Workflow management : net-based concepts, models, techniques, and tools (WFM '98) : proceedings of the workshop, June 22, 1998, Lisbon, Portugal

Eindhoven University of Technology
Department of Mathematics and Computing Science

Proceedings of the workshop on
Workflow Management:
Net-based Concepts, Models, Techniques, and Tools

(WFM'98)

June 22, 1998
Lisbon, Portugal

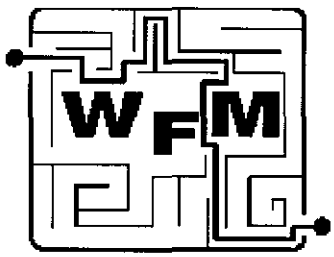
edited by W. v.d. Aalst

ISSN 0926-4515

editors: prof.dr. R.C. Backhouse
prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Reports 98/07
Eindhoven, June 1998



Workflow Management:
Net-based Concepts, Models, Techniques, and Tools

June 22, 1998
Hotel Costa da Caparica,
Lisbon, Portugal

Edited by Wil van der Aalst

Proceedings of the workshop on

Workflow Management: Net-based Concepts, Models, Techniques, and Tools

(WFM'98)

June 22, 1998
Hotel Costa da Caparica,
Lisbon, Portugal

Organized by

W.M.P. van der Aalst
(Eindhoven University of Technology)

G. De Michelis
(University of Milano)

C.A. Ellis
(University of Colorado)

Preface

Welcome to the workshop on *Workflow Management: Net-based Concepts, Models, Techniques and Tools* (**WFM'98**). This workshop has been organized to discuss the application of formal methods to the design, analysis and execution of work processes. WFM'98 is part of the 19th International Conference on Applications and Theory of Petri Nets. Therefore, it is not a surprise that most of the papers use Petri nets as a design language. The contributions show that Workflow Management (WFM) and Business Process Reengineering (BPR) are challenging application domains for Petri nets. On the one hand, people in industry are in need of concepts, methods, techniques and tools to support WFM/BPR efforts. On the other hand, Petri nets are a proven technology to describe and analyze business processes. Therefore, Petri nets seem to be a good candidate for becoming a standard technique for the modeling of workflows.

We received 21 paper submissions from more than 10 countries. Each of these papers has been reviewed by three reviewers. Based on these reviews 12 high quality papers have been accepted for presentation at the workshop. These papers are included in the proceedings. Topics addressed by the papers presented at the workshop:

- process modeling techniques for workflow management
- design and analysis of workflow processes
- business processes reengineering
- verification of workflow procedures
- performance analysis
- software architectures for workflow management
- coordination languages
- workflow management systems
- business process support systems

The presentations have been grouped into four sessions: (1) Workflow modeling and specification, (2) Verification of workflow specifications, (3) Adaptive workflow, and (4) Organizational context and applications. We hope that the presentations will lead to stimulating discussions and new ideas for future research directions.

To conclude, we would like to thank the authors for submitting excellent papers and the reviewers for their comments and constructive suggestions. We would also like to thank the local organizers of UNINOVA. In particular we would like to thank Luis Gomes for taking care of numerous organizational matters. We, the organizers of this workshop, are convinced that these efforts will help to make WFM'98 a successful event.

Wil van der Aalst
(Eindhoven University of Technology, The Netherlands)
Giorgio De Michelis
(University of Milano, Italy)
Skip Ellis
(University of Colorado, USA)

Table of Contents

A Workflow Specification Environment

Pierre Azéma, François Vernadat, Pierre Gradit

LAAS/CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4
{azema, vernadat, gradit}@laas.fr

Abstract

This paper proposes a workflow specification formalism, based upon reactive objects. This formalism is valid for several paradigms such as message passing or agent oriented programming, and consequently may describe distinct workflows, that is flows of electronic documents and/or human activities. A contribution is the introduction of generic modules for determining specific architectures, e.g. specific multicast protocols, or agent instance creation. A basic workflow, for object collecting, is introduced as illustrative example. Several architecture definitions are then proposed for this example : static, hierarchical and dynamic architectures.

keywords: Communicating Agents, Predicate/Transition Nets, State Space Analysis.

