Fluent Logic Workflow Analyser: A tool for The Verification of Workflow Properties

Germán Regis Fernando Villar Nicolás Ricci
Departamento de Computación
Facultad de Cs. Exactas Fco.-Qcas. y Naturales
Universidad Nacional de Río Cuarto
Argentina
{gregis,fvillar,nricci}@dc.exa.unrc.edu.ar

In this paper we present the design and implementation, as well as a use case, of a tool for workflow analysis. The tool provides an assistant for the specification of properties of a workflow model. The specification language for property description is Fluent Linear Time Temporal Logic. Fluents provides an adequate flexibility for capturing properties of workflows. Both the model and the properties are encoded, in an automated way, as Labelled Transition Systems, and the analysis is reduced to model checking.

1 Introduction

The importance of efficiency in companies requires constant improvement to their organizational processes. This has led to the need for expressing such processes, typically referred to as *workflows*, and to the proposal of various workflow languages. There exist many workflow languages, differing in their degree of formalisation (e.g., informal, only with a formal syntax, etc.), their corresponding approaches for workflow description (e.g., declarative or procedural), their expressiveness (e.g., some support advanced conditional routing and some not), their support for automated analysis, etc. An aspect that we consider particularly important is formal semantics. This aspect is crucial for the analysis of models in the language, and is also strongly related to expressiveness, since more expressive languages are more difficult to fully formalise. Furthermore, expressiveness and automation in analysis are typically conflicting aspects, and the design of a good language involves the search of an adequate balance between these aspects. This applies not only to the language in which a workflow is expressed, but also to the language used for describing declarative properties of a workflow. The importance of declarative properties of workflows is acknowledged by several researchers (see for instance [7, 10, 12]). In particular, in [10] a declarative approach to business process modelling and execution is proposed, where declarative behavioural properties of procedural workflow models are a central characteristic.

In this paper, we present a tool for workflow analysis. This tool allows the user to describe properties over a workflow model and verify these properties in an automated way. The formal language used for property specification is a known temporal logic, *fluent linear temporal logic* (FLTL) [5], which is well suited for formally expressing declarative properties of workflows [11].

Basically, FLTL provides a convenient way of expressing state properties of a labelled transition system, associated with the occurrence of events in the system. More precisely, FLTL extends LTL by incorporating the possibility of describing certain abstract states, called *fluents*, characterized by events of the system. As defined in [9], Fluents are time-varying properties of the world, which hold at particular instants of time if they have been initiated by a triggering event (occurring at some earlier instant in time), and have not been terminated by any terminating event since its initiation.

Submitted to:
LAFM 2013

For the verification we employ Model Checking [4], a well established automated method for verifying properties of finite state systems. In order to apply this technique using the *Labelled Transition System Analyser* (LTSA), our tool encodes workflow models as Labelled Transition Systems, following the characterisation presented in [11]. Given a property, the tool guarantees that it is satisfied, or generates violating workflow executions when the property does not hold, as is typical with model checking.

As the input language for workflow description, the tool adopts YAWL (Yet Another Workflow Language) [2]. YAWL is a powerful workflow language based on the use of workflow patterns [3]. It is considered an expressive formalism, as various works dealing with its expressiveness in relation to other business process demonstrate [6]. Indeed, the use of YAWL allows us to ensure the usability of our tool for other workflow languages, in many cases via the use of available automated tools mapping other formalisms into YAWL.

In the remainder of the paper we present the main features of the tool and exemplify their use for describing, specifying and analysing properties of workflow models. Then, we describe the tool as an aggregation of two modules: the encoding manager and the environment that assists in property specification and realizes the integration between the encoder and the LTSA model checker. Finally we conclude with a discussion on our conclusions and future work.

2 Tool usage

Let us illustrate the use of the tool via a simple hypothetical workflow model. The model, as depicted in Fig. 1, describes the process of making a trip. This process begins with the registration task, then the customer can book a flight, hotel or car. When some (may be all) of them are booked with the corresponding task, the customer must pay for them. A simple property of this process may be that once some booking was made, then the payment must take place.

