Communication based workflow loop formalization using Temporal Logic of Actions (TLA)

José L. Caro[1], Antonio Guevara[1], Andrés Aguayo[1], and José L. Leiva[1]

[1] Computer Science Department, Malaga University, 29071 Málaga, Spain

Abstract. The workflow map development for an organization is a highly complex process. Therefore, the workflow map should be tested and validated before it is implemented (into a WfMS). Most current workflow systems deal with this validation issue by using simulation modules that "execute" the model and examine the possible problems before it is truly "executed" and implemented. Although these simulation modules are very useful for the management team to detect problems in the business processes represented by the workflow, it would be advisable to find other more reliable methods. In this paper we propose a formal method based on Temporal Logic of Actions to formalize workflow maps (based on a communication modelling methodology).

1 Introduction

Actually there are two main research areas related to workflow technology [6]: (i) workflow management systems (WfMS) (the implementation of workflow technology), and (ii) workflow modelling techniques (the capability to express the process, work, information, methods, etc. for a organization into a specification called worklow map).

A workflow map could describe business a process at the conceptual level needed for the evaluation, understanding and design of such business processes, as well as capturing information process tasks at level that describes such process requirements for information systems and human skills [2]. This model should allow and facilitate the automated demonstration of properties. For example: will any workflow never be executed? Will this workflow ever be executed? Is the operation carried out with a specified time cost? Formal proving mechanisms will provide a practical solution to these kinds of problems [3].

In this paper, our objective is to develop a language/action paradigm formalization [5], based on a extension of temporal logic, known as Temporal Logic of Actions (TLA) [4]. This paper is organized as follows: we begin with a brief description of workflow (sec. 2). In section 3, the TLA elements needed for the formalization are described. In section 4, we develop the TLA formalization of the language/action paradigm. The last section summarizes the paper conclusions.

2 Workflow

Workflow has a wide range of possibilities related to group support and the automation of organizational processes. In general terms we can define workflow as [9]: "workflow

L. Caro J., Guevara A., Aguayo A. and L. Leiva J. (2004).
Communication based workflow loop formalization using Temporal Logic of Actions (TLA).
In *Proceedings of the 1st International Workshop on Computer Supported Activity Coordination*, pages 233-238
DOI: 10.5220/0002670002330238
Copyright © SciTePress

is comprised by a set of activities dealing with the coordinated execution of multiple tasks developed by different processing entities in order to reach a common objective". A good approach to workflow modelling techniques, that includes a formal method for achieve demonstrations, can be found in [1, 7, 10, 6, 2].

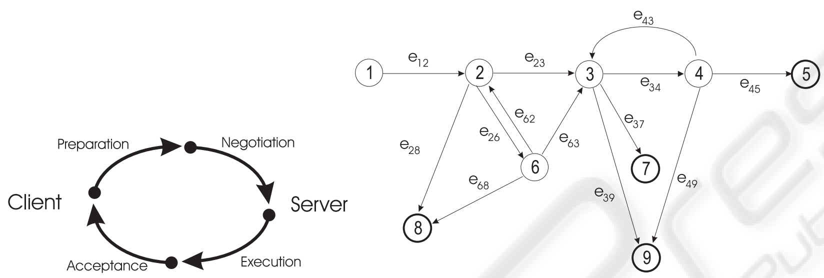

Fig. 1. Workflow loop & state diagram

Communication-based methodologies stem from the "Conversation for Action" model developed by Medina-Mora, Winograd, and Flores [5]. The communication between client (Cli) and server (Svr) can be defined in four steps (figure 1): *request/preparation*, *negotiation*, *development* and *acceptance*. The language/action perspective modelling is based on the "speech-act" theory developed by John Searle [8] that has been adapted for workflow management (statechart).