1 Introduction

A flexible workflow specification environment is proposed. A first motivation is to handle distinct points of view, in order to identify properties which have to be fulfilled whatever the implementation policy. In the case of computer supported cooperative activities, two complementary aspects may be considered: document circulation and people circulation. Either the documents move and message passing protocols have to be implemented; or people move and agent oriented programming is suitable. Both points of view are worth to be considered.

A second motivation is the processing of dynamic system configurations. A configuration is considered dynamic when, according to task execution, the number of involved agents is changing, as in the case of group membership, or dynamic resource allocation.

Several features characterize the proposed workflow specification environment : rapid prototyping, modularity, communication primitives. The derived specifications are executable, resulting into a **rapid prototyping** facility. This allows an early debugging through a step by step execution. Furthermore, a

verification processing is easily initiated, as far as the reachable state space may be enumerated.

The entities involved within the workflow, either agents or preformatted documents, are considered as reactive elements, that is they have to react on line to external events. The behavioral characteristics of these entities, their functional attributes, are encapsulated within modules. This **modularity** leads to a rather easy scalability, that is the size of the specified system is under control by considering either a limited amount of components, for analysis purpose, or a large number of components for implementation purpose.

In order to separate the concerns about behavior and architecture, the **communication primitives** have to be powerful. The proposed communication facilities are similar to actor language [Agh86] or to protocol specification language [CCI92]: asynchronous communication through FIFO queues. Other built-in primitives include synchronous (rendez-vous) messages, and pattern matching via logic unification.

In the next section, the VAL formalism is introduced by means of a generic producer-consumer example. Section 3 presents the application: the collecting processing of distributed objects, according to two versions: message passing and agent mobility. A hierarchical version of the application is presented in Section 4. Section 5 deals with verification.

2 VAL Description

2.1 Overview

VAL formalism is based on Logic Programming and Predicate/Transition Net [Gen91] allowing the description of dynamic systems of communicating and mobile agents [VAL95]. Agents are the basic ingredients of the formalism. They are active elements of the system: they communicate with partners, disappear or create new agents. Agent Behavior is described by an extended Predicate/Transition Net. Proposed extensions concern communication and dynamism.
The communication operates directly between agents as in Actor languages [Agh86] according to their acquaintances.

The system is dynamic because an active agent may create new agents or(and) disappear by stopping its activity. Every agent may create new agents of any class.

The system description is performed by means of a PROLOG-like language.

A self-referencing mechanism is available. An agent knows its own identity: attributes *@Class* and *@Ref* allow an agent for referencing its class and its reference. Two types of components are considered: agents and structures. Structures are passive elements, that is without behavior. They declare the interconnections between the agents which constitute a configuration, and the agents involved in such a configuration share these interconnections. A structure may define the architecture of an application. Agents are the active system

elements: they are autonomous, they have attributes, methods and behaviors. They send and receive messages, they may disappear and/or create new agents.

A class behavior is depicted by a set of transitions, where self-referencing, rendez-vous, on-line creation are allowed.

Predicates are used for the state description: an agent state consists of a set of predicates. The system state consists of the set of agent states, the content of the input message queues.

Agent behavior

Each transition is specified by a specific clause whose the distinct fields are the followings.

Trans	introduces the name of the transition,
From	refers to preconditions (predicate list),
To	gives the postcondition (predicate list),
When	defines a message reception on a given port,
Send	introduces the receiver agent (class and identifier), the reception port and the message to be sent,
Rdv	declares a rendez-vous,
Create	declares new agents to be created, and their initial state,
Exit	indicates whether the current agent disappears or not.
Cond	introduces local constraints (conditions to be fulfilled).