The use of the tool starts by opening the YAWL[1] specification of the workflow. Our tool allows the user to import such a specification, showing it in a graphical way. Once a workflow description is opened, we can add intended properties, in our case the above mentioned one. The properties are formulas specified in FLTL. For the proposed property, one possible specification putting emphasis on the fluents usage, may be `[](someBook -> <>(pay.start))`. The formula establishes that whenever a booking occurs, the payment must take place.

Using *drag and drop*, we can shape the structure of the formula by incorporating the desired operands from the operators bar. The operands can be events of the model, i.e. *start* and *end* task events, or *fluents*. Fluents are binary variables whose values depend on two sets of events: activating and deactivating events. In our case, the operands of the formula are: the fluent `someBook`, that captures the occurrence of some booking, and the reference to the start of the payment process through the `pay.start` event.

In order to specify fluents, we use the fluent definition feature of the tool, starting with the new fluent definition (*main menu* option `add fluent`). Then using the fluent activating or deactivating tools, we set the corresponding model events for each fluent. In case of `someBook`, we select the events that enable this fluent, i.e., the `start` events of the `flight`, `hotel` and `car` booking processes. Note that the *start* events are depicted at the left of tasks and the *end* at the right of them, i.e, work flows from left to right. In a similar way, using the deactivating tool, we can set up the events that turn off the selected fluent.

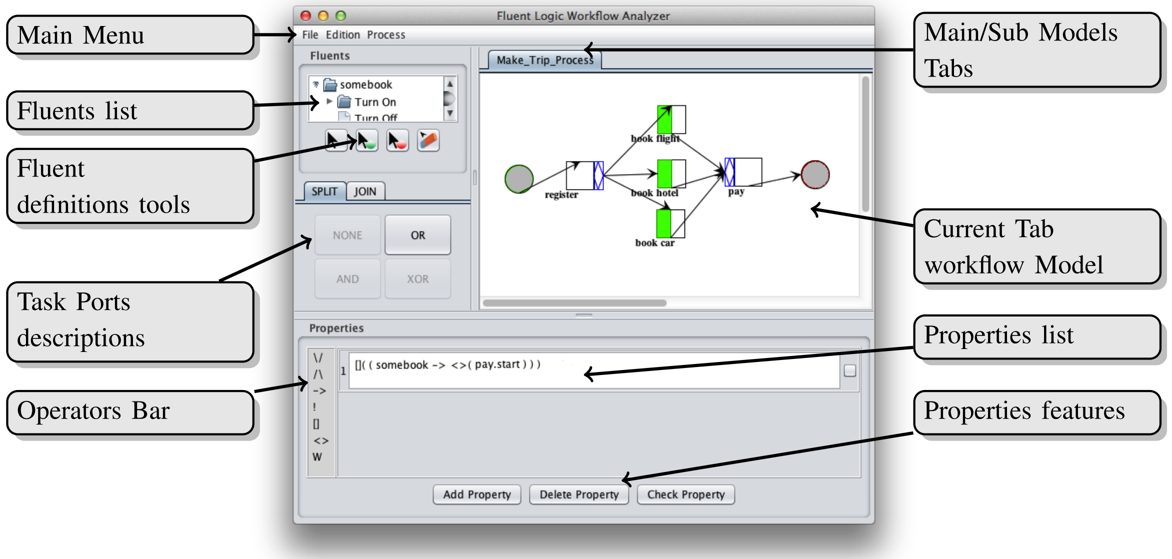
To assist us, the tool provides an auto-complete feature. This feature shows, when we write an operand of a formula, a pop up list containing possible events of the model. The list starts showing the names of tasks or conditions and then a choice for each event about them. Similar to the operators, the

[1]YAWL is a free workflow modelling tool that can be downloaded from `http://sourceforge.net/projects/yawl/`

fluents can be incorporated to the property specification by means of drag an drop from the fluents list to the desired place in the formula. We can of course avoid these assistants and simply type the formula.

