ORB ACADEMIC PUBLISHER

Int. J. Comput. Sci. Theor. App., 2014, vol. 1, no. 1., p. 1-11.
Available online at www.orb-academic.org

International Journal of Computer Science: Theory and Application
ISSN: 2336-0984

Proposing a Formal Method for Workflow Modelling: Temporal Logic of Actions (TLA)

Jose L. Caro[1]

[1] *Computer Science Department, University of Malaga, Malaga, Spain*
Email: jlcaro@uma.es

ABSTRACT
The study and implementation of formal techniques to aid the design and implementation of Workflow Management Systems (WfMS) is still required. Using these techniques, we can provide this technology with automated reasoning capacities, which are required for the automated demonstration of the properties that will verify a given model. This paper develops a formalization of the workflow paradigm based on communication (speech-act theory) by using a temporal logic, namely, the Temporal Logic of Actions (TLA). This formalization provides the basic theoretical foundation for the automated demonstration of the properties of a workflow map, its simulation, and fine-tuning by managers.

KEYWORDS
Workflow — Workflow Management Systems — Temporal Logic — Temporal Logic of Actions.
© *2014 by Orb Academic Publisher. All rights reserved.*

1. Introduction

Workflow concept has been received with great interest in the business world and in the area of software development. Workflow technology and Workflow Management Systems (WfMS) are based on several disciplines: CSCW (Computer Supported Cooperative Work) and OIS (Office Information Systems) [1] are the main topics. Workflow includes a set of technological solutions that allow us to automate work processes previously described by a formal model (called workflow map). The modelling of business processes into a workflow map is aimed at obtaining total automation and optimization of such processes.

Business processes reengineering, work simulation, organization modelling, resource management, and work automation are some aspects under the general issue of what is nowadays known as workflow technology [2].

The development of a workflow management system for an organization is a highly complex process. Therefore, the workflow map should be tested and validated before it is implemented; in other words, it should be analyzed prior to implementation. 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 in real life [3].

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 other words, the model should allow and facilitate the automated demonstration of properties and characteristics. 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 [4], [5].

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, satisfability) [6], [7].

However, the efficiency of visual modelling tools should be preserved and the introduction of new technology avoided. To achieve this we have to establish a direct and unambiguous relationship between current workflow paradigms and temporal logic. In this way, and depending on the particular case, we will be able to use one or the other representational model: the visual/graphic model for design, and the formal model for automated handling and analysis [8], [9].

In this paper we aim at approaching workflow modelling in a different way. Our object is to make a formalization of the language/action paradigm [10], based on a extension of temporal logic. This extension is known as Temporal Logic of Actions (TLA) [11], and allows the easy modelling of transition states.

Our approach is as follows:

1. Specification of the workflow loop semantics (section 2.2).
2. Translation of the workflow loop into a state transition diagram (section 2.4.1).
3. The formalization in TLA of Searle's state transition diagram (section 4).

4. The formalization in TLA of workflow conections (section 5).

This paper is organized as follows: we begin with a description of workflow, workflow management systems, and the modelling of workflow processes (sec. 2). In section 2.2 we analyze the basis of communication-based methodology ("speech-act"). In section 3 the TLA elements needed for the formalization are described. The core of this paper is (section 4 and 5), where the TLA formalization of the language/action paradigm is developed. The last section includes some relevant conclusions and future work.

2. Workflow and WfMS

Workflow includes a set of technological solutions aimed at automating work processes that are described in an explicit process model called the workflow map. Workflow has a wide range of possibilities as demonstrated by group support and the automation of organizational processes.

In general terms we can define workflow as [7]:

> workflow 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

This definition of workflow does not indicate the nature of the processing entity, which, therefore, can be a person, a computer, a machine, etc. [12], [13].

This technology is made tangible as information technology systems in the form of workflow management systems (WfMS). WfMS can be defined as [14]:

> "A system that defines, creates, and manages automatically the execution of workflow models by the use of one or more workflow engines in charge of interpreting process definitions (workflow maps), interacting with agents and, when required, invoking the use of information systems involved in the work"

The workflow engine is in charge of coordinating the execution of the workflow model, by determining the agents involved (whether humans or not), the data, and the applications required to carry out the workflow. The WfMS is made up of many modules, but this paper pays special attention to the simulation module which is used to test, via simulation, the workflow map before it is really implemented. Our aim is to propose a new reasoning workflow module that is able to analyze the model before its implementation. This reasoning procedure will establish the consistency of the model by demonstrating the properties it should satisfy [15].

To achieve this objective, the workflow map has to be expressed in a way that allows such demonstrations. For this reason we will focus on demonstrating the possibility of translating workflow technology into a logic tool.

2.1 Modelling techniques for workflow processes

Many authors agree on splitting workflow methodologies into two main categories [2]:

- Activity-based methodology. These focus on modelling the activities that will take place during the development of the workflow [14].
- Communication-based methodologies. These stem from Searle's theory, known as "speech-acts" [16], [17].

Other techniques, like Petri Nets, Trigger Modelling, etc can be found in the literature [18]-[23], although other authors englobe this techniques in the activity based group.

As an example for good approach to time modelled, formal methods, Petri nets and in general modelling techniques that includes a formal method for achieve demonstrations can be found in [24]-[27] but these methods are centered in activity bases methodologies. Also we can introduce a new approach in the form of temporal logic formal methods into workflow technologies.

As our case study falls into the communication-based category, to demonstrate that the formal methods can be applied. We will now describe the basic principles underlying this methodology.

2.2 Communication-based methodologies

Communication-based methodologies stem from the "Conversation for Action" model developed by Medina-Mora, Winograd, and Flores [10], [28]. They view workflow as a sequence of conversations between a client and a server. In this section, the agents involved are described as the client requiring a service that will be developed or performed by the server .

The communication previously described between client (*Cli*) and server (*Svr*) can be defined in four steps (figure 1):

- Request/preparation. The client requests an action and establishes the criteria for completing it successfully.
- Negotiation. The conditions for being satisfied with the work to be done are negotiated.
- Development. The action is carried out by the server.
- Acceptance. The workflow loop is finished by accepting the work under the terms of satisfaction established in the second step.