Communications Patterns For communication structuring purpose, a **topic**, that is a gate or an interaction point, is associated with any message.

- **Elementary synchronization**, and **emission** are specified by 4-tuple $< Class, Ref, topic, message >$ where $Class, Ref$ represent the synchronization partner, *topic* is the communication interaction point, *message* stands for the message content.

- **Elementary reception** is specified by a pair $< topic, message >$ where topic is the name of the message queue and *message* refers to the first queue element.

An **Elementary creation** is specified by a 3-tuple $< Class, Ref, initial >$ where $Class, Ref$ represents the new instance identity and *initial* is the initial state.

Firing Rule: A VAL transition (instance) is **firable** when pre-conditions hold, the expected messages are present and the needed synchronisations are possible. The transition firing removes the preconditions and adds the post-conditions, consumes the received messages, issues new messages and creates agent instances as specified by the designer.

2.2 Producer Consumer Example

As preliminary step, a simple request acknowledgement protocol is considered. Agent consumer send a request towards agent supplier, which returns the required item. Two behavior classes are introduced : *consumer* and *supplier*. Class *consumer* consists of two transitions : *order*, *receive*, while class *supplier* is reduced to single transition *deliver*. Instances of these classes are characterized by class identifiers and their initial states.

In addition to the behavior description, for running a specification, two other data files are needed in order to declare a system: file *structure* specifies the needed agent classes, and possibly local data processing ; file *configuration* defines the set of initial agent instances and their initial states. The former system consists of two agent classes *consumer* and *supplier*, and an initial configuration has to be defined. A consistent configuration may be composed of a consumer and of the associated supplier, that is the supplier which delivers the required items.

With respect to the case consumer-supplier, an initial configuration may consist of consumer instance c and supplier instance *couvert*. The initial state is then the following:

$consumer\_c([request([(couvert, 1])], supplier\_couvert([idle, stock(2)])$

that is consumer c is ready to order quantity 1 of product *couvert*, while supplier *couvert* is *idle* and the available quantity in *stock* is equal to 2.

Several other configurations could be considered: several consumers ordering the same product, or a single consumer ordering several products. These configurations are easily declared at initialization.

A consumer behavior is depicted by transitions *order* and *receive*. These two transitions are specified by parameterized clauses. For transition firing, these clauses have to match with the current state, A global state is composed of the component states, and of the content of message queues. For each transition, an example of the instanciated code is given.

Generic code		*Instantiated Code*	
TRANS	*order*	TRANS	*order*
FROM	$request(\_product, \_num)$	FROM	$request(couvert, 1)$
SEND	$(supplier, \_product,$	SEND	$(supplier, couvert,$
	$req(@class, @ref), \_num)$		$req(consumer, c), 1)$
TO	$wait(\_num)$	TO	$wait(1)$
END		END	

Transition *order* is enable by precondition $request(\_product, \_num)$, that means that there exists, for the current state, a substitution of logic variables $\_product$ and $\_num$. The occurrence of this transition send to agent whose class and identifier are *supplier* and $\_product$ respectively, at interaction point $req(@class, @ref)$, message $\_num$, and changes the current state of agent *consumer* from $request(\_product, \_num)$ to $wait(\_num)$. It must be noticed that the (chosen) way to declare the interaction point introduces a message signature. The supplier will know the sender identity.

From the initial state, the reached global state is then the following:

$consumer\_c([wait([1])]), supplier\_couvert([idle, stock(2)])$

$input(supplier\_couvert(req(consumer, c), [[1]]))$

The supplier state is left unchanged, while on port $req(consumer, c)$ of the supplier, the input file contains message 1.

TRANS	*receive*	TRANS	*receive*
FROM	$wait(\_num)$	FROM	$wait(1)$
WHEN	$(\_product, \_num)$	WHEN	$(couvert, 1)$
TO	*happy*	TO	*happy*
END		END	

Transition *receive*: from state $wait(\_num)$, when interaction point *_product* offers message *_num*, transition *receive* is enable, the firing of this transition reset agent *consumer* into state *happy*.

