USING LOTOS IN WORKFLOW SPECIFICATION

Vincenza Carchiolo, Alessandro Longheu, Michele Malgeri
Dipartimento di Informatica e Telecomunicazioni - Facoltà di Ingegneria – Università di Catania
Viale Andrea Doria, 6 - I95125 Catania – Italy – Phone: +39-095-738-2359
E-mail : { Vincenza.Carchiolo, Alessandro.Longheu, Michele.Malgeri } @ diit.unict.it

Keywords: Workflows, Formal description techniques

Abstract Complexity of business processes is getting higher and higher, due to the rapid evolution of market and technologies, and to the reduced time-to-market for new products. Moreover, it is also essential to check workflow (WF) correctness, as well as to guarantee specific business rules. This can be achieved using specific tools within workflow management systems (WFMS), but a formal approach (mathematical-based) is a more effective methodology. Formal description techniques (FDT) based on process algebra allow both to formally describe WF at any level of abstraction, and to formally verify properties as correctness and business rules. In this paper, we apply FDT in production workflows specification using the LOTOS language. In particular, we first show how most recurrent WF patterns can be described in LOTOS, then showing an application example of how WFs can be defined based on LOTOS patterns.

1 INTRODUCTION

Modern business processes are increasing in their complexity due to the global competition, which determines the rapid evolution of both market and technologies, also reducing the available time-to-market for new products.

This scenario makes a critical factor for the success of a company the effective use of information technology for business process management.

Even if Workflow Management (WF) (Cichocki, 1998)(Leymann, 1999) technologies significantly improve business automation, many high-value (e.g. safety critical) business processes needs a more formal approach for specification and verification of workflows (e.g. absence of deadlock, abnormal termination, or satisfaction of specific business rules).

Current WF Management Systems (WFMS) actually offer some mechanism to specify the flow of control (Lawrence, 1997), but a complete formal approach is generally not present, mainly due to the application-oriented nature of WFMS tools.

The idea of using formal techniques (Clarke, 1996) in workflow specification comes from this gap between processes and tools used to manage them. It is indeed important that mapping the requirements on to a given implementation is not error-prone, e.g. introducing errors or unexpected behaviours due to an incomplete or ambiguos specification. Formal techniques based on process algebra allow both to provide a formal workflows specification and to perform a verification about some desired properties.

In this paper, the application of formal techniques to workflow specification is considered. In particular, since we want to exploit to what extent formal techniques can be effectively used to specify a generic workflow, we consider WF patterns (Van der Aalst, 2000-2), (Van der Aalst, 2000-1), i.e. most recurrent pre-defined schemas specifying the flow of sequence. Since a generic workflow is defined using such basic building blocks, it is significant to provide their formal specification, in order to allow the application of FDTs in as many cases as possible.

The application of FDT to workflow specification is more interesting since such techniques are not currently extensively adopted within WF context.

In this paper, we focus on the LOTOS language (ISO, 1988), an FDT based on process algebra used for the design and validation of distributed systems.

2 WORKFLOW AND PATTERNS

A *workflow* is the abstraction of a real-world *business process* (Leymann, 1999). WFs aim at defining the *process logic*, i.e. a process description made in terms of *working step*s, also known as *activities*, which are the atomic operations to be performed. The *control flow* is the activities execution sequence,and represents paths joining activities, including the conditions that define which paths have to be taken if more branches comes out from an activity (split) or go into an activity (join). Due to the central role of process logic, it is important to provide an effective WF specification methodology, in order to easily and rapidly specify even complex workflows. To meet these requirements, in (Van der Aalst, 2000-2) a systematic analysis over several different real workflow requirements has been conducted, in order to determine which most recurrent sequences of activites are generally used, arranging them into patterns. Such patterns represent the abstraction of activities sequences (Riehle, 1996), and are composed when defining new workflows. Patterns are grouped into *basic control flow*, e.g. simple sequence, split, or join of activites; *advanced control flow*, where multiple active branches and synchronization are considered; *structural* patterns, e.g. loops; *temporal* and *state-based* patterns; *inter-workflow synchronization* patterns.

3 LOTOS FOR WF MODELING

The choice of LOTOS language for workflow modeling comes from considering the most two commonly used FDTs classes (Clarke, 1996). i.e. FDTs using temporal logic to express the properties a system should have (CTL, LTL), and FDTs in which specification is made in terms of a high-level model of the system (First-order predicate logic, CSP, UNITY).