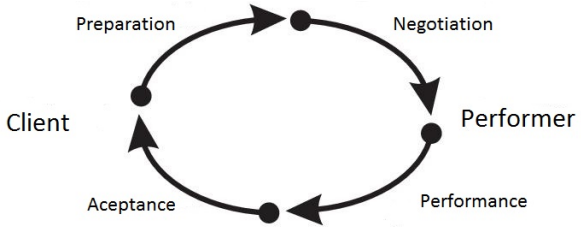

Figure 1. *Workflow loop*

This model has two behaviors:

- internal behavior or micro–level: the workflow individual behavior.
- external behavior or macro–level : the relations inter workflows.

2.3 External behavior. Workflow map

A workflow map is the overall representation of all process into a organization.

Each stage or step intro a workflow can be broken down into several workflows which will help to make them more specific. The set of subdivisions within the workflow loops is known as a Business Process Map (BPM).

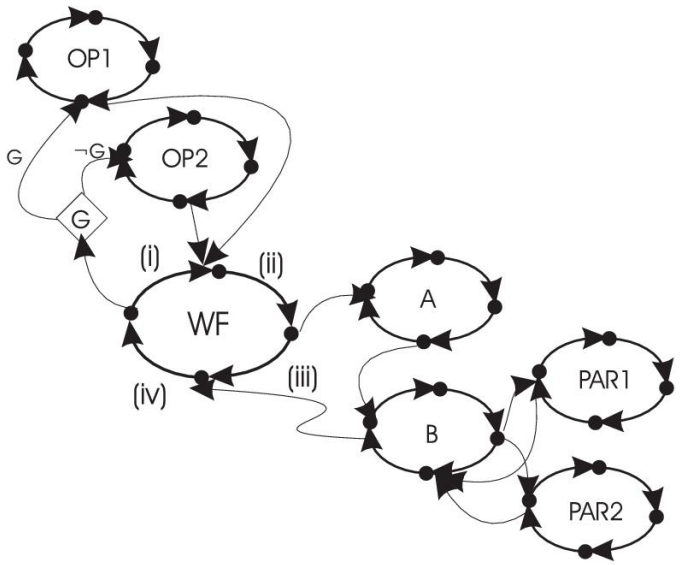

Figure 2. *Example model*

At any phase we can connect workflows in three modes (see figure 2): (a) Sequential mode: like *A* and *B* workflows. (b) Conditional mode: like *OP*1 and *OP*2 that can be executed depending on the guard *G*. (c) Parallel model: like *PAR*1 and *PAR*2.

The workflow can be terminated at any time without being completed, which will raise an exception state where the task is not successfully finished.

2.4 Internal behavior. Workflow loop and Searle's state transition diagram

The language/action perspective on modelling is based on the "speech-act" theory outlined by Austin [29] and further developed by John Searle [16], [17]. Speech-acts occurring between two agents to carry out a given task are called "conversations" and they are the core framework for developing and performing the work.

2.4.1 Conversation for action

Conversation for action is the foundation of the theory upon which the workflow modelling studied in 2.2 is based. There are two speakers in this kind of conversation: one takes the role of client (*Cli*) requesting a service and the other the role of the server (*Svr*) that will carry out the task.

In order to carry out the action, a sequence of request and commitment acts are established that will coordinate the action. The state diagram in figure 3 shows all possible transitions in a conversation for action. This diagrams corresponds with the representation at intra–workflow level. This level is how internally works an individual workflow.

The original theory establishes two different possibilities for initiating the action: offering it or requesting it.

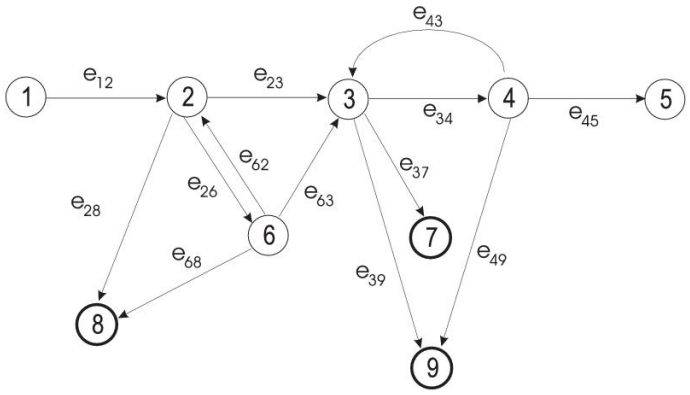

Figure 3. *State diagram of a workflow loop*

The diagram starts at state 1, opening the conversation, where *Cli* makes an initial request. This states triggers the transition to state 2, where the server *Svr* has three options:

- Promise: the server commits itself to perform the work (state 3).
- Refuse it/Decline: closes the conversation without performing the work and transits to the final state 8.
- Counteroffer: the terms for performing the work are negotiated (transition to state 6).

If *Svr* initiates a counteroffer (transition from state 2 to 6), *Cli* has three options:

- Accept the terms: Accept the counteroffer and initiate the work (transition to state 3, i.e., carrying out the work).
- Counteroffer: New terms of satisfaction are proposed (back to state 2, work evaluation).
- Refusing/Decline: The service is refused and we transit to the final state 8.

The simplest path from the moment the task has been accepted is to successfully conclude it; i.e., going through the following transitions:

- Petition: the action transits from 1 to 2.
- Promise: the task is accepted (transition from 2 to 3).
- Report: the accepted task is done (transition from 3 to 4).

- Declare: if the product or service satisfies the clients' expectations, there is a transition from 4 to 5.

Another path involves transitions requiring negotiation, i.e., 3 and 4 transitions. From state 3:

- Renege: Not performing the task accepted (from 3 to 7).
- Withdraw: *Cli* automatically withdraws the request (state 3 to 9).

After *Svr* reporting that the work is concluded, several actions are still possible:

- Declare: *Cli* declares the work has not been concluded satisfactorily, and *Svr* has to do it again (stage 4 to 3).
- Withdraw: *Cli* automatically withdraws the petition (from state 4 to 9).

This complex structure makes it possible for a system to computationally coordinate a task.

3. Temporal Logic of Action

Since the first temporal logic was proposed by Pnueli [6] many variants have been developed. In this paper, we make use of Temporal Logic of Actions (TLA) which allows us to model state transition diagrams in a relatively easy manner. Therefore, we now describe the basic principles described by Lamport [11] which are required to understand our work.