The supplier behavior is depicted by transition *deliver*. The current state of agent *supplier* is depicted by predicate $stock(\_s)$, that is number *_s* of items is available. By receiving an order of quantity *_num* of items, this amount will be send back, if two conditions, introduced by keyword *COND*, are fulfilled: the required quantity is less than or equal to the available stock, the next stock value is decremented by the just delivered quantity.

TRANS	*deliver*	TRANS	*deliver*
FROM	$stock(\_s)$	FROM	$stock(2)$
WHEN	$(req(\_class, \_ref), [\_num])$	WHEN	$(req(consumer, c), [1])$
COND	$\_num =< \_s,$	COND	$1 =< 2,$
	$\_ns$ *is* $\_s - \_num$		1 *is* $2 - 1$
SEND	$(\_class, \_ref, @ref, \_num)$	SEND	$(consumer, c, couvert, 1)$
TO	$stock(\_ns)$	TO	$stock(1)$
END		END	

2.3 Rendez-vous

The former example could be implemented by means of a strong synchronization between consumer and supplier, that by rendez-vous: the consumer issues the request and simultaneously receives the answer from the supplier. The supplier, by receiving the request, immediately delivers the object.

The following two transitions concurrently fire: *request* on the consumer side, *deliver* on the supplier side. Each partner must precise class and identity of the other, while terms of port and message have to match. The partner states change from preconditions to postconditions.

TRANS	*request*	TRANS	*request*
FROM	$request(\_product, \_num)$	FROM	$request(couvert, 1)$
RDV	$(supplier, \_product,$	RDV	$(supplier, couvert,$
	$port, \_num)$		$port, 1)$
TO	*happy*	TO	*happy*
END		END	

TRANS	*deliver*	TRANS	*deliver*
FROM	$stock(\_s)$	FROM	$stock(2)$
COND	$\_num =< \_s$,	COND	$1 =< 2$,
	$\_ns$ *is* $\_s - \_num$		1 *is* $2 - 1$
RDV	$(consumer, \_consumer,$	RDV	$(consumer, c,$
	$port, \_num)$		$port, 1)$
TO	$stock(\_ns)$	TO	$stock(1)$
END		END	

3 Collecting Strategies

In this section, message passing and agent creation are considered. The producer consumer case is extended. The product may not consist of a single object, the product may be composed of several objects which are separately ordered: a *couvert* is composed of $knife$ and $fork$. The resulting requirement is the online computation of message patterns.

3.1 Communication and Mobility

When an order arrives at a desk, this order is translated into a list of required items, and requests are dispatched towards the corresponding suppliers. A supplier checks whether the required item quantity is available, then returns the items to the desk. The desk finally collects the items and delivers them. For collecting the distinct elements, two strategies are considered: message passing and mobile agents.

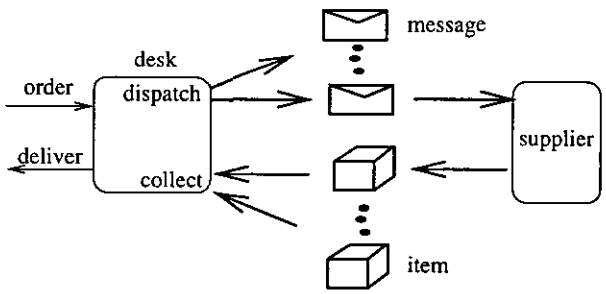


Figure 1: Message Passing

message passing: a multicast is performed, that is messages are sent to the respective component suppliers, which by receiving the request will return the expected number of items.

Communication Policy As illustrated in the previous example, each correspondent of the instance disposes of a specific communication queue. In our case, the name of the queue (or the interaction point) is a binary predicate

`req(_class,_ref)` where variables `_class` and `_ref` indicate the identity of the sender. This knowledge will be used for the reply message.

