2012 IEEE Fourth International Conference on Technology for Education

Algebraic modelling of educational workflows

Ankur Goel
Software Engineering Research Lab
International Institute of Information Technology
Hyderabad, INDIA
Email: ankur.goel@research.iiit.ac.in

Venkatesh Choppella
Software Engineering Research Lab
International Institute of Information Technology
Hyderabad, INDIA
Email: venkatesh.choppella@iiit.ac.in

***Abstract*—Workflows are ubiquitous in educational ERP and process management. The question we address in this paper is how to specify and design verifiable workflows for educational processes. To specify workflows, we borrow a simple algebraic notation from computer science. We illustrate the use of this notation through a series of typical workflow examples. We show how an algebraic specification is more expressive than a graphical specification. We then show how these algebraic specifications may be represented in the Scribble specification language, which has in built tools for verification.**

Keywords- Education, processes, Scribble, algebraic specifications, workflows

I. Introduction

Most educational institutions today are growing at a rate like never before with many evolving into large scale virtual organizations. This impacts the requirements of educational software in terms of the scale and complexity. One component of this complexity is the type of complex workflows that capture educational processes.

Any implementation of enterprise resource planning (ERP) for educational systems will need to model such complex workflows. By workflows, we mean a collection of coordinated tasks designed to carry out a well-defined process typically involving multiple identities like teachers, students, administrators, etc. [1]. Conducting an exam is a workflow, even taking attendance can be expressed as a workflow. In simple words, it is the "flow of work", which we try to formally specify in this paper. For the purpose of this paper, we focus on the specification aspect of workflows and leave out the implementation aspect. We do not consider specific implementations of systems that use the modelling methodology we specify. Interested readers may refer to Pantoto [2], a software focused on building such systems based on workflows that uses some ideas based on the models described in this paper. Time constraints are also an important factor while designing workflows [3], but we leave that in the paper for the sake of simplicity.

Graphical notation, e.g, Petri nets are a common way of specifying workflows. However, we prefer to use an algebraic notation, which is more general in that it can express more complex workflows. To this end, we use the π-calculus to model and specify educational workflows [4]. π-calculus is an idealized concurrent programming language which was originally designed to express inter-process communication between computers. Workflows also involve communication between entities (processes), hence the π-calculus notation seems to be most suitable for expressing workflows. Also, educational workflows are mostly parallel and dynamic in nature, π-calculus being designed for concurrent systems enables us to express almost all workflows algebraically.

Workflows are expressed as a sequence of π-calculus process definitions [5], followed by a π-calculus process expression, usually a parallel composition, which denotes the initial state of the workflow. Each user participates in zero or more workflows as a π-calculus process in one of a set of *states*. The interface of the state of each user with respect to a workflow is given by a defining equation specifying how that process evolves upon *actions*. We omit the details of π-calculus for lack of space. See Milner's book [5] for details about the π-calculus.

We then express our example in a standard specification language called Scribble [6]. Scribble is a formal yet intuitive language to model workflows, with tools for specifying and reasoning about communication protocols. It is used to express π-calculus expressions, and provides tools to verify protocols. These verification tools help us with static as well as dynamic verification of workflows, which is a very important reason for specification in the first place.

Throughout the paper we work on the most common example that everyone in education know about, the exam workflow. The student and the teacher are the two roles that make this workflow. The teacher sets the paper, gives it to the student, who submits the solution to the teacher, who then gives the grade back to the student. For simplicity, we assume without loss of generality that the exam consists of just one question and one corresponding answer. We first model and specify this worfklow and show the importance of specification, and then express this process in Scribble, and show that verification, both static and dynamic can be easily done using Scribble.

II. Modelling an exam workflow

Figure 1 specifies the workflow for the one student, one teacher exam scenario. It consists of the π-calculus process definitions, followed by a π-calculus process expression. A single process definition is of the form:

$$A \stackrel{\text{def}}{=} b.\, C$$

978-0-7695-4759-6/12 $26.00 © 2012 IEEE
DOI 10.1109/T4E.2012.60

$$
\begin{array}{lcl}
\textit{s-ready} & \stackrel{\text{def}}{=} & \textit{paper}.\ \textit{s-writing} \\
\textit{s-writing} & \stackrel{\text{def}}{=} & \overline{\textit{submit}}.\ \textit{s-waiting} \\
\textit{s-waiting} & \stackrel{\text{def}}{=} & \textit{grade}.\ \textit{s-done} \\
 & & \\
\textit{t-ready} & \stackrel{\text{def}}{=} & \overline{\textit{paper}}.\ \textit{t-waiting} \\
\textit{t-waiting} & \stackrel{\text{def}}{=} & \textit{submit}.\ \textit{t-grading} \\
\textit{t-grading} & \stackrel{\text{def}}{=} & \overline{\textit{grade}}.\ \textit{t-done}
\end{array}
$$

Initial Process Expression:

$\textit{s-ready} \mid \textit{t-ready}$

Figure 1. Exam workflow: one Student & one Teacher

where A and C are states and b is the channel of transmission. A is the initial which on receiving/sending some data on the channel b, transitions to the state C. $\overline{b}$ implies that data is sent on the channel, otherwise the data is received on the channel. It is important to observe that we are only modelling the control flow on the channels, not the actual data transferred.