TLA combines two types of logic: Action logic, used to represent relationships between states, and temporal logic, dealing with the reasoning involved in an infinite sequence of states.

All TLA formulas are TRUE or FALSE in a behavior ($\doteq$ denotes equal to by definition). We define behavior σ as an infinite sequence of states $< s_0, s_1, s_2, \cdots >$, where each state s_i has been assigned a corresponding variable.

3.1 Elements of State Logic in TLA

3.1.1 Variables

An infinite number of variable names (e.g., x or y) and a value class set that can be assigned to the variables are assumed. These value classes include strings, numbers, sets, and functions. If x is a variable, $[[x]]$ is the function that semantically maps the value of x in the states. Similarly, $[[x]](s)$ is the function of the value of x in the state s.

3.1.2 State and predicate functions

A state function is a non-Boolean expression built from variables, constants, and standard arithmetic operators. The semantics of $[[f]]$, where f is a state function, consists of mapping states into values. To obtain the value of f in state s, we replace each variable x_i of f with $[[x_i]](s)$.

Similarly, a predicate function or predicate P is a Boolean predicate. $[[P]]$ is an application of the set of states in a Boolean value. s fulfills P iff $[[P]](s)$ is equal to TRUE.

3.1.3 Actions

An action is a Boolean expression containing non-qualified primed variables (such as x'), standard operators, and values. An action represents an atomic operation of the system. Semantically, an action A is true or false for a pair of states, and takes the primed variables belonging to the second state. If we take an old state s, a new state t, and an action A, 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 equal to TRUE.

3.1.4 Active action in a state and execution

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 (equation 1).

$$[[Enabled\ A]](s) \doteq \exists t \in \sigma : [[A]](s,t) \tag{1}$$

An action A can be broken down into two logical formulae: G refers to the precondition, and B to the body of the action in itself (eq. 2).

$$A \equiv G \wedge B \tag{2}$$

3.2 Elements of Temporal Logic in TLA

In TLA, the behavior of a system is modeled as an infinite sequence of states, where their basic elements are actions and temporal logic. The actions help us in a simple and specific way to control the potential next step. Temporal logic includes predicates, actions, logical operators, and temporal operators.

In order to define the semantics of temporal formulae, we need to extend the semantic definition of the predicates whose value will be TRUE or FALSE in a given behavior.

A behavior satisfies the predicate P iff (eq. 3) is satisfied in the first state.

$$[[P]](< s_0, s_1, s_2, \cdots >) \Rightarrow [[P]](s_0) \tag{3}$$

Similarly, a behavior satisfies the action A iff the first pair of states of the given behavior is an A-step (eq. 4).

$$[[A]](< s_0, s_1, s_2, \cdots >) \Rightarrow [[A]](s_0, s_1) \tag{4}$$

3.2.1 The Always Operator

The operator $\Box$ (always) is the basic block of any temporal logic. Given a formula F, $\Box F$ asserts that F is always TRUE (eq. 5):

$$[[\Box F]](< s_0, s_1, s_2, \cdots >) \doteq \forall n \geq 0 : [[F]](< s_n, s_{n+1}, s_{n+2}, \cdots >) \tag{5}$$

From equations 3 and 4 we define a behavior σ that satisfies $\Box P$ iff all the states of the behavior σ satisfy P. Similarly, a behavior σ satisfies $\Box A$ iff all steps (s_i, s_{i+1}) are A-steps.

3.2.2 The Operator Eventually

All temporal formulae can be constructed with traditional operators of first-order logic and the operator $\Box$. However, it is useful to define other operators such as $\Diamond$ (eventually). The formula $\Diamond F$ asserts that F is eventually TRUE (eq. 6):

$$[[\Diamond F]](< s_0, s_1, s_2, \cdots >) \doteq \exists n \geq 0 : [[F]](< s_n, s_{n+1}, s_{n+2}, \cdots >)$$

(6)

In other words, $\Diamond F$ indicates that F is not always FALSE. Therefore:

$$\Diamond F \equiv \neg\Box\neg F \tag{7}$$

3.2.3 Validity

A formulae F is valid iff it is satisfied for all behaviors (eq. 8).

$$\vDash F \doteq \forall \sigma \in S^{\infty} : [[F]](\sigma) \tag{8}$$

S^{∞} denotes the set of all possible behaviors.

3.2.4 Specification in TLA

All previous definitions can be summed up in a single one to make a formal specification. A formal specification has the following general formula (eq. 9).

$$\Pi \doteq Init \wedge \Box(A_1 \vee A_2 \vee \cdots \vee A_n) \tag{9}$$

A formula Π is TRUE in a behavior iff its first state satisfies the predicate *Init* and each step at least satisfies an action A_i.

Actions in TLA are allowed only if the predicate *Enabled* is TRUE and the context can be expressed as in equation 10.

$$[A]_v \doteq A \vee v' = v \tag{10}$$

This expression indicates that a new v-step is a step where either A is an A-step or the values of v do not change. Bear in mind that v is a set of important variables in the action to be executed. The action $v' = v$ is usually called implicit stuttering action.

Similarly, a non-stuttering execution can be defined:

$$\langle A \rangle_v \doteq A \vee v' \neq v \tag{11}$$

That is to say, the execution is a v–step if the step changes the values of some of the variables indicated in v.

3.2.5 Fairness Operators

The fairness operators are in charge of ensuring that "nothing abnormal will happen". There are two types: weak fairness (WF) and strong fairness (SF) operators.

Weak fairness. The weak fairness formula asserts that an action has to be infinitely executed frequently if it is continuously enabled for an infinitely long time.

$$WF_v(A) \doteq \Box\Diamond\langle A \rangle_v \vee \Box\Diamond\neg Enabled\langle A \rangle_v \tag{12}$$

Strong fairness. The strong fairness formula asserts that an action has to be infinitely executed frequently if it is often infinitely enabled.

$$SF_v(A) \doteq \Box\Diamond\langle A \rangle_v \vee \Diamond\Box\neg Enabled\langle A \rangle_v \tag{13}$$

Similarly, the following theorem can be established:

$$SF_v(A) \Rightarrow WF_v(A) \tag{14}$$

3.2.6 Formal specification of a system

The formal specification of a system is as follows (eq. 15):

$$\Pi \doteq Init \wedge \Box[N]_v \wedge L \tag{15}$$

3.3 Formal approach to state diagram modelling

Although it is possible to model any system with the structures previously described, it would be very useful in this case study to establish a formal mechanism for modelling state diagrams in TLA, since this will be the basis for formalizing the language/action paradigm.