When specifying a system, both FDT classes can be used, though while the former simply provide a set of system properties (i.e. requirement-based approach), the latter specifically concerns the system's behavioural semantics, i.e. considering the execution order of all operations performed by the system, providing them in an event-based fashion; this makes the second class of FDTs more suitable to evaluate the workflows and their behaviour.

Among FDT belonging to the second class, we choose the LOTOS language (ISO, 1988)(Quemada, 1987) for its subsystems composition support. Besides, LOTOS naturally supports concurrency, making it possible to model systems made up of various parts which evolve in parallel, a situation typically present in workflow models. Finally, LOTOS is widely used in system design context; this guarantees the use of a standardized technique, whereas several graphical tools, as well as libraries, are currently available for LOTOS, thus providing more flexibility for workflow modelers.

LOTOS (Language Of Temporal Ordering Specification) (ISO, 1988) is an FDT used for the description of concurrent and/or distributed systems (Bolognesi, 1987).Its basic idea is that the system is viewed as a black-box which interacts with the environment by means of events, whose occurrence is described by LOTOS behavioural expressions. The language has two components: the description of processes behaviour (Milner, 1980)(Hoare, 1985) and the description of its data structure and expressions (Mahr, 1985). Here we focus on LOTOS behavioural part, which is suitable for WF specification (control flow). In LOTOS a system is described in terms of processes; the system itself may consist of a hierarchy of processes (subprocesses) which interact with each other and with the environment. The atomic forms of interaction with the outside world are called events. The syntax of a process in LOTOS is:

process <*process-identifier*> <*parameter-list*> :=
<*behaviour-expressions*>
endproc

where: <*process-identifier*> is the name to be assigned to the process; <*parameter-list*> is the list of events through which the process can interact with the environment; <*behaviour-expressions*> are LOTOS expressions which define the behaviour of the process. A special process (stop) models a inactive process. A detailed specification of LOTOS operators used throughout this paper can be found in (ISO, 1988).

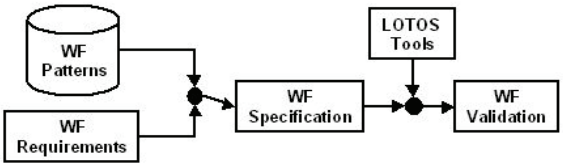

Figure 1: Schema of proposed approach

4 WF PATTERNS WITH LOTOS

As mentioned in sec.1, our goal is the use of LOTOS to improve WF specification; since a generic workflow can be defined through patterns, we consider LOTOS patterns description. In (Longheu, 2001), all patterns are described; here we focus on a restricted set of patterns used to provide a real case example (sec. 5).

The integration of patterns and FDTs allows on one hand to provide workflow modelers with a set of useful predefined schemas to easily and rapidly specify the flow of sequence, while on the other it guarantees correctness and verification of business rules over that specified sequence.

The proposed approach is schematized in Fig. 1: LOTOS-based patterns can be functionally viewed as a database from which basic elements are retrieved, and subsequently properly arranged to build workflow specification from given requirements. The composition can be easily performed thanks to LOTOS subsystems composition support, while LOTOS concurrency support allows to model parallel activities, a situation typically present in workflows.

Then, the specification is formally validated thanks to the availability of several LOTOS tools. The resulting WF can finally be instantied or further revised.

To populate a LOTOS patterns database, in the following we consider a set of patterns, which are informally described and formally specificated with LOTOS. Note that a finite set of activites will be used in specifications to model different patterns; this is simply to allow faster simulation of patterns, i.e. it does not affect the generality of patterns. In the following, emphasys is given to synchronization related issues, effectively supported by LOTOS.

AND-Split pattern models the concurrent execution of a set of activities (B,C,D) which run independently (no synchronization is actually required), all after the completion of a main activity A.

The LOTOS specification in Fig. 2 contains the activity A, followed by concurrent activities B, C, D; note that each activity is mapped onto a LOTOS process, even if this is not mandatory, rather the high abstraction provided by LOTOS allows to model with a process both a group of activities, as well as single details of a single activity, according to the required level of specification.

Note that the process in the workflow sense should not be confused with LOTOS processes: while the former is used to indicate the set of activities performed by a company (see sec. 2), the latter is the abstraction provided by LOTOS to model any set of operations, as stated proviously.

Besides, based on the same principle of abstract black-boxes, no internal details of each activity are actually given, in order to provide patterns specification valid for a general workflow; when needed, however, it will be possible to go into further details for one or more activities by simply modeling them with LOTOS subprocesses, following a top-down approach.

