behavioural* properties like deadlock and livelock freedom at the implementation level. By modelling such business process in a process algebra like CSP, we can explore properties such as deadlock freedom by proving assertion (1). The notion of process refinement allows us to prove such assertion by model checking if P refines the *characteristic process* of the property we are interested in proving. For deadlock freedom we can model check the refinement assertion (2) where $DF = \sqcap\, x : \Sigma \bullet x \rightarrow DF$ is the most non-deterministic deadlock-free process.

$$\forall\, tr : \mathcal{T}(P) \quad \bullet \quad (tr, \Sigma) \notin \mathcal{F}(P) \tag{1}$$

$$DF \sqsubseteq_F P \tag{2}$$

We now turn to the definition of the CSP model for this workflow process. We use TL to denote the CSP process describing control flow of the workflow model and define the set $\mathcal{W}_{tl}$ as the set of workflow activities performed by TL. The set $\mathcal{S}_{tl}$ is defined as the set of CSP processes to represent the states of control flow of the workflow model.

$$\mathcal{W}_{tl} = \{order, change, cancelit, reserve, cancelres, timeout, accept, book, ticket, statemt\}$$
$$\mathcal{S}_{tl} = \{ORDER, CHANGE, CANCEL, RESERVE, CANRES, ACCEPT, BOOK, TIME, TICKET, STATE\}$$

We use the events *start*, *complete* and *fail* to denote the start, the completion and the abortion of the business process. We define αTL as the set of all events performed by process TL. We use the event *init.fault* to denote a fault has occurred and to represent an unsuccessful completion of the business process. We use the event *init.succ* to denote a successful completion of the business process; we use *init.fault* to denote an occurrence of cancellation during reservation; we use *init.itinfault* to denote an occurrence of cancellation before reservation. The events *init.null* and *init.acts* denote the triggering of out of scope activities as defined in Section 3.

$$\alpha TL = \{\, a : \mathcal{W}_{tl} \bullet init.a, work.a \,\}$$
$$\cup \{\, start, complete, fail, init.null, init.acts, init.fault, init.itnfault, init.succ \,\}$$

Here we specify some behavioural properties that the CSP process TL must satisfy. These properties are specified by the following assertions (3)–(6). Property (3) asserts TL to be a deadlock-free process; property (4) asserts that the business process must issue tickets if a booking has been made; here,

$$HB = init.book \rightarrow init.ticket \rightarrow HB$$

Property (5) asserts that the business process either aborts due to cancellation or completes successfully; here,

$$CSet = \{\, cancelres, cancelit, timeout \,\}$$
$$CC = (\sqcap\, x : CSet \bullet init.x \rightarrow fail \rightarrow CC)$$
$$\sqcap\ complete \rightarrow CC$$

Property (6) asserts that traveller can change her itinerary until she decides to make her reservation or to cancel her itinerary; here,

$$\begin{aligned} ITIN = {} & init.change \rightarrow ITIN \\ & \sqcap\ init.reserve \rightarrow (complete \rightarrow ITIN \sqcap fail \rightarrow ITIN) \\ & \sqcap\ init.cancelit \rightarrow fail \rightarrow ITIN \end{aligned}$$

$$DF \sqsubseteq_F TL \tag{3}$$
$$HB \sqsubseteq_F TL \setminus (\Sigma \setminus \alpha HB) \tag{4}$$
$$CC \sqsubseteq_F TL \setminus (\Sigma \setminus \alpha CC) \tag{5}$$
$$ITIN \sqsubseteq_F TL \setminus (\Sigma \setminus \alpha ITIN) \tag{6}$$

By employing the CSP models of the workflow patterns described in Section 3, we define each CSP process in $\mathcal{S}_{tl}$ as shown in Figure 2. Processes $SR4(a, b, X)$ and $DC1(a, X, Y)$ are defined in Figure 3. Process $SR4(a, b, X)$ is a combination of the processes $SR3(a, b, c)$ and $XSP(a, X)$ defined in Section 3's *Multiple Instances without a priori Runtime Knowledge* and *Exclusive Choice* patterns. Process $DC1(a, X, Y)$ is a combination of the processes $XSP(a, X)$ and $DEF(a, X)$ defined in Section 3's *Exclusive Choice* and *Deferred Choice* patterns.