Lamport [30, 31] suggests the following formal approach to represent an action-predicate diagram. An action-predicate diagram is a diagram with a subset of nodes identified as initial nodes, where each node is labeled by a state predicate and every edge is labeled by an action. The following notation is used:

- 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: The predicate labeling node n.
- ε_e: The action labeling edge n.

The formula Δ, representing the diagram is defined as follows:

$$\begin{aligned} Init_{\Delta} &\doteq \exists n \in I : P_n \\ \mathscr{A}_n &\doteq \exists e \in E(n) : \varepsilon_e \wedge P'_{d(e)} \\ \Delta &\doteq Init_{\Delta} \wedge \forall n \in N : \Box[P_n \Rightarrow \mathscr{A}_n]_v \end{aligned} \tag{16}$$

If no specific label is attached to edge e, we take ε_e to be TRUE. When no set of initial nodes is explicitly specified, we take I to be N. $\mathscr{A}_n$ is FALSE if no edges originate in node n.

From this definition, we can now represent the diagram in TLA.

4. Formalizing the language/action paradigm

In order to formalize in TLA the state diagram introduced in section 2.4.1, its edges are labeled as e_{ij} as shown in figure 3 in order to follow the reference model later.

As a prior step, we will define what is understood by work. Work W_k is a quadruple expressed in the equation 17 where:

$$W_k \doteq \{I, H, P, SC, V\} \tag{17}$$

- I: Information needed to carry out the task.
- H: Tools and methods needed for performing the task.
- P: Person, role or agent with the capacity to perform the task.
- SC: Terms of satisfaction for the work to be considered completed.

- V: Set of state variables belonging to the workflow.

Each of these sub-variables is denoted as $W_k.I$, $W_k.H$, $W_k.P$, $W_k.SC$, $W_k.V$, respectively, where W_k is the work element.

The set of nodes in our diagram will be denoted by N_i, where i is the number of the current state in the figure 3 (eq. 18).

$$N \doteq \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9\} \tag{18}$$

The initial state will be N_1, since communication starts there. Therefore:

$$I \doteq \{N_1\} \tag{19}$$

In this way, and bearing in mind the equation to be satisfied in the initial state, we have:

$$Init_\Delta \doteq \exists n \in I : P_n \tag{20}$$

Therefore, to complete the definition of the initial state we have to give meaning to P_{N_1}, and this has to be done in the form of equation (eq. 21).

$$\begin{aligned} P_{N_1} \doteq\ & \text{DetectTrigger(Type,Origine,Workflow)} \wedge \\ & Workflow = IDWF \wedge Type \in T_{WFID} \wedge Origine = A \end{aligned} \tag{21}$$

This predicate indicates that the workflow will start when an event coming from agent A is triggered and detected (DetectTrigger). From this point on, the agent A corresponds to the client Cli, and the event is a request for service identified as IDWF and corresponding to the workflow.

Once the definition of the initial state N_1 is completed, we can define the next state variables belonging to the set V. For simplicity sake, we will not follow the notation but rather the prefix V:

- W_kS: State of the work.
- W_fS: Current state of the workflow.
- W_fP: Current phase of the workflow.
- W_k: The variable representing the current work.
- XW_k: External work proposed by A (where A is the workflow's client).
- W_fCli: Workflow client.
- W_fSvr: Workflow server.

Once the workflow has started, there is a transition to state N_2 through edge e_{12}. This edge assigns the variables shown in equation 22.

$$\begin{aligned} e_{12} \doteq\ & W_k' = XW_k \wedge W_fP' = \text{“preparation”} \\ & \wedge\ W_fS' = \text{“active”} \\ & \wedge\ W_kS' = \text{“feasibility study”} \\ & \wedge W_fCli' = A \\ & \wedge W_fSvr' = \text{SelectAgent}(OrgDB, B) \end{aligned} \tag{22}$$

The workflow is still in progress and the task is assigned to the variable that records it within the system. The workflow $IDWF$ is now instantiated and the requested external work is assigned to the execution model. This variable includes work XW_k and the server B (Srv), which up to this point was unknown, as well as the initial terms of satisfaction $XW_k.SC$.

To assign or chose the server, we will use the function SelectAgent(OrgDB,B) that will select an agent from the organization among those fulfilling B type requirements (this function will have to access the knowledge base of the organization $OrgDB$).

The workflow is now at state N_2. In this state, the work B will perform has to be evaluated. The petition and result of such an evaluation will be represented by the function EvalWork(W_k,Agent).

This function has two parameters, the work to be evaluated W_k and the agent in charge of such an evaluation. It also includes the option for the evaluating agent to modify or make proposals to change the response state. This response is obtained from CounterOffer(W_k,Agent) that returns a new offer of the work W_k from the agent. It is possible to return: Commitment, counteroffer, Decline or cancel.

In this way P_{N_2} will be in charge of carrying out the work evaluation.

$$\begin{aligned} P_{N_2} \doteq\ & S = \text{EvalWork}(W_k, B) \wedge \\ & (S = \text{“Commitment”} \\ & \vee\ S = \text{“Counteroffer”} \\ & \vee\ S = \text{“Decline”}) \end{aligned} \tag{23}$$

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 of the workflow and comes from state 2 due to a decline of the offer (eq. 24).

$$\begin{aligned} e_{28} \doteq\ & S = \text{“decline”} \\ & \wedge W_fP' = \text{“preparation”} \\ & \wedge\ W_fS' = \text{“abort”} \\ & \wedge\ W_kS' = \text{“Svr aborts”} \end{aligned} \tag{24}$$

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 (eq. 25). This possibility is only open if the evaluation leads to “decline”.

$$P_{N_8} \doteq \text{Exception}W_f(IDWF, \text{“abort work”}, W_k) \tag{25}$$

In this state, the function ExceptionW_f is used. This 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}. In this way the assignation of state variables is as shown in equation 26.

$$\begin{aligned} e_{26} \doteq\ & S = \text{“counteroffer”} \\ & \wedge W_fP' = \text{“negotiation”} \\ & \wedge\ W_fS' = \text{“active”} \\ & \wedge\ W_kS' = \text{“negotiation – Svr study the proposals”} \\ & \wedge\ W_k{}' = \text{CounterOffer}(W_k, W_fSvr) \end{aligned} \tag{26}$$