As each desk represents a specific product, order message is reduced to the number of required items, and the requester identity is encoded in the name of the queue,

delegate creation as many agents as the number of different sorts of items are created and each agent will collect the requested components.

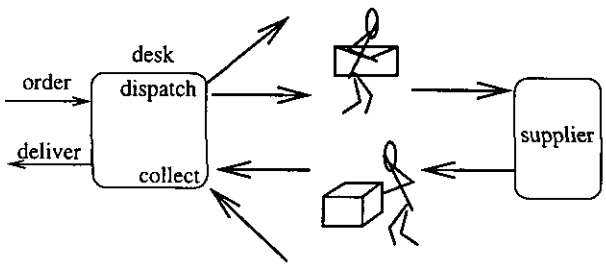


Figure 2: Delegate Creation.

3.2 Communication Pattern

The principle is to consider that the desk decomposes a consumer request into a list of ingredients, each one being delivered by a distinct supplier.

For instance, object *couvert* is composed of pair $knife, fork$, two secondary requests have to be issued towards the respective suppliers, as depicted by Figure 3.

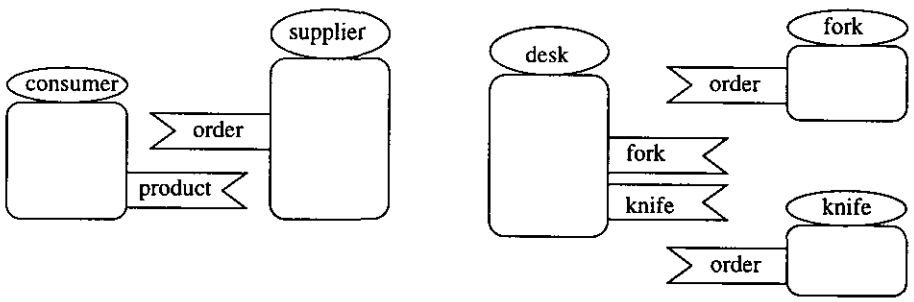


Figure 3: Hierarchical Collecting.

An important new functionality is introduced: the on-line computation of messages to be issued or received.

When a customer issues a request at a front desk, the desk is in charge of collecting the needed items. From the component list, two lists are simultaneously computed: order list and expected list, denoted below *_send*, *_wait*, respectively.

This computation is the purpose of predicate
compute_mes(*_gloss*, *_request*, *_send*, *_wait*). Two input parameters are
_gloss, which supplies the component list of an object, and
_request, which supplies the request.

Predicate *compute_mess* determines two terms *_send*, *_wait*, which represent messages to be sent and messages to be received, respectively.

Let *gloss*([(*couvert*, [*knife*, *fork*])]) be the predicate which associates with *request* = *couvert*, component list [*knife*, *fork*], then the following pattern are derived by *compute_mess*:
(*desk*, *fork*, *req*(*desk*, *couvert*), [1])
(*desk*, *knife*, *req*(*desk*, *couvert*), [1])

whose interpretation is two requests are issued on the behalf of desk *couvert* towards desks *knife* and *fork*. Of course one knife and one fork are expected as answer.

The behavior of agent class *desk* then consists of transitions *order*, *collect*.

TRAN	*order*	(instantiation)
FROM	*idle*,	idle
	gloss(*_gloss*)	gloss([(couvert.[knife,fork])]))
WHEN	(*req*(*_class*, *_ref*),	(*req*(*consumer*, c),
	_request)	1)
COND	*compute_mes*	
SEND	*_send*	(desk, fork,req(desk,couvert) [1])
		(desk, knife,req(desk,couvert) [1])
TO	*collect*((*_class*, *_ref*),	collect((consumer, c),
	_request, *_wait*),	[1], (fork, knife),
	gloss(*_gloss*)	gloss([(couvert.[knife,fork])]))
END		

With respect to transition *collect*, the purpose of predicate *whengpe*(*_wait*, *_ack*) is to derive the expected messages from list *_wait* of expected ingredients. By receiving the ingredients, the collecting desk returns the (re)composed object.