The diagram starts at state 1, opening the conversation (by Cli), that triggers the transition to state 2, where the server Svr has three options: *promise*, perform the work (state 3); *refuse*, closes the conversation (state 8); *counteroffer*, negotiation of satisfaction terms (state 6). The simplest path is going through the following transitions: *promise*, the task is accepted (from state 2 to 3); *report*, the accepted task is done (from state 3 to 4); *declare*, the service is accepted (from 4 to 5). Other option is if Svr initiates a counteroffer (from state 2 to 6), Cli has three options: *accept*, state 3; *counteroffer*, back to state 2; *decline*, the service is refused (state 8). From state 3 there are this additional options *renege*, not performing (from 3 to 7) and *withdraw* the request (state 3 to 9). After Svr reporting that the work is concluded the possible actions are: *declare*, the work has not been concluded satisfactorily and Svr has to do it again (from state 4 to 3) and *withdraw* the petition (from state 4 to 9).

3 Temporal Logic of Actions

Temporal Logic of Actions (TLA) combines two logics the action logic and temporal logic [4]. All TLA formulas are TRUE or FALSE in a behavior σ defined as a infinite sequence of states $< s_0, s_1, s_2, \ldots >$.

3.1 Elements of State Logic in TLA

– *Variables*. The semantic of $[[x]](s)$ is the function of the value of x in the state s.

– *State and predicate functions*. The semantics of $[[f]]$ is a mapping of states into values.
– *Actions*. We obtain $[[A]](s,t)$, by first replacing each variable x with $[[x]](s)$ and each variable x' with $[[x]](t)$ to later evaluate the expression. It is said that the state pair (s,t) is a "A-step" iff $[[A]]$(s,t) is TRUE.
– *Active action in a state*. An action A is said to be active in a state s if there is a state t such that (s,t) is a A-step: $[[Enabled\ A]](s) \doteq \exists t \in \sigma : [[A]](s,t)$.

3.2 Elements of Temporal Logic in TLA

In TLA, the system behavior is modelled as an infinite sequence of states. The basic elements are actions and temporal logic:

– *Predicates*. A behavior satisfies the predicate P iff (*if and only if*) is satisfied in the first state: $[[P]](< s_0, s_1, s_2, \dots >) \Rightarrow [[P]](s_0)$. Similarly, a behavior satisfies the action A iff the first pair of states of the given behavior is an A-step: $[[A]](< s_0, s_1, s_2, \dots >) \Rightarrow [[A]](s_0, s_1)$.
– *"Always" ($\Box$) operator*. Given a formula F, $\Box F$ asserts that F is always TRUE: $[[\Box F]](< s_0, s_1, s_2, \dots >) \doteq \forall n \geq 0 : [[F]](< s_n, s_{n+1}, s_{n+2}, \dots >)$.
– *"Eventually" ($\Diamond F$) operator*. The formula $\Diamond F$ asserts that F is eventually TRUE: $[[\Diamond F]](< s_0, s_1, s_2, \dots >) \doteq \exists n \geq 0 : [[F]](< s_n, s_{n+1}, s_{n+2}, \dots >)$.
– *Validity*. A formulae F is valid iff it is satisfied for all behaviors ($\models F \doteq \forall \sigma \in S^\infty : [[F]](\sigma)$), where S^∞ denotes the set of all possible behaviors: $\models F \doteq \forall \sigma \in S^\infty : [[F]](\sigma)$.
– *Specification in TLA*. A formal specification has the following general formula: $\Pi \doteq Init \wedge \Box(A_1 \vee A_2 \vee \dots \vee A_n)$.
– *Fairness Operators*. The fairness operators are in charge of ensuring that "nothing abnormal will happen":
  • *Weak fairness* (WF). Asserts that an action has to be infinitely executed frequently if it is continuously enabled for an infinitely long time: $WF_v(A) \doteq \Box\Diamond\langle A\rangle_v \vee \Box\Diamond\neg Enabled\langle A\rangle_v$.
  • *Strong fairness* (SF). Asserts that an action has to be infinitely executed frequently if it is often infinitely enabled: $SF_v(A) \doteq \Box\Diamond\langle A\rangle_v \vee \Diamond\Box\neg Enabled\langle A\rangle_v$.
– *Formal system specification*. A formal specification of a system, within fairness conditions, can be represented as: $\Pi \doteq Init \wedge \Box[N]_v \wedge L$.