In state N_6 another evaluation is carried out, but this time is done by A (P_{N_6}. This is shown in equation 27).

$$\begin{aligned} P_{N_6} \doteq\ & S = EvalWork(W_k, W_f Cli) \wedge \\ & (S = \text{“accepted”} \\ & \vee\ S = \text{“counteroffer”} \\ & \vee\ S = \text{“decline”}) \end{aligned} \tag{27}$$

From state 6 we have three options: go to state 2 so that B reconsiders the offer made by the agent A (eq. 28), or it refuses it and goes to state 8, where the incomplete termination will call for exception (eq. 29).

$$\begin{aligned} e_{62} \doteq\ & S = \text{“counteroffer”} \\ & \wedge W_f P' = \text{“negotiation”} \\ & \wedge\ W_f S' = \text{“active”} \\ & \wedge\ W_k S' = \text{“negotiation - Cli study the proposals”} \\ & \wedge\ W_k{}' = \text{ConunterOffer}(W_k, W_f Cli) \end{aligned} \tag{28}$$

$$\begin{aligned} e_{68} \doteq\ & S = \text{“decline”} \\ & \wedge W_f P' = \text{“negotiation”} \\ & \wedge\ W_f S' = \text{“aborted”} \\ & \wedge\ W_k S' = \text{“Cli refuse the work”} \end{aligned} \tag{29}$$

The third option open to A is to accept the current work W_k, modified by B in this case, and transit to state N_3 (eq. 30). Similarly, if the evaluation leads to B accepting the work, we could directly go from state N_2 to N_3, (eq. 31).

$$\begin{aligned} e_{63} \doteq\ & S = \text{“accept”} \\ & \wedge W_f P' = \text{“development”} \\ & \wedge\ W_f S' = \text{“execution”} \\ & \wedge\ W_k S' = \text{“accepted”} \end{aligned} \tag{30}$$

$$\begin{aligned} e_{23} \doteq\ & S = \text{“accept”} \\ & \wedge W_f P' = \text{“development”} \\ & \wedge\ W_f S' = \text{“execution”} \\ & \wedge\ W_k S' = \text{“accepted”} \end{aligned} \tag{31}$$

Both equations are similar and do not have any effect on the variable W_k, which represents the current work, whether this is the original one or a counteroffer made by A or B (we assume that the equation 10 has been applied, i.e., if the work does not change, the variable preserves its value in the next state).

In order to define P_{N_3} we have to use some back-up functions that will enable us to express the semantics of this state. These functions are :

- 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.

Let us bear in mind that in this state all subtasks have to be run for the workflow to be completed. In addition, one of the following 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 $W_k.a_i$ be each of the subtasks comprising the task of a workflow W_k, then P_{N_3} is defined as (eq. 32 and 33):

$$\begin{aligned} P_{N_3} \doteq\ & \forall (a_i \in W_k, t_i / t_i = Trg(a_i))\ \text{Trigger}(t_i, W_k.a_i) \\ & \wedge (ExecutionPredicate) \end{aligned} \tag{32}$$

$$\begin{aligned} ExecutionPredicate \doteq\ & \Box \forall a_i \in W_k / \text{Completed}(W_k.a_i) \\ & \vee \Box \exists a_i \in W_k / \text{Aborted}(W_k.a_i) \\ & \vee \text{Abort}(W_k Cli) \end{aligned} \tag{33}$$

It is important to bear in mind that this equation can be applied to the states where the task to be carried out is evaluated, since such evaluation can give rise to a explosion of subtasks.

Two options are possible: aborting the execution and going to state N_7 or N_9 depending on who aborted or going to evaluation state N_4. These cases are expressed in equations 34, 35, and 36, respectively.

$$\begin{aligned} e_{39} \doteq\ & \exists a_i \in W_k / \\ & (W_k.a_i.W_f S = \text{“aborted”} \\ & \wedge W_k.a_i.W_k S = \text{“Svr refuses the work”}) \\ & \wedge (W_f P' = \text{“execution”} \\ & \wedge\ W_f S' = \text{“aborted”} \\ & \wedge\ W_k S' = \text{“Svr refuses the work”}) \end{aligned} \tag{34}$$

$$\begin{aligned} e_{37} \doteq\ & \exists a_i \in W_k / \\ & (W_k.a_i.W_f S = \text{“aborted”} \\ & \wedge W_k.a_i.W_k S = \text{“Cli refuses the work”}) \\ & \wedge (W_f P' = \text{“execution”} \\ & \wedge\ W_f S' = \text{“aborted”} \\ & \wedge\ W_k S' = \text{“Cli refuses the work”}) \end{aligned} \tag{35}$$

$$\begin{aligned} e_{34} \doteq\ & \forall a_i \in W_k / \\ & (W_k.a_i.W_f S = \text{“accepted”} \\ & \wedge W_k.a_i.W_k S = \text{“completed”}) \\ & \wedge (W_f P' = \text{“evaluation”} \\ & \wedge\ W_f S' = \text{“active”} \\ & \wedge\ W_k S' = \text{“Cli's final evaluation”}) \end{aligned} \tag{36}$$

State N_4 consists in evaluating the final work (see equation 37) using a predicate named EvalReport.

$$\begin{aligned} P_{N_4} \doteq\ & \text{EvalReport}(W_k, B) = \text{“correct”} \\ & \vee\ \text{EvalReport}(W_k, B) = \text{“incorrect”} \\ & \vee\ \text{Exception}W_f(IDWF, \text{“abort work”}, W_k) \end{aligned} \tag{37}$$

If the work satisfies the terms, there is a transit to a final and 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. These cases are expressed in equations 38, 39, and 40.

$$\begin{aligned} e_{49} \doteq\ & W_fP' = \text{“evaluation”} \\ & \wedge W_fS' = \text{“aborted”} \\ & \wedge W_kS' = \text{“Cli rejects the work”} \end{aligned} \quad (38)$$

$$\begin{aligned} e_{45} \doteq\ & W_fP' = \text{“Complete”} \\ & \wedge W_fS' = \text{“Completed”} \\ & \wedge W_kS' = \text{“work accepted for complete”} \end{aligned} \quad (39)$$

$$\begin{aligned} e_{43} \doteq\ & W_fP' = \text{“Execution”} \\ & \wedge W_fS' = \text{“Active”} \\ & \wedge W_kS' = \text{“work not accepted for complete : reexecute”} \end{aligned} \quad (40)$$

State N_5 only has to send a completed signal (eq. 41).

$$P_{N_5} \doteq \text{Exception}W_f(IDWF, \text{“Completed WF successful”}, W_k) \quad (41)$$

Once these definitions are completed and the model equations are applied to formalize a state diagram (eq. 16) we obtain the formal representation.

