Cost Analysis for an Actor-Based Workflow Modelling Language *

Muhammad Rizwan Ali[1] and Violet Ka I Pun[1]

Western Norway University of Applied Sciences, Norway
{mral,vpu}@hvl.no

1 Introduction

Workflow planning is a process to define a series of tasks that are carried out based on user-defined rules to accomplish some particular goals. These goals can be organizing surgeries in a hospital, coordinating different maintenance activities on an oil platform, scheduling takeoff and landing of planes at an airport, and so on. In addition, workflow planning allows the early discovery of potential problems. Also, workflow planning helps to anticipate unforeseen situations that may happen during the implementation of workflows, which enables advance preparation of reparative steps and alternative implementations.

With the boom of e-commerce and the digital industry, organisations usually work beyond their boundaries and collaborate with others to achieve specific goals. Cross-organisational workflows usually comprise various workflows and very often they run concurrently in different departments within the same organisation or in different organisations.

Domain-specific knowledge is required to perform optimal resource allocation and task management, making cross-organisational workflow planning especially challenging. Furthermore, changing workflows is error-prone: a change in one workflow may propagate to other concurrently running workflows, and a minor mistake might have significant negative consequences.

Considerable research has been done in the area of workflow automation and tools such as Process-Aware Information Systems (PAIS) [8] and Enterprise Resource Planning (ERP) systems [1, 2] have been developed to assist workflow planning. Furthermore, Petri-nets [12] has been used to formalize Business Process Model and Notation (BPMN) [6]. Moreover, Temporal Logic of Actions (TLA) [5] has been used for the formalisation of the workflows. However, the available tools and techniques often lack adequate resource management or do not support domain-specific knowledge, making cross-organisational workflow planning relatively manual. Moreover, the planners of cross-organisational workflows may not have a common understanding of all the collaborative workflows, which can be catastrophic.

In this extended abstract, we first present a formal modelling language $\mathcal{R}$PL. The language has specific notions for task dependencies, resource usage and time consumption, allowing planners to couple multiple workflows employing resources and task dependencies. Moreover, we also present a static analysis based on the work in [11] to over-approximate the worst execution time of the workflows modelled as an $\mathcal{R}$PL program by translating the program into a set of cost equations. The analysis also allows planners to predict the effects of the changes in cross-organisational workflows in terms of execution time before the actual implementation.

2 Formal Workflow Modelling Language $\mathcal{R}$PL

The language $\mathcal{R}$PL is inspired by ABS [10], an active object language extending the Actor model of concurrency with asynchronous method calls and synchronisation using futures. It

*Partially supported by *CroFlow: Enabling Highly Automated Cross-Organisational Workflow Planning* and *COEMS training network*.

has specific notions for resource acquisition and task dependencies. The syntax is given below.

$$
\begin{array}{ll}
P ::= R\ \overline{Cl}\ \{\overline{T\ x};\ s\} & e ::= x \mid g \mid \textbf{this} \\
Cl ::= \textbf{class}\ C\ \{\overline{T\ x};\ \overline{M}\} & g ::= b \mid f? \mid g \wedge g \\
M ::= Sg\ \{\overline{T\ x};\ s\} & s ::= x = rhs \mid \textbf{skip} \mid \textbf{if}\ e\ \{s\} \mid \textbf{wait}(f) \mid \textbf{return}\ e \\
Sg ::= B\ m(\overline{T\ y}) & \quad \mid \textbf{hold}(\overline{r, e}) \mid \textbf{release}(\overline{r, e}) \mid \textbf{cost}(e) \mid s\ ;\ s \\
B ::= \texttt{Int} \mid \texttt{Bool} \mid \texttt{Unit} & rhs ::= e \mid \textbf{new}\ C \mid f.\textbf{get} \\
T ::= C \mid B \mid \textbf{Fut}\langle B\rangle & \quad \mid m(x, \overline{e})\ \textbf{after}\ \overline{f?} \mid\ !m(x, \overline{e})\ \textbf{after}\ \overline{f?}
\end{array}
$$

An $\mathcal{R}$PL program P comprises resources R, a sequence of class declarations $\overline{Cl}$ and a main method body $\{\overline{T\ x};\ s\}$. Types T in $\mathcal{R}$PL are basic types B, a class C and future types $\textbf{Fut}\langle B\rangle$, which types asynchronous method invocations. A class declaration has a class name C and a class body $\{\overline{T\ x};\ \overline{M}\}$ comprising state variables and methods of the class. Methods in $\mathcal{R}$PL have a method signature Sg followed by a method body $\{\overline{T\ x};\ s\}$. Expressions e include guards g, variables x and self-identifier **this**.

Statements like sequential composition, assignment, conditional, **skip**, and **return** are standard. $\mathcal{R}$PL uses $\textbf{hold}(\overline{r, e})$ and $\textbf{release}(\overline{r, e})$ to acquire and return e number of resources r. Statement $\textbf{wait}(f)$ suspends the current process until future f is resolved, while other processes in the same object can be scheduled for execution. Statement $\textbf{cost}(e)$, the only term in $\mathcal{R}$PL that consumes time, represents e units of time advancement. The right-hand side rhs of an assignment includes expressions e, object creation $\textbf{new}\ C$, method invocations and synchronisation. Communication in $\mathcal{R}$PL is based on method calls, which can be either synchronous, written as $m(x, \overline{e})\ \textbf{after}\ \overline{f?}$, or asynchronous, written as $!m(x, \overline{e})\ \textbf{after}\ \overline{f?}$, where x is the callee object and $\overline{f?}$ is a sequence of futures that must be resolved prior to invoking method m. An asynchronous method invocation is associated to a future variable of type $\textbf{Fut}\langle B\rangle$, where B is the return type of the invoked method. Moreover, the expression $f.\textbf{get}$ blocks all execution in the object until future f is resolved. The full semantics, the type-system and subject reduction of $\mathcal{R}$PL can be found in the technical report [4].