In this section we discuss the example of the exam workflow mentioned in the above section. We take two use cases:

A. Simple use case: one Student & one Teacher

In this case, shown in Figure 1 at any given time, the student is in one of the following possible states: ready to take the exam *s-ready*, writing the exam *s-writing*, waiting for the grade *s-waiting*, and done, *s-done*. The teacher is in one of the following states: ready with the exam, *t-ready*, distributing the exam paper and waiting for the student to submit the exam, *t-waiting*, grading the exam, *t-grading*, and done, *t-done*.

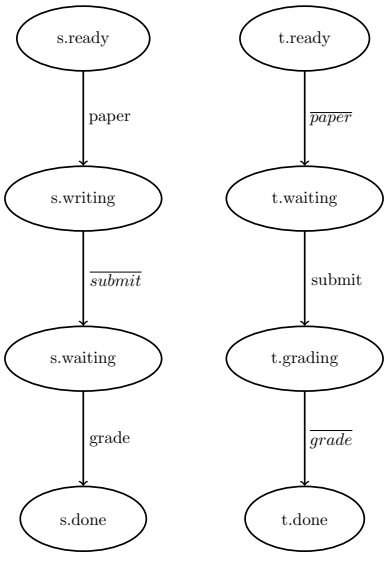

Figure 2. Graph for a Student & a Teacher participating in a one-student, one-teacher exam workflow

Figure 2 demonstrates the workflow for a student (left) and a teacher (right). The nodes of the graph are the states, and the arrows are the transition between states. The labels on the arrows are events, τ denotes a silent transition caused autonomously. Each process proceeds to the next state on a send/receive action only when there is an interaction with another process executing a complementary receive/send action respectively.

Let us consider the case of a student, who starts at state, *s-ready*, receives the paper on the channel, and transitions to state *s-writing*. On sending the submission on the channel, he/she moves to the next state, *s-waiting*, and on getting the grade (signal) on the channel, finishes at the state *s-done*. The teacher has a very similar graph, refer to Figure 2 for it. The specification of this interaction is expressed by the defining process equations and an initial process expression.

Now we show how this specification occurs algebraically.

$$
\begin{array}{c}
\textit{s-ready} \mid \textit{t-ready} \\
= \\
\textit{paper}.\ \textit{s-writing} \mid \overline{\textit{paper}}.\ \textit{t-waiting} \\
\Rightarrow \\
\textit{s-writing} \mid \textit{t-waiting} \\
\Rightarrow \\
\ldots
\end{array}
$$

B. Use case : N students & one teacher

Now consider the more common use case involving n students taking an exam. The teacher distributes n exam papers to the n students, waits for all of them to complete and submit their answers, and then returns them to each student with a grade.

$$
\begin{array}{lcl}
\textit{s}_i\textit{-ready} & \stackrel{\text{def}}{=} & \textit{paper}_i.\ \textit{s}_i\textit{-writing} \\
\textit{s}_i\textit{-writing} & \stackrel{\text{def}}{=} & \overline{\textit{submit}_i}.\ \textit{s}_i\textit{-waiting} \\
\textit{s}_i\textit{-waiting} & \stackrel{\text{def}}{=} & \textit{grade}_i.\ \textit{s}_i\textit{-done} \\
 & & \\
\textit{t-ready} & \stackrel{\text{def}}{=} & \tau.\ \textit{t-dist}(\{\}) \\
\textit{t-dist}(A) & \stackrel{\text{def}}{=} & \sum_i \overline{\textit{paper}}_i.\ \textit{t-dist}(A \cup \{i\}) \\
\textit{t-dist}(N) & \stackrel{\text{def}}{=} & \tau.\ \textit{t-waiting}(\{\}) \\
\textit{t-waiting}(A) & \stackrel{\text{def}}{=} & \sum_i \textit{submit}_i.\ \textit{t-waiting}(A \cup \{i\}) \\
\textit{t-waiting}(N) & \stackrel{\text{def}}{=} & \tau.\ \textit{t-grading}(\{\}) \\
\textit{t-grading}(A) & \stackrel{\text{def}}{=} & \sum_i \overline{\textit{grade}}_i.\ \textit{t-grading}(A \cup \{i\}) \\
\textit{t-grading}(N) & \stackrel{\text{def}}{=} & \tau.\ \textit{t-done}
\end{array}
$$