Another feature that the tool supports is a property specification assistant, that provides a set of templates, as shown in Fig. 2. These templates allows the user to instantiate a generic property about the system. To use this templates, we can navigate over a list of properties, with each one containing its own description. When a property is selected, for each parameter (operand of the underlying formula) a box and button are displayed for assigning the event or fluent of the model desired. Our sample property corresponds to a *response* property that asserts that, given two activities A and B, "whenever A is executed, then B has to be eventually executed afterwards". If we wish to use this template, instead of the handmade specification, we simply instantiate the template by assigning the fluent `someBooking` and the event `pay.start` as the A and B parameters respectively.

Finally by pressing the `check Property` button, we can verify if the property holds in our workflow model. In order to store the fluents and properties specifications for future use, we can save our current job as a file by using the corresponding menu option.



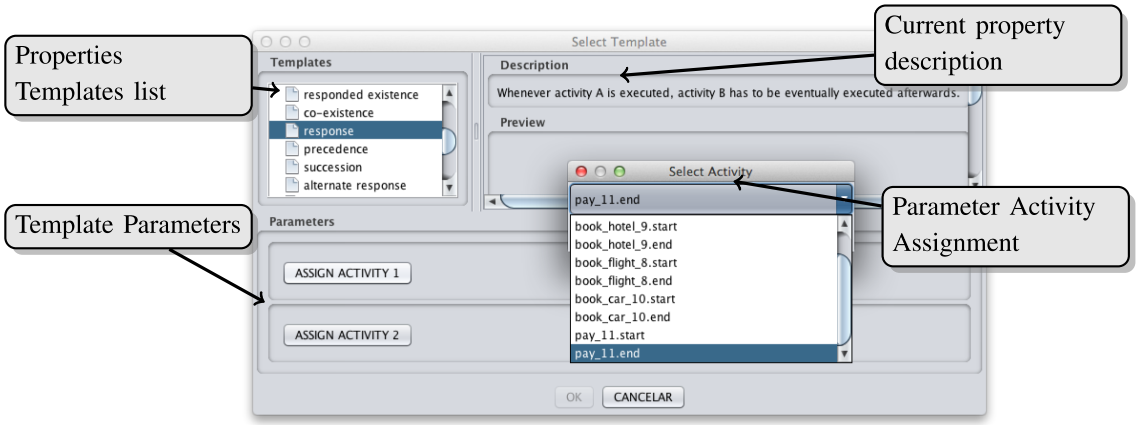
1 []((somebook -> <>(pay.start)))

Figure 1: FLYAnalyser Full view

Note that the sets for activating or deactivating a fluent can be conformed by events corresponding to different sub-workflows, i.e., events of the workflow detailing a composite task. This flexibility allows the user to capture, in a simple way, complex situations in a model, such as for example execution traces between activities.

3 Tool Design

Fig. 3 depicts the architectural design of the process of verifying properties about a workflow model. The process begins with a model of workflow described using the YAWL workflow modelling tool. YAWL is a powerful workflow language based on the use of workflow patterns [3]. In order to verify properties of workflows, in particular those described with YAWL, we develop a tool called FLYAnalyser. Our tool takes as input a workflow model and assists the user to easily and in a graphical way, specify and



Whenever activity A is executed, activity B has to be eventually executed afterwards.

Figure 2: Properties Template wizard

verify properties of the model. The verification process is handled by the LTSA model checker, through a translation of the workflow and properties to a labelled transition system (LTS). Thus we will be able to express behavioural properties of these workflows declaratively, using the FLTL.

The tool has two main modules developed separately: a compiler called YAWL2FSP that encodes a YAWL workflow model into an LTS and the environment FLYAnalyser, which is responsible of the analysis and specification of properties, and handles the integration between the YAWL2FSP and LTSA.