TRAN	*collect*	instantiated by
FROM	*collect*((*_class*, *_ref*),	collect((consumer, c),
	_request, *_wait*)	[1], (fork, knife),
COND	*whengpe*(*_wait*, *_ack*)	
WHEN	*_ack*	(fork, 1), (knife, 1)
SEND	(*_class*, *_ref*, @*ref*, *_request*)	(consumer, c, couvert, 1)
TO	*idle*	idle
END		

3.3 Mobility

Instead to send an order and to receive the requested item, another approach is to create a new agent, let *clerk* be the class name, whose purpose is to collect the components.

Two policies are possible : either as many agents are created as the number of distinct ingredients, or a single clerk will collect the ingredients, by visiting

each supplier. The first policy only is described: the desk behavior is modified, a clerk behavior is introduced.

3.3.1 Desk Behavior

The desk transitions are modified, for creating new agents. The new transitions *order, collect* are defined by the following clauses.

Trans	receive_order
From	idle
Rdv	(Consumer,_consumer,order,_list)
Cond	Compute_team(_list,_team)
Create	_team
To	wait(_consumer,_list)
End	
Trans	collect
From	wait(_consumer,_list)
Cond	Compute_Synchronization(_list,_team)
Rdv	[(Consumer,_consumer,deliver,_list)|_team]
To	idle
End	

Predicate $Compute\_team(\_list, \_team)$ receives as input the list of requests and produces as output the pattern of clerk agents to be created. In a similar way, predicate $Compute\_Synchronization(\_list, \_team)$ determines the pattern of a multiple rendez-vous with several clerk agents.

Let $[(knife, 1), (fork, 1)]$ be the list of requests, the produced pattern is then:
$\_team = (Clerk, fork, [from(couvert), req(1)]),$
$(Clerk, knife, [from(couvert), req(1)])$

This pattern corresponds to the tuple $class, identifier, initialmarking$, that is two clerk agents will be created, with respective names $knife, fork$, and their initial state is composed of the pair $from(\_desk), req(\_num)$ which specifies the requesting desk, and the needed quantity.

3.3.2 Clerk Behavior

A clerk instance is created by a desk, and corresponds to an object to be collected. After being created by a desk, a clerk agent meets the right supplier, picks the needed quantity, then returns to the desk.

A clerk behavior is defined by transitions $go, pick, return, deliver$.

TRANS	go	TRANS	back
FROM	from(_desk)	FROM	atsupplier
TO	atsupplier, from(_desk)	TO	atdesk
END		END	

```
TRANS   pick
FROM    atsupplier,req(_r)
RDV     (Supplier,@ref,pick,_r)
TO      atsupplier,ack(_r)
END
```

```
TRANS   deliver
FROM    atdesk,from(_desk),ack(_r)
RDV     (Desk,_desk,ack,_r)
TO
EXIT    true
END
```

The transition interpretation should be self explanatory: a clerk moves from the desk to a supplier, once at the supplier location it picks the needed quantity, returns to the desk, and there delivers the item.

It is worth to notice that at the delivery the clerk agent disappears.

4 Hierarchical Structure

This section deals with nested structures: component clusters are defined for encapsulating parameterized agent structures. The system is no more composed of direct instances of classes, the ordered object is itself an object hierarchy. The interpretation of a received request is locally performed, the customer is not aware of the process which may internally be performed by a desk.

The former glossary associates with an object, a list of items. In the extended glossary, an item may be tree structured. The implementation policy may remain unknown for an external user : either a single dispatch manager knows the whole glossary and issues queries for terminal objects, or a desk solicits an other desk, without knowing whether the expected object is terminal or not (i.e. it is to be decomposed). In the sequel, this latter policy is described.