Initial Process Expression:

$\textit{s}_1\textit{-ready} \mid \ \ldots\ \mid \textit{s}_n\textit{-ready} \mid \textit{t-ready}$

Figure 3. Exam workflow: n Students & one Teacher

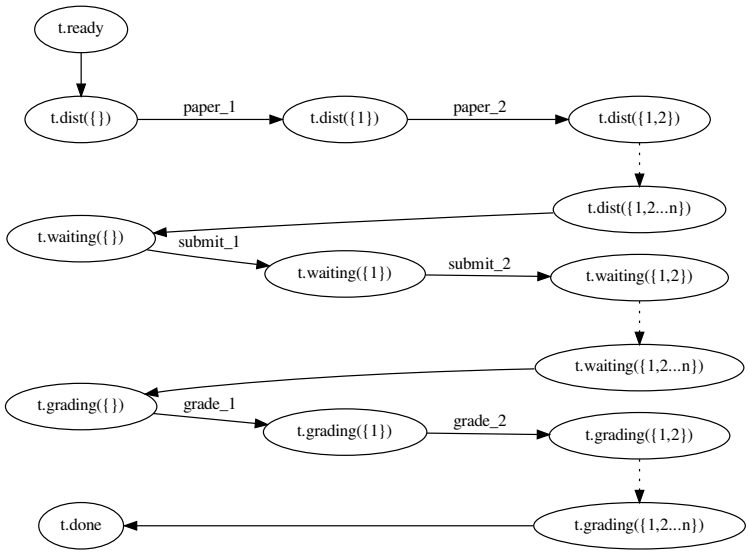

Figure 4. An attempt at constructing a Graph for a teacher participating in a n-student, one-teacher exam workflow

Figure 3 specifies the workflow for the n-student, one teacher exam scenario. At any given time, the ith student is in one of the following possible states: ready to take the exam *s_i-ready*, writing the exam *s_i-writing*, waiting for the grade *s_i-waiting*, and done, *s_i-done*. The teacher is in one of the following states: ready with the exam, *t-ready*, distributing the exam to the students, *t-dist*, waiting for all the students to submit the exam, *t-waiting*, grading, *t-grading*, and *t-done*. The state *t-dist* is parametrized by the parameter 'A' that denotes the set of students who have already received the question paper, *t-waiting* by the set of papers already submitted, and *t-grading* by the set of papers already graded.

The teacher starts in the state, *t-ready*, transitions to the state *t-dist* with an empty parameter list, sends the paper to each student (in any order) on the channel, and reaches the state *t-dist* parametrized with a subset A of the set {1,2...n} abbreviated N. On the distribution of the papers, the teacher transitions into the *t-waiting* state again with an empty parameter list. On receiving the submissions from all students on the channel (again in any order), it moves to the *t-waiting* parametrized by the set {1,2...n}. Next, on receiving all the grades it transitions to the *t-grading* state again with no parameters. All the grades are then sent out to the students (in any order), and on reaching the *t-grading* state with the parameter as the set {1,2...n}, the teacher finally transitions into the *t-done* state and exits. We can see that because of any number of random orderings possible, it is very difficult to represent the flow of states in a single graph.

In Figure 4, we make an attempt to show the graph for the workflow of a teacher in this case, where there are n students taking the exam. The graph for a student in this case isn't shown, as it is similar to the one in Figure 2. The nodes of the graph shown refer to the parametrized states. This graph assumes that the ordering in which the exam is distributed,collected & graded is {1,2...n}. However this is only one case, we can calculate that there will be O(n!) such cases.

C. Discussion on the exam workflow example

With this example, we can see how one can model and algebraically specify workflows which in this case was an exam workflow. This was a simple example, but such a method of formalization can be extended to large educational process as well. Intuitively, from this example it seems that all possible educational processes can be expressed by these ideas of "flow of work".

We also observe that graphical notation of modelling can't handle complex workflows, in turn proving the power of an algebraic notation such as π-calculus.

We see that any system built on this algebraic specification will essentially be just an implementation which follows this algebraically specified workflow, taking action based on the current state and the information on the communication channel. This is a big advantage of expressing workflows in an algebraic way. It also allows such specifications to be used to statically and dynamically validate the protocol the system is following. This is shown in the next section.

III. Scribble: Specification language based on π-calculus

In the previous section, we defined the workflow associated with conducting an exam. In this section, we show how we can specify the exam workflow in Scribble [6]. Scribble is a specification language that provides its own protocol validator to do runtime monitoring of implementations, as well as a static protocol type checker.

