Towards A Unified Model for Workflow Processes (extended abstract)

Peter Y.H. Wong
Oxford University Computing Laboratory

May 2006

1 Introduction

The emergence of workflow management systems offers supports for composing, coordinating and monitoring the execution of human tasks and component services. One of the key challenges is to provide a formal semantics and the facility to model, analyse and reason about workflow processes at both *orchestration* and *choreography* levels. This paper describes the formalisation of van der Aalst et al.'s workflow patterns [12] using the process algebra CSP [5, 9], and examines a case study of a real life business process. The modelling example in this paper captures both workflow orchestration and choreography. Whereas workflow orchestrations represents a local, single participant viewpoint of the workflow model, workflow choreography captures collaboration between processes involving multiple participants and elevates workflow models to a global viewpoint where the primary concern is external, observable behaviour. The formalisation described in this paper lays down the foundation for a unified model for workflow process specification, reasoning and verification. The complete formalisation of workflow patterns can be found in a longer paper [15]; the complete formalisation of the workflow choreography model will appear in a future publication.

2 Workflow Patterns

Workflow patterns, introduced by van der Aalst et al., are the "gold standard" for benchmarks of workflow languages [12]. These patterns range from simple constructs to complex routing primitives; their scope is limited to static control flow.

We model each control flow pattern in CSP. A basic workflow activity is defined as the CSP process $SP(a, b)$. It is first able to perform the event $init.a$. This represents an external trigger to the start of the activity a. After the trigger has occurred, the event $work.a$, which represent activity a, will be ready to perform. After performing $work.a$, the process is then ready to perform the event $init.b$ which is the trigger or activity b defined in another process. The process $P(a, X)$ extends the above definition into a more generic process description.

$$\begin{aligned} SP(a,b) &= init.a \rightarrow work.a \rightarrow init.b \rightarrow STOP \\ P(a,X) &= init.a \rightarrow work.a \rightarrow |||\, b : X \bullet init.b \rightarrow STOP \end{aligned}$$

The processes $SP(a, b)$ and $P(a, b)$ are the basic primitives of the formalism of all subsequent workflow models. In general any process Q representing some workflow activities will have a corresponding process Q'' which has the execution of its workflow activities internalised to avoid any external intervention. In this paper we present one of the workflow pattern formalisations utilised in our business process case study. More detailed textual descriptions of the pattern can be found in van der Aalst et al.'s original work [12]; the complete formalisation of all workflow patterns can be found in the longer paper [15].

Multiple Instances with a priori Runtime Knowledge - In this pattern multiple instances of activity b are triggered sequentially after activity a has completed execution. Two CSP events *sig* and *done* are introduced in the formalisation of this pattern to record the number of instances of activity b being triggered at runtime and to record the number of instances having completed execution respectively. Activity c is triggered after the determined number of instances of activity b have completed execution. This pattern is modelled by the process $RUNSEQ(a, b, c)$.

$$\begin{aligned} INIT(b) &= init.b \rightarrow (INIT(b) \sqcap SKIP) \fatsemi sig \rightarrow done \rightarrow SKIP \\ SET(a,b,c) &= init.a \rightarrow work.a \rightarrow ((INIT(b) \fatsemi init.c \rightarrow STOP) \sqcap init.c \rightarrow STOP) \\ RUN(a,b) &= init.a \rightarrow (sig \rightarrow work.a \rightarrow init.b \rightarrow done \rightarrow STOP \;|||\; RUN(a,b)) \\ RUNSEQ''(a,b,c) &= (SET''(a,b,c) \;[\![\, \{sig, init.b, done\} \,]\!]\; RUN''(b, null)) \;[\![\, \{init.c\} \,]\!]\; SP''(c, acts) \end{aligned}$$

3 Case Study: Ticket Reservation System

This case study examines a business process of reserving and booking airline tickets, defined in an XML choreography description language called Web Service Choreography Interface (WSCI) [13]. This example includes three participants: a traveler, a travel agent and an airline reservation system, each described by a WSCI `<interface>` construct. This section gives an overview of the orchestration of the traveler, and describes our approach to modelling the choreography of the complete system. Each `<action>` defined in a WSCI interface is modelled as a CSP process in the form of a workflow activity described in Section 2; in particular, we model each interface action by its action name, porttype and operation and so an interface action `a1` which performs an operation `o1` from porttype `p1` is modelled by a corresponding CSP compound event $a1.o1.p1$.