5. Formalizing the language/action paradigm as external behavior

As a prior step, we will define what is understood by *work*. Work w is a quadruple expressed in the equation $w \doteq \{I, H, P, SC, V\}$ where: I: Information needed to carry out the task. H: Tools and methods needed to perform the task. P: Person, role or agent with the capacity to perform the task. SC: Terms of satisfaction for the work to be considered completed. V: Set of state variables belonging to the workflow.

Each of these sub-variables is denoted as $W_k.I$, $W_k.H$, $W_k.P$, $W_k.SC$, $W_k.V$, respectively, where W_k is the work element.

Also we define a set of workflow attributes that can be used as control variables for its execution (these attributes are include in V): W_kS: State of the work. W_fS: Current state of the workflow. W_fP: Current phase of the workflow. W_k: The variable representing the current work. XW_k: External work proposed by A (where A is the workflow's client). W_fCli: Workflow client. W_fSvr: Workflow server.

At this point we can get the workflow behavior formalization. We use the example from figure 2 to illustrate how the main primitives formalize in TLA. We use the W_fP and W_fS workflow attributes for the specification. The control variable W_fP that contains the actual worflow phase is as follows:

$$W_fP \in \{\text{“}preparation\text{”}, \text{“}negotiation\text{”}, \text{“}development\text{”}, \text{“}acceptance\text{”}, \text{“}final\text{”}\}$$

W_fS represents the internal status value for a workflow. This variable is equal to “finished” when the workflows is terminated.

Also for the sub–workflows sequentiation into each phase we use the control attribute *sec*. This variable stores the actual execution position at each workflow phase. The *sec* type is an array that can be noted with $W_f.sec$ if it has only one position or $W_f.sec[1], W_f.sec[2], \ldots W_f.sec[n]$ that specifies multiples ways of execution.

5.1 Workflow loop formalization

With these elements we can formalize the case study in TLA. For the formalization we use the following notation:

- Ψ_{WF} a formal specification of workflow WF.
- WF the relationship with the following workflow state.
- $Init\Psi_{WF}$ initial condition for workflow execution. This condition is composed by $Init_{WF}$ (initial condition for work variables of workflow WF) and $Init_\Psi$ or initial condition for WF control variables. Therefore $Init\Psi_{WF} = Init_{WF} \wedge Init_\Psi$.
- $WF_{\{phase\_name\}}$ the equation that defines the behavior at workflow level for each phase. $WF_{\{negotiation\}}$ corresponds with the negotiation phase specification. If the specification does not exists its value will be $WF_{\{phase\_name\}} = \top$ and therefore always correct.
- $\Psi_{WF_{\{phase\_name\}}}$ corresponds with the phase specification.
- $Ex(WF_{\{phase\_name\}})$ corresponds with the execution level specification of $\Psi_{WF_{\{phase\_name\}}}$. This is an atomic sub–workflow and therefore equivalent at the definition level with the general formulae Ψ_{WF}. We use this term to nest specifications.
- f corresponds with the set of variables used by workflow both control variables $f_{<control>}$ and work variables $f_{<WF>}$ ($f = f_{<control>} \cup f_{<WF>}$).

To begin with the WF specification we define the general formulae Ψ_{WF} (equation 42).

$$\begin{aligned} \Psi_{WF} \equiv\ & Init\Psi_{WF} \wedge \Box[WF]_f \\ & \wedge SF(WF_{\{preparation\}}) \\ & \wedge SF(WF_{\{negotiation\}}) \\ & \wedge SF(WF_{\{development\}}) \\ & \wedge SF(WF_{\{acceptance\}}) \end{aligned} \quad (42)$$

The initial state is the one at both control variables and work variables of WF (to simplify we do not use this type of variable). This state indicates that the workflow is in the “preparation” phase (equation 43).

$$\begin{aligned} Init\Psi_{WF} &\equiv Init_\Psi \wedge Init_{WF} \\ Init_\Psi &\equiv W_fP = \text{“}preparation\text{”} \end{aligned} \quad (43)$$

The workflow execution behavior consists in a sequential execution for each phase. Each phase executes its specification;

and when it finishes then the workflow transits to the next phase until the workflow has bee correctly completed (equation 44).

$$\begin{aligned} WF \equiv\ & \vee WF_{\{preparation\}} \\ & \vee WF_{\{negotiation\}} \\ & \vee WF_{\{development\}} \\ & \vee WF_{\{acceptance\}} \end{aligned} \tag{44}$$

Each specification can be found in equations 45, 46, 47 and 48.

$$\begin{aligned} WF_{\{preparation\}} \equiv\ & W_fP = \text{“}preparation\text{”} \\ & \wedge \Psi_{WF_{\{preparation\}}} \\ & W_fP' = \text{“}negotiation\text{”} \\ & \wedge Unchg_{<f-\{W_fP\}>} \end{aligned} \tag{45}$$

$$\begin{aligned} WF_{\{negotiation\}} \equiv\ & W_fP = \text{“}negotiation\text{”} \\ & \wedge \Psi_{WF_{\{negotiation\}}} \\ & W_fP' = \text{“}development\text{”} \\ & \wedge Unchg_{<f-\{W_fP\}>} \end{aligned} \tag{46}$$

$$\begin{aligned} WF_{\{development\}} \equiv\ & W_fP = \text{“}development\text{”} \\ & \wedge \Psi_{WF_{\{development\}}} \\ & W_fP' = \text{“}acceptance\text{”} \\ & \wedge Unchg_{<f-\{W_fP\}>} \end{aligned} \tag{47}$$

$$\begin{aligned} WF_{\{acceptance\}} \equiv\ & W_fP = \text{“}acceptance\text{”} \\ & \wedge \Psi_{WF_{\{acceptance\}}} \\ & W_fP' = \text{“}final\text{”} \\ & \wedge Unchg_{<f-\{W_fP\}>} \end{aligned} \tag{48}$$

5.2 Sequential tasks formalization

Continuing with the example, we formalize tasks A and B where execution must be sequential. We use the *sec* attribute, which notes the actual execution sequence into each phase.