3.3 Formal approach to state diagram modelling

Δ formula represents the statechart (equation 1).

$$\begin{aligned} Init_\Delta &\doteq \exists n \in I : P_n \\ \mathcal{A}_n &\doteq \exists e \in E(n) : \varepsilon_e \wedge P'_{d(e)} \\ \Delta &\doteq Init_\Delta \wedge \forall n \in N : \Box[P_n \Rightarrow \mathcal{A}_n]_v \end{aligned} \qquad (1)$$

Where, N: set of nodes; I: set of initial nodes; $E(n)$: set of edges originating at node n; $d(e)$: destination node of edge e; P_n: predicate labelling node n; ε_e: action labelling edge n.

4 Formalizing the language/action paradigm

In order to formalize in TLA the state diagram its edges are labelled e_{ij} as shown in figure 1. Work W_k is a quadruple expressed in the equation $W_k \doteq \{I, H, P, SC, V\}$ where: $W_k.I$: information; $W_k.H$: tools and methods; $W_k.P$: human, role or agent to develop the task; $W_k.SC$: terms of work satisfaction; $W_k.V$: set of state variables belonging to the workflow. The set of nodes in our diagram will be denoted by N_i, where i is the number of the current state (initial state N_1). In this way, and bearing in mind the equation to be satisfied in the initial state, we have $Init_\Delta \doteq \exists n \in I : P_n$. Therefore, to complete the definition of the initial state we have to give meaning to P_{N_1} (equation 2).

P_{N_1} indicates that the workflow will start when an event coming from agent A is triggered ($DetectTrigger$). The agent A corresponds to the client Cli, and the event is a request for service identified as IDWF and corresponding to the workflow. The next state variables belonging to the set V are defined as follows (we will omit the V): W_kS: work state, W_fS: current state, W_fP: current phase, W_k: current work, XW_k: external work proposed by A, W_fCli: client and W_fSvr: server. Once the workflow has started the system transits to N_2 through edge e_{12} (equation 3).

$$
\begin{aligned}
P_{N_1} \doteq\ & DetectTrigger(Type, Origin, Workflow) \\
& \wedge Workflow = IDWF \wedge Type \in T_{WFID} \wedge Origin = A \qquad (2) \\
e_{12} \doteq\ & W_k' = XW_k \wedge W_fP' = \text{“}prep.\text{”} \wedge\ W_fS' = \text{“}active\text{”} \wedge\ W_kS' = \text{“}study\text{”} \\
& \wedge W_fCli' = A \wedge W_fSvr' = SelAgent(OrgDB, B) \qquad (3)
\end{aligned}
$$

The workflow IDWF is instantiated and the requested external work is assigned to the execution model that includes work XW_k, that will be assigned by the function *SelAgent*(OrgDB,B) (usign the organizational knowledge database OrgDB) and the initial terms of satisfaction $XW_k.SC$. In the state N_2, the work has to be evaluated. The petition and the result of such evaluation is resented by ***EvalWk***(W_k,Agent), where W_k is the work to be evaluated W_k. The response is obtained from ***CounterOffer*** is possible to return: “commitment”, “counteroffer” (and the consecuent new W_k), “decline” or “cancel” (eq. 4). From state N_2 we can advance through edges e_{22}, e_{26} and e_{28}. Edge e_{28} leads to state N_8, which is a final state of uncompleted termination (eq. 5). Consequently, a predicate that will abort the execution of the workflow will be used in N_8. This will trigger an exception so that the system can take appropriate actions, therefore, $P_{N_8} \doteq Exception(\text{IDWF}, \text{“abort”}, W_k)$. The function ***Exception*** triggers the exception of aborting the task, and goes back to TRUE when completed. The edge e_{26} corresponds to a counteroffer from B after the evaluation done in P_{N_2} (eq. 6).

$$
\begin{aligned}
P_{N_2} \doteq\ & S = EvalWk(W_k, B) \wedge (S = \text{“commit.”} \vee\ S = \text{“c.off.”} \vee\ S = \text{“dec.”}) \qquad (4) \\
e_{28} \doteq\ & S = \text{“decl.”} \wedge W_fP' = \text{“prep.”} \wedge\ W_fS' = \text{“abort”} \wedge\ W_kS' = \text{“Svr}_{\text{ab.}}\text{”} \qquad (5) \\
e_{26} \doteq\ & S = \text{“c.offer”} \wedge W_fP' = \text{“neg.”} \wedge\ W_fS' = \text{“act.”} \wedge\ W_kS' = \text{“neg.”} \\
& \wedge W_k{}' = \text{CounterOffer}(W_k, W_f.Svr) \qquad (6)
\end{aligned}
$$