Traveler's Interface - In order to model the traveler's orchestration, the pattern "Deferred Choice" is combined with the "Exclusive Choice" pattern [15], and is defined by the process $DC1(a, b, c)$.

$$\begin{aligned} INIT2(m, n) &= (\sqcap x : m \bullet init.x \rightarrow STOP) \,\Box\, (\Box\, x : n \bullet init.x \rightarrow STOP) \\ DC1(a, b, c) &= init.a \rightarrow work.a \rightarrow INIT2(b, c) \end{aligned}$$

Also the "Multiple Instances with a priori Runtime Knowledge" pattern described in Section 2 is generalised to allow the invocation of parallel activities, and is defined below. Note processes $INIT(b)$ and $RUN(a, b)$ remain the same.

$$SET2(a, b, X) = init.a \rightarrow work.a \rightarrow ((INIT(b) \,{}_{\S}\, \sqcap c : X \bullet init.c \rightarrow STOP) \sqcap (\sqcap c : X \bullet init.c \rightarrow STOP))$$

In this interface the traveler can order a trip by setting up an itinerary for airline tickets; this is modelled by the process *ORDER*. She may then change her itinerary many times or cancel the itinerary, as modelled by processes *CHITIN* and *CAITIN* respectively. Thereafter she can reserve the seats, as modelled by the process *RTICKET*. After the reservation she can proceed with the booking, or the reservation may be cancelled due to expiry; these are modelled by processes *BOOK* and *TIME* respectively. After she has booked her ticket, the travel agent and the airline will send her the tickets and statement, modelled by the processes *TICKET* and *STATE* respectively. The CSP process *TRAVEL* models the complete orchestration of the traveler.

$$\begin{aligned} ORDER &= SR''(Aordertrip.tta.ordertrip, Achangeitinerary.tta.changeitinerary, \\ &\qquad \{Acancelitinerary.tta.cancelitinerary, Areservetickets.tta.reservetickets\}) \\ CHITIN &= RUN''(Achangeitinerary.tta.changeitinerary, null) \\ CAITIN &= SP''(Acancelitinerary.tta.cancelitinerary, itin) \\ RTICKET &= DC1''(Areservetickets.tta.reservetickets, \\ &\qquad \{Acancelreservation.tta.requestcancellation, Abooktickets.tta.booktickets\}, \\ &\qquad \{Areservationtimedout.tta.acceptcancellation\}) \\ CANRES &= SP''(Acancelreservation.tta.requestcancellation, \\ &\qquad Aacceptcancellation.tta.acceptcancellation)[init.acts \leftarrow init.fault] \\ BOOK &= P''(Abooktickets.tta.booktickets, \\ &\qquad \{Areceivetickets.ta.receivetickets, Areceivestatement.tta.receivestatement\}) \\ TIME &= SP''(Areservationtimedout.tta.acceptcancellation, fault) \\ TICKET &= SP''(Areceivetickets.ta.receivetickets, acts) \\ STATE &= SP''(Areceivestatement.tta.receivestatement, acts) \end{aligned}$$

$$\begin{aligned} &TRAVEL = \\ &\quad let \\ &\qquad switch1 = \{init.Areservetickets.tta.reservetickets, init.Acancelitinerary.tta.cancelitinerary\} \\ &\qquad switch2 = \{init.Acancelreservation.tta.requestcancellation, \\ &\qquad\quad init.Abooktickets.tta.booktickets, init.Areservationtimedout.tta.acceptcancellation\} \\ &\qquad receive = \{init.Areceivetickets.ta.receivetickets, init.Areceivestatement.tta.receivestatement\} \\ &\quad within \\ &\qquad ((ORDER \,[\![\, \{init.Achangeitinerary.tta.changeitinerary, done\} \,]\!]\, CHITIN) \,[\![\, switch1]\!] \\ &\qquad ((CAITIN \,|||\, RTICKET) \,[\![\, switch2]\!] \,((TIME \,|||\, (CANRES \,|||\, BOOK)) \\ &\qquad [\![receive]\!] \,(TICKET \,|||\, STATE)))) \end{aligned}$$

Modelling Choreography - Apart from traveler process, the choreography of the complete airline tickets reservation business process also includes the airline reservation system and the travel agent processs, both of which are modelled as processes *AIRLINE* and *AGENT* respectively. In this paper we describe our approach to model this choreography.

The WSCI definition describes a multi-participant view of the overall message exchange by means of the global model (`<model>`). A global model describes the choreography between participants defined by `<interface>` constructs via `<connect>` constructs. Each `<connect>` defines the message flow between a pair of operations performed by two different participants, hence indicating which operations from which participants should exchange messages.

Based on the WSCI global model of the ticket reservation system, the collaboration of the three participants is modelled by the CSP process *MODEL* where the set *CONNECT* is the set of pairs of WSCI operations modelled as a set of pairs of CSP events. The complete choreography, defined as the process *GLOBAL*, is modelled by composing the three participant processes and the process *MODEL* in parallel.

$$MODEL = \Box(a, b) : CONNECT \bullet init.a \rightarrow init.b \rightarrow MODEL$$
$$INTER = AIRLINE \,[\![\, \{init.fault\} \,]\!]\, (TRAVEL \,[\![\, \{init.itin, init.fault\} \,]\!]\, AGENT)$$
$$START = start \rightarrow init.Aordertrip.tta.ordertrip \rightarrow STOP$$
$$ABORT = init.fault \rightarrow init.throw \rightarrow STOP \,\Box\, init.itin \rightarrow init.throw \rightarrow STOP$$
$$END = init.throw \rightarrow cancel \rightarrow STOP$$
$$SUCC = init.acts \rightarrow init.acts \rightarrow init.acts \rightarrow init.acts \rightarrow complete \rightarrow STOP$$
$$GLOBAL = START \,[\![\, startSet \,]\!]\, ((INTER \,[\![\, interface \,]\!]\, MODEL) \,\triangle\, END) \,[\![\, endSet \,]\!]\, (ABORT \,|||\, SUCC))$$