3 Analysis of $\mathcal{R}$PL Program

The analysis over-approximates the overall execution time of cross-organisational workflows modelled in $\mathcal{R}$PL. The analysis enables the planners to predict the effect of changes on the overall execution time of the collaborative workflows before implementation. We assume all $\mathcal{R}$PL programs terminate and all methods return and are synchronised, i.e., the return values are retrieved.

The flow of the cost analysis is depicted in Figure 1. Given an $\mathcal{R}$PL program modelling some cross-organisational workflows, for each of the method in the program, the analysis first identifies all the objects whose processes have implicit dependencies; that is, the processes of these objects may in turn influence the process pools of other objects through method invocations and synchronisations. The analysis then translates the method body by parsing each statement into a set of cost constraints. The solution to this constraint set correspond to the worst execution time of the method. By translating the body of all methods, including `main`, of the $\mathcal{R}$PL program, the analysis produces a set of constraints, which can be fed into an off-the-shelf constraint solver (e.g., [9, 3]), to capture the (over-approximated) worst execution time for the program.

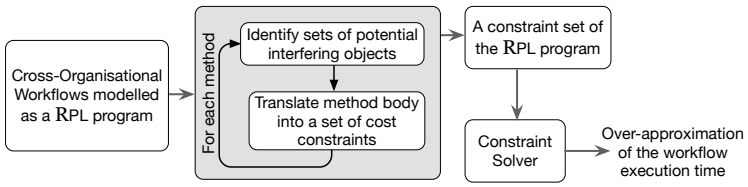

Figure 1: Cost Analysis of $\mathcal{R}$PL programs

4 Properties

Theorem 1 below formuThe correctness of our analysis follows the. This theorem states that for an $\mathcal{R}$PL program $\mathcal{P}$, the execution time to reach from initial state cn to any reachable state cn', calculated by $\texttt{time}(cn')$, will never exceed the $\mathcal{U}(\mathcal{P})$, cost calculated by the analysis presented in Section 3.

Theorem 1. *Let $\mathcal{P}$ be an $\mathcal{R}$PL program, whose initial configuration is cn, and $\mathcal{U}(\mathcal{P})$ be the closed-form solution of $\mathcal{P}$. If $cn \Rightarrow^* cn'$, then $\texttt{time}(cn') \leq \mathcal{U}(\mathcal{P})$.*

5 Conclusion

In this paper, we have presented a formal language, $\mathcal{R}$PL, intended for modelling cross-organisational workflows. Additionally, we have proposed a static analysis to over-approximate the computational time of an $\mathcal{R}$PL program. As for the immediate next steps, we plan to enrich the language such that the resource features, e.g., the experience of a person and speed of a machine, can be explicitly specified and to extend the analysis to handle non-terminating programs and unsynchronised method invocations. Furthermore, we intend to develop verification techniques to ensure the correctness of workflow models in $\mathcal{R}$PL for cross-organisational workflows. A possible starting point is to investigate how to extend KeY-ABS [7], a deductive verification tool for ABS, to support $\mathcal{R}$PL.

References

[1] Safran project. http://www.safran.com/. Accessed in Oct-2021.

[2] SAP ERP. http://www.sap.com/. Accessed in Oct-2021.

[3] Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. Closed-form upper bounds in static cost analysis. *Journal of automated reasoning*, 46(2):161–203, 2011.

[4] Muhammad Rizwan Ali and Violet Ka I Pun. Cost Analysis for an Actor-Based Workflow Modelling Language (long version). Research Report 15, Western Norway Univ. of Applied Sciences, 2021.

[5] Jose L Caro. Proposing a formal method for workflow modelling: Temporal logic of actions (TLA). *Int. Journal of Computer Science Theory and Application*, 1(1):1–11, 2014.

[6] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Formal semantics and analysis of BPMN process models using Petri nets. *Queensland Univ. of Technology, Tech. Rep.*, pages 1–30, 2007.

[7] Crystal Chang Din, Richard Bubel, and Reiner Hähnle. KeY-ABS: A deductive verification tool for the concurrent modelling language ABS. In *Intl. Conf. on Automated Deduction*, volume 9195 of *LNCS*, pages 517–526. Springer, 2015.

[8] Marlon Dumas, Wil M van der Aalst, and Arthur H Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005.

[9] Antonio Flores-Montoya and Reiner Hähnle. Resource analysis of complex programs with cost equations. In *Asian Symp. on Programming Languages and Systems*, pages 275–295. Springer, 2014.

[10] Einar Broch Johnsen, Reiner Hähnle, Jan Schäfer, Rudolf Schlatte, and Martin Steffen. ABS: A core language for abstract behavioral specification. In Bernhard Aichernig, Frank S. de Boer, and Marcello M. Bonsangue, editors, *Int. Symp. on Formal Methods for Components and Objects*, volume 6957 of *LNCS*, pages 142–164. Springer, 2011.

[11] Cosimo Laneve, Michael Lienhardt, Violet Ka I Pun, and Guillermo Román-Díez. Time analysis of actor programs. *Journal of Logical and Algebraic Methods in Programming*, 105:1–27, 2019.

[12] Wil M van der Aalst. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.