$$\begin{aligned} ORDER &= SR4'(order, change, \{cancelit, reserve\}) \\ CHANGE &= RP1'(change) \\ CANCEL &= SP'(cancelit, fault) \\ RESERVE &= DC1'(reserve, \{cancelres, book\}, \{timeout\}) \\ CANRES &= SEQ'(cancelres, accept)[init.acts \leftarrow init.fault] \\ BOOK &= P'(book, \{ticket, statemt\}) \\ TIME &= SP'(timeout, fault) \\ TICKET &= SP'(ticket, succ) \\ STATE &= SP'(statemt, succ) \end{aligned}$$

Fig. 2. The definition of CSP processes in $\mathcal{S}_{tl}$

Figure 4 is the definition of the CSP process TL which models the semantics of the control flows of the airline ticket reservation business process model by parallel composition of processes from set $\mathcal{S}_{tl}$.

$$
\begin{aligned}
SR4(a, b, X) = \mathsf{let}\ \ & SR41(a, X) = (IT31(a, 0) \fatsemi \sqcap b : X \bullet init.b \rightarrow Skip) \\
& \qquad \sqcap (\sqcap b : X \bullet init.b \rightarrow Skip) \\
\mathsf{within}\ \ & init.a \rightarrow work.a \rightarrow SR41(b, X) \\
DC1(a, Y, Z) = \mathsf{let}\ \ & CHO(X, Y) = \sqcap b : X \bullet init.b \rightarrow Skip \\
& \qquad \Box (\Box c : Y \bullet init.c \rightarrow Skip) \\
\mathsf{within}\ \ & init.a \rightarrow work.a \rightarrow CHO(Y, Z)
\end{aligned}
$$

Fig. 3. The definition of processes $SR4(x, y, X)$ and $DC1(x, Y, Z)$

4.2 Composition and Refinement

Behavioural properties specified by assertions (3)–(6) can be readily checked by asking the FDR model checker about refinement assertions. Alternatively behavioural specifications can be composed to give a composite specification in which many of the assertions can be proved under a single refinement check. Property (7) asserts that the traveller may change her itinerary or cancellations may happen, otherwise she must commit to her itinerary and completes her transaction.

$$
\begin{aligned}
COMP = \ & init.change \rightarrow COMP \\
& \sqcap (\sqcap x : CSet \bullet init.x \rightarrow fail \rightarrow COMP) \\
& \sqcap init.book \rightarrow init.ticket \rightarrow complete \rightarrow COMP
\end{aligned}
$$

$$
COMP \sqsubseteq_F TL\ (\Sigma \setminus \alpha COMP) \tag{7}
$$

A CSP model of the business process like the one described in this paper can be placed in parallel with CSP models of other business processes to describe their collaboration where each business process interacts by communicating. The term *service choreography* has been coined for such collaboration description. In our airline ticket reservation example we can define a global business collaboration protocol between the traveller's workflow, the airline reservation system and the travel agent models. We use process names AL and TA to denote the control flow description of the airline reservation system and the travel agent workflow models. An example collaboration between these business processes is depicted as a BPMN diagram in Figure 5. In this paper we do not define AL and TA, a complete description of their models can be found elsewhere [22].

One effect that can be anticipated when composing complex process definitions like TL in parallel is an exponential state explosion. For example given that

$TL_c =$
  let
    $final = \{init.cancelres, init.book, init.timeout\}$
    $RECEIVE = TICKET \;|||\; STATE$
  within
    $((ORDER \;[\![\, \{init.change, end, done\} \,]\!]\; CHANGE) \setminus \{end, done, exec\})$
      $[\![\{init.cancelit, init.reserve\}\,]\!]\; (CANCEL \;\Box\; (RESERVE \;[\![\, \{\, x : final \bullet init.x \,\}]\!]$
        $(TIME \;\Box\; (CANRES \;\Box\; (BOOK \;[\![\, \{init.ticket, init.statemt\} \,]\!]\; RECEIVE)))))$
$TL =$
  let
    $decision = \{init.succ, init.fault, init.itinfault\}$
    $COM = start \rightarrow init.order \rightarrow Skip$
    $FIN = init.succ \rightarrow init.succ \rightarrow complete \rightarrow Skip$
    $FAULT = init.fault \rightarrow cancel \rightarrow Skip \;\Box\; init.itinfault \rightarrow cancel \rightarrow Skip$
  within
    $(COM \;[\![\, \{init.order\} \,]\!]\; (TL_c \;[\![\, decision \,]\!]\; (FIN \;\Box\; FAULT))) \;\fatsemi\; TL$