In N_6 another evaluation is carried out by A (equation 7). We have three options: reconsidering the offer i.e., transit to N_2 (eq. 8), refuse the offer and raise an exception (transit to N_8, eq. 9).

$$\begin{aligned} P_{N_6} &\doteq S = \text{EvalWk}(W_k, W_f.Cli) \wedge (S = \text{“ac.”} \vee S = \text{“c.off.”} \vee S = \text{“decl.”}) && (7)\\ e_{62} &\doteq S = \text{“c.off.”} \wedge W_f P' = \text{“neg.”} \wedge W_f S' = \text{“act.”} \\ &\quad \wedge W_k S' = \text{“neg.”} \wedge W_k{}' = \text{ConunterOffer}(W_k, W_f.Cli) && (8)\\ e_{68} &\doteq S = \text{“dec.”} \wedge W_f P' = \text{“neg.”} \wedge W_f S' = \text{“abort”} \wedge W_k S' = \text{“Cli}_{\text{refuses}}\text{”} && (9) \end{aligned}$$

The third option is to accept the work W_k and transit to N_3 (eq. 10). Similarly, if the evaluation leads to B accepting the work, we could directly transit from N_2 to N_3 (eq. 11).

$$\begin{aligned} e_{63} &\doteq S = \text{“acc.”} \wedge W_f P' = \text{“devel.”} \wedge W_f S' = \text{“exec.”} \wedge W_k S' = \text{“acc.”} && (10)\\ e_{23} &\doteq S = \text{“acc.”} \wedge W_f P' = \text{“devel.”} \wedge W_f S' = \text{“exec.”} \wedge W_k S' = \text{“acc.”} && (11) \end{aligned}$$

To define the semantic of P_{N_3} we need the following functions: Trigger(t_i,W_f): trigger t_i to enable the sub-workflow W_f; Completed(W_f): TRUE if W_f has been satisfactorily completed; Aborted(W_f): TRUE if W_f has terminated as abort; Abort(X): TRUE if X aborts the workflow. In P_{N_3} all workflow subtasks must to be executed. The following case has to take place: a) all subtasks a_i have to be completed; b) the incorrect termination of some subtasks has to be detected; or c) the client aborts the workflow. Let $W_k.a_i$ each of the subtasks comprising the task of W_k, then P_{N_3} is defined as the equations 12 and 13:

$$\begin{aligned} P_{N_3} &\doteq \forall (a_i \in W_k, t_i / t_i = Trg(a_i)) Trigger(t_i, W_k.a_i) \wedge (ExecP) && (12)\\ ExecP &\doteq \Box \forall a_i \in W_k / Completed(W_k.a_i) \\ &\quad \vee \Box \exists a_i \in W_k / Aborted(W_k.a_i) \vee Abort(W_k.Cli) && (13) \end{aligned}$$

At this point two options are possible: aborting the execution and transit to state N_7 or N_9 depending on who aborted or transit to evaluation state N_4 (eq. 14, 15, 16 and 17).