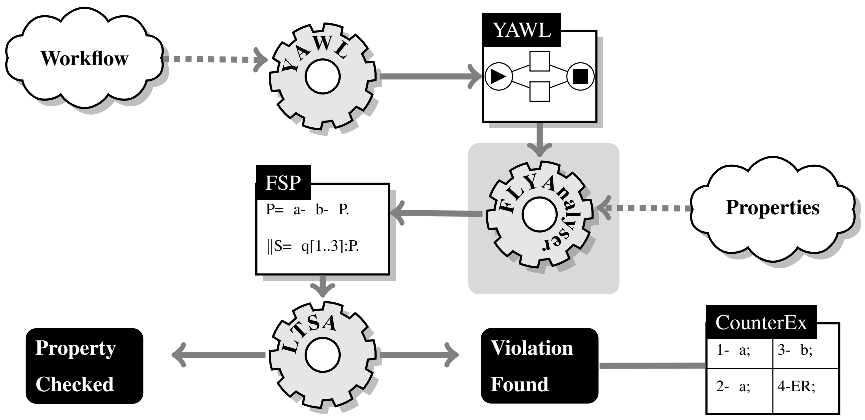


Figure 3: Verification Workflow

3.1 YAWL2FSP

As a module of the FLYAnalyser, we first present an encoding from YAWL models into Finite State Processes (FSP)[8]. FSP is a process algebra that provides us a compact and flexible way to specify LTS, i.e., FSP expressions can be automatically mapped into LTS. Basically, this encoding allow us to interpret YAWL procedural workflows as FSP processes. The encoding will enable us to employ the LTSA model checker for *verifying* behavioural properties of task activities of the workflow specifications. The basic intuition behind the encoding of a YAWL models into FSP is the following. A system's behaviour is characterised by the occurrence of its tasks. In an abstract way, we can capture a task as an entity having some activity in the system between its *start* and *end* events. So, a trace of these events describes a possible execution of the system. In this way, a system's behaviour, is captured by the set of all its execution traces. These traces are obviously constrained according to the control flow of the system.

According to our previous observation, it is straightforward to see that a task activity can be captured by means of a *fluent*, becoming *true* when its *start* event takes place, and turning back to *false* when its *end* event task occurs. In order to capture the behaviour of the workflow's control flow, we will need to introduce appropriate event synchronisations and process compositions, relating the events related to the tasks that conform the workflow. Once we achieve a characterisation of workflows as FSP processes, we can express properties of the workflows by expressing temporal formulas, employing task-related fluents as the basic ingredient.

To formally describe our translation from YAWL into FSP, we consider a formal semantics of YAWL models [6], given in terms of Reset Petri Nets. Taking into account this semantics, we propose an encoding for tasks and conditions. In order to represent a workflow behaviour, we specify how to compose tasks and conditions. In this composition we consider the control flow operators associated with the tasks of the workflow, and provide an encoding for them. Finally, we address especially sophisticated elements of YAWL constructions, such as *cancel regions*, *multiple instance tasks* and *composite tasks*. For a full description of the encoding process, see [11].

3.2 The Environment

The environment is a graphical application written in Java. It can open a workflow model specified with the YAWL tool. These models are saved in files with XML format. The application has the following features:

- It shows the model in a graphical way to help the user understand and analyse the model. In presence of composite task sub-workflows specifications, for each of them, the application generates a tab with the corresponding graphical workflow specification.
- It provides a flexible and agile way to specify fluents by simply clicking on the desired events of the workflow that activates or deactivates them.
- It assists the user in the property specification by means of *drag and drop* operators or fluents, and auto-completing with events of the models.
- It aids the workflow analysis by means of a property template wizard.
- It holds the integration with other modules and tools, in particular, it invokes the compiler to obtain the FSP model of the workflow and composes it with the selected properties. Then, it calls the model checker in order to perform the verification.

Note that the property templates listed by the wizard are those analysed in [10] and they are stored in the file `templates.properties` in XML format. The user can incorporate new templates by adding them into the file, filling the certain required fields.

4 Conclusion and Future Work

We have described the *Fluent Logic Workflow Analyser*, a tool for the specification and verification of workflows and properties of these. The tool focuses on the analysis of *declarative properties* of procedural descriptions of workflows. This tool is publicly available[2].