Since the process *GLOBAL* describes precisely the complete dynamic control flow of the choreography, by using refinement [9], we can formally make assertions about properties which the business process must satisfy. One such assertion is deadlock freedom upto the point of performing either the event *complete* or *cancel* and is expressed as the following *failures* refinement where Σ is the set of all possible events.

$$SPEC \sqsubseteq_F GLOBAL \setminus hide$$
$$SPEC = start \rightarrow (complete \sqcap cancel)$$
$$hide = \Sigma \setminus \{start, complete, cancel\}$$

Assertions made using refinements can be formally verified using the CSP model checker FDR [2]. Note the CSP model given in this paper is simplified to illustrate our approach to modelling choreography. We have excluded the application of constraints by parallel compositions as conjunction [9] for avoiding unwanted deadlock, and the reduction of the model's state by *piece-wise abstraction* which leverages on the monotonicity and compositionality of refinement.

4 Related Work

Currently little research has been done into the application of CSP to workflow specification. The only approach that has applied CSP in workflow process [11] applied CSP 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 [8] and CCS [10], a subset of π-calculus. These formalisations did not focus on formal verification and they did not demonstrate their applications. Moreover the semantics of π-calculus and CCS do not provide a refinement relation, which we believe is crucial in formal specification and verification. 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, as shown by Welch et al. [14].

Whereas Puhlmann et al.'s formalisation is not oriented towards automated verification [8] and Stefansen's can be "cryptic" and is not machine-readable [10], we have implemented our CSP models using standard CSP syntax and it is possible to translate our models directly into CSP_M, the machine-readable dialect of CSP [9], for model checking. Although Stefansen [10] mentioned a model checker called Zing which bears some similarities with FDR, implementing a conformance checker based on stuck-freedom [4], it is more discriminative and only resembles the CSP concept of deadlock-freedom.

In terms of workflow choreography, other process algebras used to reason about choreography included works from Brogi et al. [1] and Foster et al. [3]. Brogi et al. applied CCS [7] to formalise WSCI and addressed issues such as the definition of compatibility and replaceability tests between Web services. Foster et al. applied FSP notation and Message Sequence Charts to reason about obligation policy of Web service choreographies defined in Web Service Choreography Description Language (WS-CDL) [6].

5 Conclusion

This paper described a formal semantics for workflow specification by expressing workflow patterns as CSP processes and an overview of a case study based on real life business processes choreography. Each participant within the choreography is a complex workflow process. The formalisation of these workflow processes employed the CSP models of several complex workflow patterns such as multiple instances, arbitrary cycle and state-based patterns. The resulted CSP model is a complete model of dynamic control flow of the workflow choreography which can be used in formal verification and subsequently model checked using FDR.

Future work will include investigating other choreography description languages such as WS-CDL [6] and developing a "generic" unified model for workflow orchestration and choreography. We will also extend our present CSP model with a formal exception and compensation handling mechanism. Furthermore, we will extend our model with dataflow semantics, hence unifying the semantics of workflow processes in both business and scientific domains, including the clinical domain.

References

[1] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing Web Services Choreographies. In *Electronic Notes in Theoretical Computer Science 105*, pages 73–94, 2004.

[2] Formal Systems (Europe) Ltd. *Failures-Divergences Refinement, FDR2 User Manual*, 1998. `www.fsel.com`.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-Based Analysis of Obligations in Web Service Choreography. In *IEEE International Conference on Internet and Web Applications and Services*, 2006.

[4] C. Fournet, T. Hoare, S. Rajamani, and J. Rehof. Stuck-Free Conformance. In *Proceedings of 16th International Conference on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 242–254, Jan. 2004.

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

[6] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. *Web Services Choreography Description Language 1.0*, 2005. W3C Candidate Recommendation.

[7] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[8] F. Puhlmann and M. Weske. Using the π-Calculus for Formalizing Workflow Patterns. In *BPM 2005*, volume 3649 of *Lecture Notes in Computer Science*, pages 153–168. Springer-Verlag, 2005.

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

[10] C. Stefansen. SMAWL: A SMAll Workflow Language based on CCS. Technical Report TR-06-05, Harvard University, Division of Engineering and Applied Sciences, Mar. 2005.

[11] S. Stupnikov, L. Kalinichenko, and J. Dong. Applying CSP-like Workflow Process Specifications for their Refinement in AMN by Pre-existing Workflows. In *Proceedings of the Sixth East-European Conference on Advances in Databases and Information Systems (ADBIS'2002)*, Sept. 2002.

[12] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.

[13] W3C. *Web Service Choreography Interface (WSCI) 1.0*, Nov. 2002. `www.w3.org/TR/wsci/`.

[14] P. Welch and F. 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.

[15] P. Wong. A Process Algebraic Approach to Workflow Verification, 2006. Submitted for publication.