$$\begin{aligned} e_{39} &\doteq \exists a_i \in W_k / (W_k.a_i.W_f S = \text{“aborted”} \wedge W_k.a_i.W_k S = \text{“Svr ref. Wk”}) \\ &\quad \wedge (W_f P' = \text{“exec.”} \wedge W_f S' = \text{“aborted”} \wedge W_k S' = \text{“Svr}_{\text{refuses}}\text{”}) && (14)\\ e_{37} &\doteq \exists a_i \in W_k / (W_k.a_i.W_f S = \text{“aborted”} \wedge W_k.a_i.W_k S = \text{“Cli}_{\text{refuses}}\text{”}) \\ &\quad \wedge (W_f P' = \text{“exec.”} \wedge W_f S' = \text{“aborted”} \wedge W_k S' = \text{“Cli}_{\text{refuses}}) && (15)\\ e_{34} &\doteq \forall a_i \in W_k / (W_k.a_i.W_f S = \text{“accepted”} \wedge W_k.a_i.W_k S = \text{“completed”}) \\ &\quad \wedge (W_f P' = \text{“eval.”} \wedge W_f S' = \text{“active”} \wedge W_k S' = \text{“Cli's}_{\text{eval.}}\text{”}) && (16)\\ P_{N_4} &\doteq EvalReport(W_k, B) = \text{“correct”} \vee EvalReport(W_k, B) = \text{“incorrect”} \\ &\quad \vee Exception(\text{IDWF}, \text{“abort”}, W_k) && (17) \end{aligned}$$

If the work satisfies the terms, there is a transition to successful state N_5. On the other hand, if it does not, we go back to the subtask execution state (this path is optional) and if it is aborted, it leads to N_9 (eq. 18, 19, and 20).

$$e_{49} \doteq W_f P' = \text{“evaluation”} \wedge W_f S' = \text{“aborted”} \wedge W_k S' = \text{“Cli}_{\text{rejects}}\text{”} \quad (18)$$
$$e_{45} \doteq W_f P' = \text{“complete”} \wedge W_f S' = \text{“completed”} \wedge W_k S' = \text{“Wk}_{\text{accept}}\text{”} \quad (19)$$
$$e_{43} \doteq W_f P' = \text{“execution”} \wedge W_f S' = \text{“active”} \wedge W_k S' = \text{“reexecute”} \quad (20)$$

State N_5 only has to send a completed signal: $P_{N_5} \doteq Signal(\text{IDWF}, \text{“WF completed”}, W_k)$. Once these definitions are completed and the model equations are applied to formalize a state diagram (eq. 1) we obtain the formal representation.

5 Conclusions

The use of formal methods based on logic in workflow modelling can establish an automated, formal, and robust reasoning mechanism that will successfully provide insight into these issues (conflict, deadlock, reacheability, reliability and satisfability). The application of TLA to workflow management systems provides three fundamental bases [3]: (i) *theory*: providing a theory with a valid and robust basis to carry out analysis; (ii) *formalization*: expressing workflow maps as TLA expressions; and (iii) *analysis*: providing a mechanism for the automated demonstration of workflow model properties.

References

1. W. Aalst. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. *Lecture Notes in Computer Science*, 1806:161–183, 2000.
2. J. L. Caro, A. Guevara, and A. Aguayo. Workflow: a solution for the cooperative development of an information system. *Business Process Management Journal*, 9(2):208–220, 2003.
3. J. L. Caro, A. Guevara, A. Aguayo, S. Gálvez, and A. Carrillo. A temporal reasoning approach of communication based workflow modelling. In O. Camp, J. Filipe, S. Hammoudi, and M. Piattini, editors, *ICEIS'2003. Internaciontal Conference on Enterprise Information Systems*, pages 245–250, Angers, France, 2003. Ecola Superior de Setubal, ACM, IEEE.
4. L. Lamport. *Specifying Systems*. Addison–Wesley, 2002.
5. R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The Action Workflow Approach to Workflow Management Technology. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, Emerging Technologies for Cooperative Work, pages 281–288, 1992.
6. H. Reijers. *Design and Control of Workflow Processes*, volume LNCS-2617 of *Lecture Notes in Computer Science*. Springer, 2003.
7. W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 21(2):117–134, April 2000.
8. J. R. Searle. A taxonomy of illocutionary acts. In K. Gunderson, editor, *Language, Mind, and Knowledge. Minnesota Studies in the Philosophy of Science, Vol. 7*, pages 344–369. University of Minnesota Press, Minneapolis, Minnesota, 1975.
9. A. Sheth and M. Rusinkiewicz. On transactional workflows. *IEEE Data Engineering Bulletin*, 16(2):37, June 1993.
10. H. Zhuge, T. yat Cheung, an d H. keng Pung. A timed workflow process model. *Journal of Systems and Software*, 55(3):231–243, January 2001.