[2]`http://sourceforge.net/projects/yawl2fsp/`

We have chosen to base our work on YAWL because it has a formal foundation, and it supports a wide range of workflow patterns, providing an expressive environment for workflows specification. The YAWL toolset provides the verification of some properties of workflows such as soundness and deadlock-freedom [1], but it does not provide a suitable flexible language for declaratively expressing other behavioural properties of its models. Our tool complements YAWL toolset in this direction.

In order to analyse the scalability of our tool, obviously bounded by the state explosion problem of the underlying model checking technique, we took as a case study a relatively complex example from [6]. The case study describes the process of order fulfilment which is divided into several phases. The complete model, flatting the composed tasks, has 58 tasks, 30 gates, 36 conditions, 2 cancel regions. The LTS was generated in 0.298 seconds, using 28,96 Mbytes of memory, containing 13164 states and 59722 transitions. Several properties were verified and the time consumption associated with this process did not exceed one second for each one. Due to space restrictions, we do not describe this model here.

As a future extension of the tool, we are working on the feedback of the information provided by the model checker in case of property violations, showing a graphical representation of the sequence of events that produces a property violation. We think that this feature can help the user to understand why the model is wrong with respect to some analysed property.

References

[1] Wil M. P. van der Aalst, Kees M. van Hee, Arthur H. M. ter Hofstede, Natalia Sidorova, H. M. W. Verbeek, Marc Voorhoeve & Moe Thandar Wynn (2011): *Soundness of workflow nets: classification, decidability, and analysis*. Formal Asp. Comput. 23(3), pp. 333–363, doi:10.1007/s00165-010-0161-4.

[2] Wil M. P. van der Aalst & Arthur H. M. ter Hofstede (2005): *YAWL: yet another workflow language*. Inf. Syst. 30(4), pp. 245–275, doi:10.1016/j.is.2004.02.002.

[3] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski & A.P. Barros (2003): *Workflow Patterns*. Distributed and Parallel Databases 14(1), pp. 5–51, doi:10.1023/A:1022883727209.

[4] Edmund M. Clarke, Orna Grumberg & Doron Peled (2001): *Model checking*. MIT Press.

[5] Dimitra Giannakopoulou & Jeff Magee (2003): *Fluent model checking for event-based systems*. In: ESEC SIGSOFT FSE, pp. 257–266, doi:10.1145/940071.940106.

[6] Arthur H. M. ter Hofstede, Wil M. P. van der Aalst, Michael Adams & Nick Russell, editors (2010): *Modern Business Process Automation - YAWL and its Support Environment*. Springer.

[7] Christos T. Karamanolis, Dimitra Giannakopoulou, Jeff Magee & Stuart M. Wheater (2000): *Model Checking of Workflow Schemas*. In: EDOC, pp. 170–181, doi:10.1109/EDOC.2000.882357.

[8] Jeff Magee & Jeff Kramer (2006): *Concurrency - state models and Java programs (2. ed.)*. Wiley.

[9] Rob Miller & Murray Shanahan (1999): *The Event Calculus in Classical Logic - Alternative Axiomatisations*. Electron. Trans. Artif. Intell. 3(A), pp. 77–105.

[10] Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello & Sergio Storari (2010): *Declarative specification and verification of service choreographiess*. TWEB 4(1), doi:10.1145/1658373.1658376.

[11] Germán Regis, Nicolás Ricci, Nazareno Aguirre & T. S. E. Maibaum (2012): *Specifying and Verifying Declarative Fluent Temporal Logic Properties of Workflows*. In: Brazilian Symposium on Formal Methods, SBMF 2012, pp. 147–162, doi:10.1007/978-3-642-33296-8_12.

[12] Peter Y. H. Wong & Jeremy Gibbons (2011): *Formalisations and applications of BPMN*. Sci. Comput. Program. 76(8), pp. 633–650, doi:10.1016/j.scico.2009.09.010.