The general equation begins with a development phase $\Psi_{WF_{\{development\}}}$ (equation 49).

$$\begin{aligned} \Psi_{WF_{\{development\}}} \equiv\ & Init\Psi_{WF_{\{development\}}} \\ & \wedge \Box[Ex(WF_{\{development\}}]_f \\ & \wedge SF(A) \\ & \wedge SF(B) \end{aligned} \tag{49}$$

The initial state variables has an effect on the sequential step (equation 50).

$$\begin{aligned} Init\Psi_{WF_{\{development\}}} \equiv\ & WF_{\{dev.\}}.sec = \text{“}A\text{”} \\ & \wedge Init_{WF_{\{dev.\}}} \end{aligned} \tag{50}$$

We used operator $\wedge$ and $\vee$ following Lamport's recommendation [31]
Unchg is 'Unchanged operator' in TLA

The execution is based on sub-workflow specifications A and B (equations 51, 52, 53 and .

$$Ex(WF_{\{development\}} \equiv A \vee B \tag{51}$$

$$\begin{aligned} A \equiv\ & \wedge WF_{\{development\}}.sec = \text{“}A\text{”} \\ & \wedge \Psi_A \wedge WF_{\{development\}}.sec' = \text{“}B\text{”} \\ & \wedge Unchg_{<f-WF_{\{development\}}.sec>} \end{aligned} \tag{52}$$

$$\begin{aligned} B \equiv\ & \wedge WF_{\{development\}}.sec = \text{“}B\text{”} \\ & \wedge \Psi_B \wedge WF_{\{development\}}.sec' = \text{“}end\text{”} \\ & \wedge Unchg_{<f-WF_{\{development\}}.sec>} \end{aligned} \tag{53}$$

5.3 Conditional task formalization

The tasks $OP1$ and $OP2$ are executed depending precondition G at WF preparation phase. The equation 54 specifies how this phase works and the fairness section specifies that both $OP1$ and $OP2$ will be executed eventually if infinitely often they are active for its execution.

$$\begin{aligned} \Psi_{WF_{\{preparation\}}} \equiv\ & Init_{WF_{\{preparation\}}} \\ & \wedge \Box[Ex(WF_{\{preparation\}})]_f \\ & \wedge (WF(OP1) \\ & \vee WF(OP2)) \end{aligned} \tag{54}$$

We used G and the predicate $Init_{WF_{\{preparation\}}}$ to decide about the execution for this sub–workflows (equation 55).

$$\begin{aligned} Init_{WF_{\{prep.\}}} \equiv\ & (G \rightarrow WF_{\{prep.\}}.sec = \text{“}OP1\text{”}) \\ & \vee (\neg G \rightarrow \\ & WF_{\{prep.\}}.sec = \text{“}OP2\text{”}) \end{aligned} \tag{55}$$

The execution definition, using the initialitation formulae, is (equations 56, 57 and 58):

$$\begin{aligned} Ex(WF_{\{preparation\}}) \equiv\ & (G \rightarrow OP1) \wedge \\ & (\neg G \rightarrow OP2) \end{aligned} \tag{56}$$

$$\begin{aligned} OP1 \equiv\ & WF_{\{preparation\}}.sec = \text{“}OP1\text{”} \\ & \wedge \Psi_{OP1} \\ & WF_{\{preparation\}}.sec' = \text{“}fin\text{”} \\ & \wedge Unchg_{<f-sec>} \end{aligned} \tag{57}$$

$$\begin{aligned} OP2 \equiv\ & WF_{\{preparation\}}.sec = \text{“}OP2\text{”} \\ & \wedge \Psi_{OP2} \\ & WF_{\{preparation\}}.sec' = \text{“}fin\text{”} \\ & \wedge Unchg_{<f-sec>} \end{aligned} \tag{58}$$

5.4 Parallel execution formalization

The *PAR*1 and *PAR*2 workflows are executed at the execution phase of workflow B. We use the Ψ_B formulae for the specification (equation 59).

$$\Psi_B \equiv Init_{\Psi_B} \wedge \Box[B] \wedge SF(B) \tag{59}$$

The B specification follows the general schema that we used for WF (equation 60).

$$\begin{aligned}
Init_{\Psi_B} &\equiv B.W_f P = \text{“}preparation\text{”} \\
&\quad \wedge Init_{B_{esp}} \\
B &\equiv B_{\{preparation\}} \\
&\quad \vee B_{\{negotiation\}} \\
&\quad \vee B_{\{development\}} \\
&\quad \vee B_{\{acceptance\}} \\
&\quad \ldots \\
B_{\{development\}} &\equiv \wedge B.W_f P = \text{“}development\text{”} \\
&\quad \wedge \Psi_{B_{\{development\}}} \\
&\quad \wedge B.W_f P' = \text{“}acceptance\text{”} \\
&\quad \ldots
\end{aligned} \tag{60}$$

Finally we used the parallel composition to define $B_{\{development\}}$ (equation 61).

$$\Psi_{B_{\{development\}}} \equiv (\Psi_{PAR1} \wedge \Psi_{PAR2}) \tag{61}$$

6. Conclusions and Further work

Workflow technology needs something else other than commercial workflow products to advance process automatization. The descriptive specification of workflow maps is very useful for managers - who have a description of business processes - and for development teams. However, we need more powerful tools to demonstrate workflow properties without the use of simulations.

This paper introduces formal methods to carry out automated demonstrations of workflow properties. The formal method chosen is TLA. In order to prove the power of this logic, we have formalized the workflow loop of communication-based technologies. Modelling the workflow structure has been addressed at a micro-level (internal behavior), i.e., at the workflow loop's inner operating level, and macro–level (external behavior), that is, the inter-workflows relations.

We have shown that combining the use of traditional modelling methodologies and TLA makes possible the representation of workflow loops in a way that user-designers without expertise in logic can understand, and at the same time the model can be automatically analyzed by demonstrating workflow properties such as consistency, time deadlines, and other desirable characteristics.

