A Logic-Based Formalism to Model and Analyze Workflow Executions

Gianluigi Greco[1], **Antonella Guzzo**[1], and **Domenico Saccà**[1,2]

DEIS[1], University of Calabria, Via Pietro Bucci 41C, 87036 Rende, Italy
ICAR, CNR[2], Via Pietro Bucci 41C, 87036 Rende, Italy
{ggreco,guzzo}@si.deis.unical.it, sacca@icar.cnr.it

Abstract. We describe a new approach to workflow analysis, which combines a rich graph representation of workflow schemes with simple (i.e., stratified), yet powerful DATALOG rules to express complex properties and constraints on executions. Both the graph representation and the DATALOG rules are mapped into a unique program in DATALOG$^{ev!}$, that is a recent extension of DATALOG for handling events. This mapping enables the designer to simulate the actual behavior of the modelled scheme by fixing an initial state and an execution scenario (i.e., a sequence of executions for the same workflow) and querying the state after such executions.

1 Introduction

A great deal of recent research concerns the task of modelling workflow schemes and several formalisms for specifying structural properties have been already proposed to support the designer in devising all admissible execution scenarios. Most of such formalisms are based on graphical representations in order to give a simple and intuitive description of the workflow structure. In particular, the most common approach is the use of a *control flow graph*, in which the workflow is represented by a labelled directed graph whose nodes correspond the task to be performed, and the arcs describe the precedences among them.

As pointed out by many authors, see e.g [2], the essential drawback of approaches based on the *control flow graph* is their limited expressive power: they are only able to specify *local* dependencies whereas properties such as synchronization, concurrency, or serial execution of tasks, also called in the literature *global constraints*, cannot be described. The current trend in workflow management system is to left unstated all the complex constraints (thus delivering an incomplete specification) or to eventually expressed them using other formalisms, e.g., some form of logics.

In this paper, we present an overview of a system which realizes a logic based formalism which combines a rich graphical representation of workflow schemes with simple (i.e., locally stratified), yet powerful DATALOG rules to express complex properties and *global constraints* on executions. Both the graph representation and the DATALOG rules are mapped into a unique program in $\mathtt{DATALOG}^{\mathtt{ev!}}$, that is a recent extension of DATALOG for handling events. We stress that we are currently on the way of implementing the system using the **DLV** system [5], with the aim of obtaining an effective tool for simulating and reasoning on workflows. Concerning the translation of $\mathtt{DATALOG}^{\mathtt{ev!}}$ programs into **DLV** programs, we mention that in [4] some of the authors have already shown how to compile them into a classical logic programming framework.

2 The Overall Architecture

Our workflow system comprises three databases: $\mathbf{DB}_{CF}(\mathcal{WS})$, storing the control flow structure, $\mathbf{DB}_{WE}(ID)$, storing information on the instance evolution, such as the status of the tasks and of the servers, and $\mathbf{DB}_{I}(\mathcal{WS})$, storing additional information needed to the execution. Note that, $\mathbf{DB}_{CF}(\mathcal{WS})$ and $\mathbf{DB}_{I}(\mathcal{WS})$ are shared among the different instances.

Moreover, all the *global constraints* and additional constraints on the scheduling of the activities can be translated into a $\mathtt{DATALOG}^{\mathtt{ev!}}$ program $\mathbf{P_{Constr}}(\mathcal{WS})$ over the predicates contained in the above databases. Finally, the run-time execution mechanism can be also defined in term of $\mathtt{DATALOG}^{\mathtt{ev!}}$ rules ($\mathbf{P}(\mathcal{WS})$). The logic program consists of a *static definition*, that is a classical DATALOG program with stratified negation, and (of a number of *event definitions*. An event definition consists of the *event declaration* within brackets and of one or more *transition rules*. Two event definitions are:

$$\begin{array}{ll}
\multicolumn{2}{l}{[\mathtt{init(ID)@(T)}]} \\
\multicolumn{2}{l}{\quad \mathtt{run()}\text{++},\ \mathtt{started}_{ID}(),\ \mathtt{ready}_{ID}(\mathtt{ST},[\,],\mathtt{T}) \leftarrow \mathtt{startTask(ST)}.} \\
\multicolumn{2}{l}{[\mathtt{run()@(T)}]} \\
\quad \mathtt{evaluate}_{ID}(\mathtt{Task},\mathtt{L},\mathtt{Duration})\text{++}, & \\
\quad \mathtt{startRunning}_{ID}(\mathtt{Task},\mathtt{L},\mathtt{Server},\mathtt{T}) & \leftarrow \neg\mathtt{unsat}_{ID}(),\ \neg\mathtt{executed}_{ID}(), \\
 & \quad \mathtt{state}_{ID}(\mathtt{Task},\mathtt{L},\mathtt{Ready}),\ \mathtt{available(Server)}, \\
 & \quad \mathtt{executable(Server,Task,Duration)} \\
 & \quad \otimes\ \mathtt{choice((Task),(Server))}\ \otimes\ \mathtt{choice((Server),(Task))}.
\end{array}$$

The first event, called `init`, is an external event which starts a new workflow instance at a certain time, and triggers the event `run()@(T)`. Each time a run event occurs, the system tries to assign the ready tasks to the available servers — as we do not use a particular policy for scheduling the servers, the assignment is made in a nondeterministic way. The predicate $\mathtt{unsat}_{ID}()$ is true if it has been already checked that the workflow instance does not satisfy possible constraints on the overall execution – this check is performed during another event `complete`. The predicate $executed_{ID}()$ is true if the workflow instance has already entered a final state so that no other task need to be performed. Once the tasks are assigned to servers, their executions start, and an event `evaluate` is triggered to continue the analysis of the workflow instance.