Note that in the AND-Split pattern the absence of synchronization leads to the absence of interaction points among processes B, C, D; this is achieved through the interleaving operator ||| (sec. 3.2).

```
specification AND_SPLIT_MODEL
[begin_A,end_A,begin_B,end_B,begin_C,end_C,begin_D,end_D]: exit
behaviour
 (i;A[begin_A,end_A])  |  [end_A]  |  ANDSPLIT[end_A,begin_B,begin_C,begin_D]  |
 [begin_B,begin_C,begin_D]  |  (  B[begin_B,end_B]  |||  C[begin_C,end_C]  |||  D[begin_D,end_D]  )
where
  process AND_SPLIT[end_a,begin_b,begin_c,begin_d]: exit:=
  DISPATCH[end_a] >>  ( (i;DISPATCH[begin_b])  |||  (i;DISPATCH[begin_c])  |||  (i;DISPATCH[begin_d]))
    where      process DISPATCH [event]: exit:= event;exit  endproc
  endproc
endspec

specification AND_SPLIT_MODEL (alternative)
[begin_A,end_A,begin_B,end_B,begin_C,end_C,begin_D,end_D]: exit
behaviour
          A >> ( B ||| C ||| D )
where
  process A [begin_A,end_A]; exit := ...
  process B [begin_B,end_B]; exit := ... [similarly, process C and D are defined]
```

Figure 2: And-Split pattern

```
specification XOR_SPLIT_MODEL
[begin_A,end_A,dec,decision,begin_B,end_B,begin_C,end_C,begin_D,end_D]: exit
library      BOOLEAN,NATURAL        endlib
behaviour
(i;A[begin_A,end_A,dec,decision])   |   [decision]   |   XORSPLIT[begin_B,begin_C,begin_D,decision]   | [begin_B,begin_C,begin_D]   |   (   B[begin_B,end_B]
            []   C[begin_C,end_C]   []   D[begin_D,end_D] )
where
  type three is sorts nat opns
     2(*! constructor *),   3(*! constructor *),   4(*! constructor *): -> nat
  endtype
  process XORSPLIT[begin_b,begin_c,begin_d,decision]: exit:=     decision?d:nat;
     (([d=2]->i;DISPATCH[begin_b])   []   ([d=3]->i;DISPATCH[begin_c])   []   ([d=4]>i;DISPATCH[begin_d]) )
  endproc
  process A [begin_A,end_A,dec,decision]: exit:=  begin_A;  dec?d:nat;  end_A;  decision!d;  exit  endproc
endspec

specification XOR_SPLIT_MODEL (alternative)
 ...
behaviour
   (i;A[begin_A,end_A,decision])  |  [decision]  |  XORSPLIT[decision]  >>  accept d:nat in
   (  ([d=2]->B[begin_B,end_B])  []  ([d=3]->C[begin_C,end_C])  []  ([d=4]->D[begin_D,end_D])  )
where
  process XORSPLIT[decision]: exit(nat):=  decision?d:nat;  exit(d)   endproc
  process A [begin_A,end_A,decision]: exit(nat):=  begin_A;  decision?d:nat;  end_A;  exit(d)   endproc
endspec
```

Figure 3: Xor-Split patterns

Finally, note that the AND-Split is a composition of activities which will end just when all its activities (i.e., B, C, D) will terminate.

The process specified in Fig. 2 aims at highlight the presence of AND_SPLIT pattern. However, LOTOS allows to adopt more specification styles, according to the aspect which needs to be highlighted, always providing the guarantee that all these specifications are formally equivalent. In the same Fig. 2, the pattern is specified in a simplified alternative form, aiming at highlight the sequence of activities rather than their synchonization.

XOR-split pattern describes the common situation in workflows when, based on control data and/or on external decisions, a specific path is chosen over a set of possible paths. Referring to Fig. 2 (the difference with And-Split is just in behaviour), the first activity (A) is executed, and then just an activity over a set of activities (B,C,D) will be executed, depending on the result of a condition.

Note that, as for AND-Split pattern, the description is made with a finite set of activities, but it can easily generalized to a larger set (i.e. $A_1 \ldots A_n$ instead of B,C,D). The LOTOS specification of XOR-Split pattern is shown in Fig. 3; the use of parallel operator after the XORSPLIT process at the beginning of specification imposes a parallelism in a situation in which it is not really required. Such specification actually just aims at highlight patterns by modeling them as independent modules used to compose a workflow. To show how LOTOS allows different styles of specification, in Fig. 3 we propose the same pattern in a slightly different style, though keeping highlighted the XORSPLIT process; note in figure just modified parts of the specification are shown.