Fig. 4. The definition of processes TL

i ranges over $\{1\,..\,N\}$ for some positive integer $N > 0$, if each process P_i has just two states, then the expression $|||\; i : \{1\,..\,N\} \bullet P_i$ has 2^N. By compositionality and monotonicity of refinement, we can reduce individual component processes' state space by abstracting them into sequential processes. For simplicity suppose the process GB is the model of the choreography by composing individual participating business process in parallel where X is a set of some events. We denote sequential process by subscript s.

$$GB = (TA \;[\![\, X \,]\!]\; AL) \;[\![\, X \,]\!]\; TL$$

We abstract GB into process GB_s then if the refinement assertions (8)–(10) hold, by monotonicity of refinement we prove assertion (11).

$$GB_s \;=\; (TA_s \;[\![\, X \,]\!]\; AL_s) \;[\![\, X \,]\!]\; TL_s$$
$$AL_s \sqsubseteq_F AL \tag{8}$$
$$TL_s \sqsubseteq_F TL \tag{9}$$
$$TA_s \sqsubseteq_F TA \tag{10}$$
$$GB_s \sqsubseteq_F GB \tag{11}$$

if GB_s refines some property defined by the characteristic process $SPEC$, by transitivity of refinement we prove GB also refines $SPEC$.

$$SPEC \sqsubseteq_F GB_s \wedge GB_s \sqsubseteq_F GB \Rightarrow SPEC \sqsubseteq_F GB$$

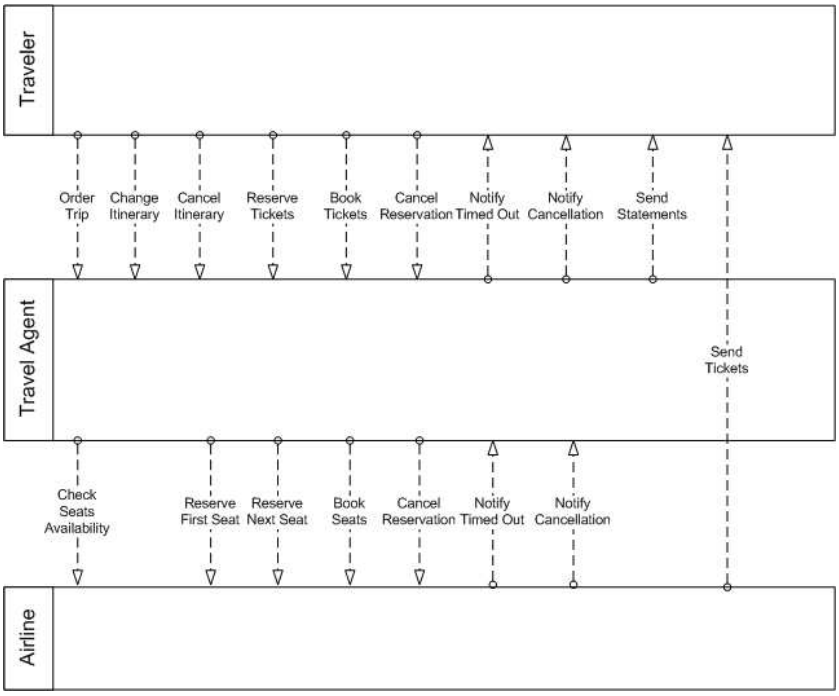

Fig. 5. Airline Ticket Reservation Choreography

5 Related Work

Little research has been done to date into the application of CSP to workflow specification and verification. We have recently defined a process semantics for BPMN in CSP [23], which allows formal comparison between workflow processes described in BPMN and encourages automated tool support for the notation. The only other approach that has applied CSP in workflow process [16] did so as an extension of abstract machine notation for process specification within the domain of compositional information systems.

Other process algebras used to model workflow patterns include π-calculus [13] and CCS [15], a subset of π-calculus. These formalisations did not focus on process-based behavioural specification, and they did not demonstrate the applications of their models in workflow design. Moreover, the operational semantics of π-calculus and CCS do not provide a refinement relation; we have demonstrated in this paper that refinement is useful in the development of workflow processes, because it allows formal comparisons between workflows. A similar observation applies to the work of van der Aalst et al. [9, 17] using Petri nets. Despite Puhlmann et al.'s advocacy of mobility in workflow modelling, our CSP models suggest it is not necessary when modelling static control flow interactions.

However, it is still possible to introduce mobility into standard CSP semantics if needed; an attempt has been made by Welch et al. [21].