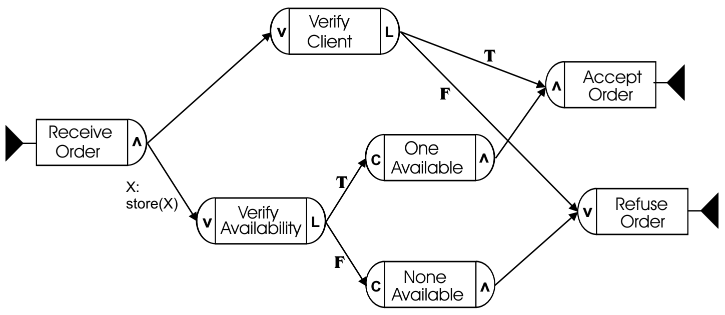

Fig. 1. Example of workflow.

The DATALOG$^{\text{ev!}}$ program can be thought of as a simulation environment for workflow executions, and it is quite intuitive, declarative in the spirit, yet so powerful to cover all the features of current workflow systems.

3 An Example of Usage

For a better understanding of the main concepts of our workflow system, let us consider the following example, which describes a typical process for a selling company.

A customer issues a request to purchase a certain amount of given product by filling in a request form on the browser (task *ReceiveOrder*). The request is forwarded to the financial department (task *VerifyClient*) and to each company store (task *VerifyAvailability*) in order to verify respectively whether the customer is reliable and whether the requested product is available in the desired amount in one of the stores. The task *ReceiveOrder* will activate both outgoing arcs after completion.

Note that the task *VerifyAvailability*, which is instantiated for each store, either notifies to the task *OneAvailable* that the requested amount is available (label 'T') or otherwise it notifies the non-availability to the task *NoneAvailable* (label 'F'). Observe that the task *OneAvailable* is started as soon as one notification of availability is received whereas the task *NoneAvailable* needs the notifications from all the stores to be activated. The order request will be eventually accepted if both *OneAvailable* has been executed and the task *VerifyClient* has returned the label 'T'; otherwise the order is refused.

As pointed out by many authors, see e.g [2], the essential limitation of the approach based on the *control flow graph* lies in the ability of specifying *local* dependencies only; indeed, properties such as synchronization, concurrency, or serial execution of tasks, also called in the literature *global constraints*, cannot be expressed. The example confirms that real world cases often includes properties, which cannot be captured by a graph; in particular, a natural constraint in every instantiation is that the company will try to satisfy the request by looking at the store nearest to the client, in order to reduce transportation costs. This constraint can be easily expressed using our logic programming formalism.

Our system is able not only to model the workflow behavior but also provides a mechanism for querying the model in order to obtain information on its (possible) evolutions. For instance, in our example, the designer may be interested in knowing whether (and when) a given task has been executed for a given predefined scenario. A typical scenario of execution consists of a number of requests that are planned in a certain period of time. For instance, the list

$$\mathtt{H} : [\mathtt{init}(\mathtt{id}_1)@(0), \mathtt{init}(\mathtt{id}_2)@(2), \mathtt{init}(\mathtt{id}_3)@(4), \mathtt{init}(\mathtt{id}_4)@(5)]$$

specifies (with the syntax of our language) that four orders are planned at times 0, 2, 3, 5.

Since for each order, the requested products with the desired quantity are assumed to be taken from a single store, it is obvious that not all the possible schedules will eventually lead to the satisfaction of all the orders. One important feature of our language is the powerful querying mechanism, that can be used for planning and scheduling purposes; for instance, by simple supplying a query of the form

$$\langle \mathtt{H}, [\exists\ \mathtt{executed}(\mathtt{id}_4)@(10)], \mathtt{R}\rangle$$

we can collect in the set of results R, all the steps in the workflow execution that will lead to the execution of the order id_4. Obviously, in the case $R = \emptyset$ we are ensured that there is no way for satisfying such an order, and, hence, we can think at rejecting it in advance, or at planning a new production.

References

1. A. Bonner. Workflow, Transactions, and Datalog. In *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 294–305, 1999.
2. H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic Based Modeling and Analysis of Workflows. In*Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 25–33, 1998.
3. S. Greco, D. Saccà and C. Zaniolo. Extending Stratified Datalog to Capture Complexity Classes Ranging from P to QH. In *Acta Informatica*, 37(10), pages 699–725, 2001.
4. A. Guzzo and D. Saccà. Modelling the Future with Event Choice DATALOG. Proc. AGP Conference,, pages 53-70, September 2002.
5. Eiter T., Leone N., Mateis C., Pfeifer G. and Scarcello F.. A Deductive System for Non-monotonic Reasoning. *Proc. LPNMR Conf.*, 363-374, 1997.
6. P. Senkul, M. Kifer and I.H. Toroslu. A logical Framework for Scheduling Workflows Under Resource Allocation Constraints. In *VLDB*, pages 694-705, 2002.
7. The Workflow Management Coalition, *http://www.wfmc.org/*.