Milestone pattern (Jablonski, 1996) describes the situation in which a given activity is enabled just if a specified set of conditions (milestone) has been reached and not expired yet. For instance, an activity A may be enabled just if an activity B has been finished and a third C has not started yet; from a temporal ordering point of view, this means A can be enabled only within the interval after the execution of B and before C starts.

As an example, consider the complaints management workflow, shown in Fig. 4. A complaint is made by a person and registered, then a form to be filled is sent to that person while at the same time the complaint is evaluated. If the form is returned within two weeks, it will be examined, otherwise it will be discarded. Meanwhile, the complaint is evaluated and a decision whether to consider it or not is made, but the execution of this decision is anyway delayed until the response about the form has been provided. If

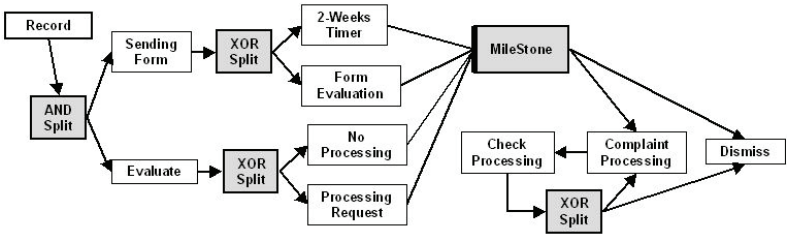

Figure 4: Milestone pattern

indeed the timer has been expired, no processing is performed, and the complaint will be discarded, otherwise the decision is considered. If this decision is negative, the complaint will be discarded again, whereas if a positive response comes from the decision, the processing of the complaint will occur, following a cycle until its termination; at that time, the complaint will be stored into archives (dismissed). The activity COMPLAINT_PROCESSING, which processes the complaint, uses the milestone pattern, obtained through a combination of an AND-Split with a XOR-Split. Indeed, the process based on this pattern must wait for the completion of a set of activities, i.e. TWO_WEEKS_TIMER, FORM_EVALUATION, and PROCESSING_REQUEST. The last is a dummy activity used to separate the state checking activity from the real complaint processing (performed by COMPLAINT_PROCESSING). The MILESTONE process is used to determine if COMPLAINT_PROCESSING will be actually executed; if not, it will select the next activity RECORD (store into the archive). The milestone pattern will be used in sec. 5.

Messaging communication pattern (Fig. 5) describes the situation when information are exchanged between different activities, which are generally considered belonging to distinct workflows: a message is sent from a sender in a workflow to a receiver in another workflow. In this pattern there is only one sender and one receiver, so neither of them needs to be explicitly named, and no information about destination must be placed inside messages; if more receivers are present, simultaneous messages communication from the sender would have place, while if more senders are present with one receiver, this should synchronize all incoming messages. These situations are described in two additional patterns (bulk messages sending/receiving, see (Van der Aalst, 2000-1) for details). The fact that messages are sent from only one sender to only one receiver is modeled through the *scopem* event, which is shared between the two workflows and allows to synchronize sending and receiving operations, actually performed by SENDER and RECEIVER activities.

5 APPLICATION EXAMPLE

In this section all patterns introduced in sec. 4 will be interconnected together in order to present a real application example. Note that the composition of patterns is naturally provided by LOTOS, as specified in sec. 3 (subsystems composition support); using this feature it will be possible to use patterns to create even complex workflows.

The example considered is that described in milestone pattern (sec. 4.3), i.e. the workflow for processing a complaint inside a company. Referring to Fig. 4, patterns needed to model this flow of control are AND-Split, XOR-Split, and Milestone.

We focus in particular on the MILESTONE process based on the milestone pattern, which is used to definitively decide about the actual processing of the complaint; it then represent a deadline for the COMPLAINT_PROCESSING activity. This activity should then ask the MILESTONE about the current state by sending it a signal; based on the response, COMPLAINT_PROCESSING will wait, terminate or execute. Actually, the implementation of this behaviour is slightly different. Indeed, when the COMPLAINT_PROCESSING activity will be executed, it will also involved (see Fig. 5) in a loop, modeled with LOOP activity, therefore the readability could be affected. To avoid this, we decided to separate the task of asking the MILESTONE about the current state from the actual processing of the

```
specification                                                                    MESSAGING_COMMUNICATION_MODEL
[begin_workflow1,end_workflow1,begin_workflow2,end_workflow2,scopem,processing,preparation,                send_message,
receive_message,evaluate]:exit
behaviour
WORKFLOW1[begin_workflow1,processing, send_message,scopem,end_workflow1]  |  [scopem]  |
WORKFLOW2[begin_workflow2,preparation,scopem,receive_message,evaluate,end_workflow2]
where
  process WORKFLOW1
  [begin_workflow1,processing,send_message,scopem,end_workflow1]: exit:=
   begin_workflow1;processing; SENDER[send_message,scopem]  >>  end_workflow1;exit
  where
     process SENDER [send_message,scopem]: exit:=  send_message;  scopem;  exit   endproc
  endproc
  process WORKFLOW2
  [begin_workflow2,preparation,scopem,receive_message,evaluate,end_workflow2]: exit:=
   begin_workflow2;preparation;        RECEIVER[scopem,receive_message]  >>  evaluate;end_workflow2;exit
  where
    process RECEIVER [scopem,receive_message]: exit:=  scopem; receive_message; exit  endproc
  endproc
endspe
```