References

[1] RODDEN, Tom et SOMMERVILLE, Ian. Supporting cooperative software engineering. In P*roceedings of the Conference on Computer Supported Cooperative Work: The Multimedia and Networking Paradigm (CSCW '91)*, July 1991, p. 166–177, Uxbridge, UK. Unicom Seminars.

[2] GEORGAKOPOULOS, Diimitrios, HORNICK, Mark, et SHETH, Amit. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 1995, vol. 3, no 2, p. 119-153.

[3] CARO, José Luis, GUEVARA, Antonio, AGUAYO, Andrés, et al. Communication based workflow loop formalization using Temporal Logic of Actions (TLA). *In : Computer Supported Acitivity Coordination*. 2004. p. 233-238.

[4] TER HOFSTEDE, Arthur HM, ORLOWSKA, Maria E., et RAJAPAKSE, Jayantha. Verification problems in conceptual workflow specifications. *Data & Knowledge Engineering*, 1998, vol. 24, no 3, p. 239-256.

[5] WYNN, Moe Thandar, VERBEEK, H. M. W., VAN DER AALST, Wil MP, et al. Business process verification–finally a reality!. *Business Process Management Journal*, 2009, vol. 15, no 1, p. 74-92.

[6] PNUELI, Amir. The temporal logic of programs. In : F*oundations of Computer Science*, 1977., 18th Annual Symposium on. IEEE, 1977. p. 46-57.

[7] SHETH, Amit et RUSINKIEWICZ, Marek. On transactional workflows. *Data Engineering Bulletin*, 1993, vol. 16, no 2, p. 20.

[8] CARO, J. L., GUEVARA, A., et AGUAYO, A. Workflow: a solution for cooperative information system development. *Business Process Management Journal*, 2003, vol. 9, no 2, p. 208-220.

[9] LEIVA, José L., CARO, José Luis, GUEVARA, Antonio, et al. A Cooperative Method for System Development and Maintenance Using Workflow Technologies. In : *ICEIS (5)*. 2008. p. 130-135.

[10] MEDINA-MORA, Raul, WINOGRAD, Terry, FLORES, Rodrigo, et al. The action workflow approach to workflow management technology. In : *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. ACM, 1992. p. 281-288.

[11] LAMPORT, Leslie. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1994, vol. 16, no 3, p. 872-923.

[12] CARO, José L., GUEVARA, Antonio, AGUAYO, Andrés, et al. Increasing the quality of hotel management information systems by applying workflow technology. *Information Technology & Tourism*, 2000, vol. 3, no 2, p. 87-98.

[13] CARO, José L., GUEVARA, Antonio, AGUAYO, Andrés, et al. Workflow management appliend to the information

system in tourism. *Journal of Travel Research (Information Technology)*, 2000, vol. 6, no 2, p.36–45.

[14] WFMC. Workflow reference model. Technical report, Workflow Management Coalition, Brussels, 1994.

[15] CARO, José Luis, GUEVARA, Antonio, GÁLVEZ, Sergio, et al. A Temporal Reasoning Approach of Communication Based Workflow Modelling. In: *ICEIS (3)*. 2003. p. 245-250.

[16] SEARLE, John R. *Speech acts: An essay in the philosophy of language*. Cambridge university press, 1969.

[17] SEARLE, John R. A taxonomy of illocutionary acts. Linguistic Agency University of Trier, 1976. John R. Searle. A taxonomy of illocutionary acts. In Keith Gunderson, editor, Language, Mind, and Knowledge. Minnesota Studies in the Philosophy of Science, 1975., vol. 7, p. 344–369. University of Minnesota Press, Minneapolis, Minnesota.

[18] CASATI, Fabio, FUGINI, Mariagrazia, et MIRBEL, Isabelle. An environment for designing exceptions in workflows. *Information systems*, 1999, vol. 24, no 3, p. 255-273.

[19] DEPKE, Ralph, HECKEL, Reiko, et KUSTER, Jochen Malte. Roles in agent-oriented modeling. *International Journal of Software engineering and Knowledge engineering*, 2001, vol. 11, no 03, p. 281-302.

[20] GEPPERT, Andreas, TOMBROS, Dimitrios, et DITTRICH, Klaus R. Defining the semantics of reactive components in event-driven workflow execution with event histories. *Information Systems*, 1998, vol. 23, no 3, p. 235-252.

[21] JOOSTEN, Stef. Trigger modelling for workflow analysis. In : *Proceedings CON94: Workflow Management, Challenges, Paradigms and Products*. 1994. p. 236-247.

[22] VAN DER AALST, W. M. P. Petri-net-based workflow management software. In : *Proceedings of the NFS Workshop on Workflow and Process Automation in Information Systems*. IEEE Computer Society, 1996. p. 114-118.

[23] VAN DER AALST, Wil MP et TER HOFSTEDE, Arthur HM. YAWL: yet another workflow language. *Information systems*, 2005, vol. 30, no 4, p. 245-275.

[24] SADIQ, Wasim et ORLOWSKA, Maria E. Analyzing process models using graph reduction techniques. *Information systems*, 2000, vol. 25, no 2, p. 117-134.

[25] VAN DER AALST, Wil MP. Formalization and verification of event-driven process chains. *Information and Software technology*, 1999, vol. 41, no 10, p. 639-650.

[26] VAN DER AALST, Wil MP et TER HOFSTEDE, Arthur HM. Verification of workflow task structures: A petri-net-baset approach. *Information systems*, 2000, vol. 25, no 1, p. 43-69.

[27] ZHUGE, Hai, CHEUNG, To-yat, et PUNG, Hung-Keng. A timed workflow process model. *Journal of Systems and Software*, 2001, vol. 55, no 3, p. 231-243.

[28] Terry Winograd. The language/action perspective. *ACM Transactions on Office Information Systems*, 1988, vol. 6, no 2, p. 83–86.

[29] John L. Austin. *How to Do Things With Words*. Oxford University Press, Oxford, 1962.

[30] LAMPORT, Leslie. Tla in pictures. Technical Report 127, Digital Equipment Corporation, Systems Research Centre, 1 September 1994.

[31] LAMPORT, Leslie. TLA in pictures. *IEEE Transactions on Software Engineering*, 1995, vol. 21, no 9, p. 768-775.