Listing 1 shows us the simple example workflow of one student and one teacher.

The code is self-explanatory, we define a protocol PaperChecker which takes as in input, two roles, a student and a teacher. The paper is sent on the channel from the teacher to the student. Then the submission is sent back to the teacher, and finally the grade is sent from the teacher to the student.

The power of Scribble comes because any implementation of the workflow (in java) we write can be verified statically as well as dynamically via the protocol type checker and protocol validator respectively.

```
import Paper;
import Grade;
import Submit;

protocol PaperChecker ( role Teacher, role
    Student ) {

        Paper from Teacher to Student;
        Submit from Student to Teacher;
        Grade from Teacher to Student;
        }
}
```

Listing 1. Scribble Protocol Specification for exam workflow

IV. Related Work

Business Process Execution Language (BPEL) [7] is a language which allows interactions with web services by using interfaces. Web services are modular functions that operate independently to accomplish business tasks. BPEL is used to model business processes by composing a workflow through an orchestration of web services.

A business process modeled via BPEL has its initial actions described in it. Once an orchestration is defined and the business process is deployed, it is not possible to add user behaviour to the business process.

One of the many issues related to workflow specification is the reachability of a state. Lyng et al. have designed a monitoring system [8] to find out whether the execution can reach an invalid state in clinical workflows. In addition to the satisfiability and reachability checks with a model checker like Spin [9], they developed a flow monitor to notify if the system reaches an invalid state during a run.

Petri nets are widely allocated for modelling workflows [10], as they feature a state-based approach towards modelling. They have good analysis techniques available. However, they are cases in which Petri nets are not suitable for modelling reactive systems, which most educational process are.

Orc is an orchestration language for achieving distributed and concurrent operations by providing uniform access to computational services through sites [11]. Sites in orc are analogous to channels or names in pi-calculus. Orc is developed to address communication of distributed sites, where as a formal model written in pi-calculus does not limit or assume the location of the processes running.

V. Conclusion

We have proposed a way to model and specify educational workflows.We have used a well known notation, the π-calculus that allows workflows to be expressed algebraically, and should have good verification tools available. We have shown why specification of workflows is such a powerful tool for the design of technology for educational processes.

As future work, we are in the process of building a system, as an extension of the Pantoto system [2], which takes this workflow specification as the input, and gives us a system which "interprets" the workflow, and runs the system as per the workflow.

References

[1] S. Mukherjee, H. Davulcu, M. Kifer, P. Senkul, and G. Yang, "Logic based approaches to workflow modeling and verification," 2003. [Online]. Available: http://www.scientificcommons.org/42916013

[2] S. Uskudarli and T. Dinesh., "Pantoto: A model for managing communities in the context of semantic web." in *ICSD: International Conference on Semantic Web & Digital Libraries*, February 2007.

[3] J. Eder, E. Panagos, and M. Rabinovich, "Time constraints in workflow systems," in *Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, ser. CAiSE '99. London, UK, UK: Springer-Verlag, 1999, pp. 286–300.

[4] F. Puhlmann and M. Weske, "Using the pi-calculus for formalizing workflow patterns," in *In Proceedings 3rd International Conference on Business Process Management (BPM 2005)*. Springer Verlag, 2005, pp. 153–168.

[5] R. Milner, *A of Mobile Processes*. Cambridge University Press, 1992.

[6] K. Honda, A. Mukhamedov, G. Brown, T.-C. Chen, and N. Yoshida, "Scribbling interactions with a formal foundation," in *In ICDCIT, LNCS*. Springer, 2011.

[7] F. Curbera, "Composed services and WS-BPEL," in *Encyclopedia of Database Systems*, L. LIU and M. T. ZSU, Eds. Springer US, 2009, pp. 413–418.

[8] K. M. Lyng, T. Hildebrandt, and R. R. Mukkamala, "From paper based clinical practice guidelines to declarative workflow management," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, D. Ardagna, M. Mecella, and J. Yang, Eds. Springer Berlin Heidelberg, 2009, vol. 17, pp. 336–347.

[9] G. Holzmann, "The model checker spin," *Software Engineering, IEEE Transactions on*, vol. 23, no. 5, pp. 279 –295, may 1997.

[10] R. Eshuis and J. Dehnert, "Reactive petri nets for workflow modeling," in *Application and Theory of Petri Nets 2003*. Springer, 2003, pp. 296–315.

[11] D. Kitchin, A. Quark, W. Cook, and J. Misra, "The orc programming language," in *Formal Techniques for Distributed Systems*, ser. Lecture Notes in Computer Science, D. Lee, A. Lopes, and A. Poetzsch-Heffter, Eds. Springer Berlin / Heidelberg, 2009, vol. 5522, pp. 1–25.