Figure 5: Messaging communication pattern

complaint, so two dummy activities are introduced, i.e. PROCESSING_REQUEST and NO_PROCESSING, simply used to carry the decision about the complaint evaluation up to the MILESTONE. In this way, the specification is easier since we simply have to wait for two results (modeled with AND-Split in Fig. 5), and based on them decisions are taken (modeled with XOR-Split in the same figure). The complete specification for complaints processing is presented in (Longheu, 2001).

Note that, as mentioned in previous sections, different styles could be adopted when providing a specification; here, we decided to highlight the use of patterns as independent modular units used to compose the workflow. To evaluate the specification described above, as well as the single patterns, we performed a simulation using the CADP tool (Fernandez, 1996)(Garavel, 1997). For simulation purposes, some parts of the specifications has been simplified, for instance using a reduced number of activities in AND-Split and XOR-Split patterns, as mentioned in sec. 4. Using the CADP toolbox, we verified the correctness of specification, and some basic properties (e.g. the absence of deadlock and livelock).

6 CONCLUSIONS

In this paper, the application of formal techniques for workflow specification has been considered. The LOTOS language has been used to describe some workflow patterns, which can be considered representative for most workflow scenarios. Such patterns have been described and interconnected together to build a real workflow application example.

Main considerations that comes from our attempt of using FDTs (in particular LOTOS) in workflow context are:

LOTOS is suitable for workflow specification, thanks to its features of systems composition support, concurrency support, and large availability of tools.

LOTOS allows to adopt different styles for the specification of workflow, in order to highlight a specific desired characteristic.

LOTOS is a simple yet powerful technique, since on one hand no significant effort is needed to describe process and their interconnection to build a control flow, whereas formal correctness and/or specific business rules can be verified using LOTOS tools.

Several future issues must be addressed. First, to improve simulation results, on one hand performing a simulation over all patterns (Longheu, 2001), whereas mechanisms to evaluate workflow specific properties (Leymann, 1999) should be developed.

Finally, another important issue is the definition of business rules through FDTs, in order to allow workflow modelers to add any needed constraint over their workflow with the guarantee it can be formally satisfied.

REFERENCES

Bolognesi, 1987. Introduction to the ISO Specification Language LOTOS. Computer Networks and ISD Systems, (14) 25-59.

Clarke, 1996. Formal Methods: State of the Art and Future Directions. CMU Computer Science Technical Report CMU-CS-96-178.

Cichocki, 1998. Workflow and process automation: Concepts and Technology, Kluwer.

Fernandez, 1996. CADP, Protocol Validation and Verification Toolbox.

Garavel, 1997. CADP, Status, Applications, and Perspectives.

Hoare, 1985. Communicating Sequential Processes. International Series in Computer Science, Prentice-Hall.

ISO, 1988. Inform. Processing Systems, OSI, LOTOS, IS-8807 - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.

Jablonski, 1996. Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press.

Lawrence, 1997. Workflow Handbook. Workflow Management Coalition. John Wiley and Sons.

Leymann, 1999. Production Workflow: Concepts and Techniques. Prentice Hall.

Longheu, 2001. Specification and simulation of Workflow patterns through Formal Methods, Technical Report DIIT-01-447.

Mahr, 1985. Fundamentals of Algebraic Specifications. 1 EATCS Monogr. on Comp. Science, Springer-Verlag.

Milner, 1980. Calculus of communicating systems. LCNS 92, Springer-Verlag.

Quemada, 1987. Introduction of Quantitative Relative Time into LOTOS. Workshop on Protocol Specification, Testing and Verification, VII North Holland.

Riehle, 1996. Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems, 2 (1):3-13.

Van der Aalst, 2000-1. Advanced Workflow Patterns. In Proceedings Seventh IFCIS - CoopIS.

Van der Aalst, 2000-2. Workflow Patterns. BETA Working Paper Series, WP 47, Eindhoven University.