Although Stefansen [15] mentioned a model checker called Zing which bears some similarities with FDR, implementing a conformance checker based on stuck-freedom [5], it is more discriminative and only resembles the CSP concept of deadlock-freedom.

6 Conclusion

In this paper we described some CSP models of van der Aalst et al.'s workflow patterns to construct workflow processes. We then modelled a realistic workflow process by using models of workflow patterns and subsequently demonstrated the use of process refinement for asserting behavioural properties about the workflow process. These properties were described by process-based specifications defined in CSP and assertions were then proved automatically using the CSP model checker FDR. Like any development of a complex system, the application of refinement in workflow design means that development of a workflow design into an implementation becomes incremental. Due to monotonicity and transitivity of process refinement, it is possible to minimise exponential state explosion when model checking complex process by abstracting its individual component processes into corresponding sequential processes.

Future work will include the following:

- extend the CSP model described in this paper into a global domain, hence allowing a unified treatment of workflow orchestration and choreography;
- augment our current CSP model with a well-defined exception and compensation semantics, perhaps building on Butler's compensating CSP [1];
- combine our CSP control flow model with a dataflow semantics to allow a unified treatment of the semantics of workflow processes, perhaps building on Josephs' CSP dataflow model [8].
- automate the translation from workflow descriptions in BPMN to CSP processes, based on our recent work on BPMN semantics [23].

7 Acknowledgements

We would like to thank anonymous referees for useful suggestions and comments. This work is supported by a grant from Microsoft Research.

References

1. M. Butler, T. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In *Proceedings of 25 Years of CSP*, volume 3525 of *LNCS*, pages 133–150, 2005.

2. S. Creese. Industrial Strength CSP: Opportunities and Challenges in Model-Checking. In *Proceedings of 25 Years of CSP*, volume 3525 of *LNCS*, page 292, 2005.
3. J. Davies. The CSP Package, Mar. 2001. `ftp://ftp.comlab.ox.ac.uk/pub/CSP/LaTeX/csp.sty`.
4. Formal Systems (Europe) Ltd. *Failures-Divergences Refinement, FDR2 User Manual*, 1998. `www.fsel.com`.
5. C. Fournet, T. Hoare, S. K. Rajamani, and J. Rehof. Stuck-Free Conformance. In *16th International Conference on Computer Aided Verification*, volume 3114 of *LNCS*, pages 242–254, Jan. 2004.
6. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
7. D. Hollingsworth. The Workflow Reference Model. Technical Report WFMC-TC-1003, Workflow Management Coalition, Jan. 1995.
8. M. Josephs. Models for Data-Flow Sequential Processes. In *Proceedings of 25 Years of CSP*, volume 3525 of *LNCS*, pages 85–97, 2005.
9. B. Kiepuszewski. *Expressiveness and Suitability of languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002.
10. J. Lawrence. Practical Application of CSP and FDR to Software Design. In *Proceedings of 25 Years of CSP*, volume 3525 of *LNCS*, pages 151–174, 2005.
11. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
12. R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.
13. F. Puhlmann and M. Weske. Using the π-Calculus for Formalizing Workflow Patterns. In *BPM 2005*, volume 3649 of *LNCS*, pages 153–168. Springer-Verlag, 2005.
14. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.
15. C. Stefansen. SMAWL: A SMAll workflow language based on CCS. Technical Report TR-06-05, Harvard University, Mar. 2005.
16. S. A. Stupnikov, L. A. Kalinichenko, and J. S. Dong. Applying CSP-like Workflow Process Specifications for their Refinement in AMN by Pre-existing Workflows. In *Proceedings of ADBIS'2002*, Sept. 2002.
17. W. M. P. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, 1997.
18. W. M. P. van der Aalst. Pi Calculus Versus Petri Nets: Let Us Eat Humble Pie Rather Than Further Inflate the Pi Hype. *BPTrends*, 3(5):1–11, May 2005.
19. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.
20. W3C. *Web Service Choreography Interface 1.0*, 2002. `www.w3.org/TR/wsci/`.
21. P. H. Welch and F. R. M. Barnes. Mobile Barriers for occam-pi: Semantics, Implementation and Application. In *Communicating Process Architectures 2005*, volume 63 of *Concurrent Systems Engineering Series*, pages 289–316, Sept. 2005.
22. P. Y. H. Wong. Towards a unified model for workflow orchestration and choreography, 2006. Transfer dissertation, Oxford University Computing Laboratory.
23. P. Y. H. Wong and J. Gibbons. A Process Semantics for BPMN, 2007. submitted for publication.