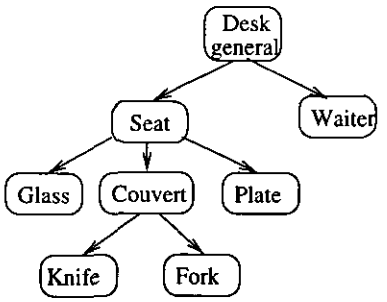


Figure 4: A query tree.

For instance, a dining room is composed of distinct agents, whose the casting depends upon organizational choices. The queries may consist of the following reactive objects: To serve a customer, the *desk* needs a *waiter* and a *seat*. A *seat* consists of *glass* and *plate* in addition to *couvert*. A *couvert* is composed of *fork* and *knife*. A key point is to consider any agent as a desk instance, whose initial parameters declare the role within the structure.

4.1 Description of the proposed solution

The purpose of this new solution is to be able to describe in a generic way a system parameterized by a hierarchical structure, as depicted in Fig 4.

Roughly speaking, the hierarchical description of the services will be interpreted at the creation of the system. To each node of the tree will be associated a specific agent desk parameterized by the set of its sons in the hierarchy. As result, the central view of the hierarchy is distributed among the different agents representing it.

The following table represents the initial state of the system encoding the abstract hierarchy depicted in Fig 4.

Note the presence of a specific desk whose reference is **general** which constitutes the root. This node is known by any consumer.

consumer_c([request([(table,1)]),service(desk)])
desk_general([idle,glossary(nil)])
desk_couvert([idle,glossary([(couvert,[knife,fork])])])
desk_seat([idle,glossary([(seat,[glass,plate,couvert])])])
desk_fork([idle,stock(2)])
desk_glass([idle,stock(2)])
desk_knife([idle,stock(2)])
desk_plate([idle,stock(2)])
desk_waiter([idle,stock(2)])

Finally, depending on the position of the node on the tree, two distinct behaviors have to be provided: a desk or a supplier behavior. In order to simplify the description of the example, these two distinct roles are merged within class *desk*.

4.1.1 Supplier Component

Predicate **stock** is an unary predicate on the number of available references.

transition **deliver** describes the actions associated with the reception of an order issued by agent instance (**_class,_ref**).

The value of the available stock is determined by instantiation of predicate stock(_s).

If the required quantity **num** is available (that is least or equal to the available stock: **_num =<** _s), then a delivery message is sent to the demander (the queue associated with this message is given by the reference of the considered instance (cf meta-variable **@ref**)), and the remaining stock is updated.

4.1.2 Desk Component

The data processing, namely with respect to the communication pattern computation, plays a crucial role in the proposed behavior.

Attributes: the state of the instance is defined by the following three predicates.

`idle` : the initial state of the instance

`glossary` : an unary predicate recording the instance knowledge of the hierarchy of services. In the case of the hierarchy depicted in Fig. , glossary of instance `seat` of agent desk will be of the form `(seat,[Glass,Couvert,Plate])`. Then, when the desk.seat is asked he knows how to fulfill the order. A set of messages is derived from the initial order, this derivation is computed according the informations recorded in the glossary.

`collect(_class,_ref),_list,_wait)` : is a ternary predicate indicating

1. `(_class,_ref)` the identity of the requester
2. `(_list)` the list of the required items
3. `_wait` the set of instance references associated with the transaction.

Data processing:

`compute_mes` When the desk.seat is asked, the received order is translated in a set of messages according the hierarchy recorded in the glossary. Procedure `compute_mes(_glossary,_list,_send,_wait)` is is in charge of this work. Variables _glossary and _list may be considered as input datas while _send and _wait are the outputs of the procedure.
- _glossary represents the value of the glossary of the considered instance,
- _list the list of ordered items
- _send is a set of emission patterns associated with the list of orders to be satisfied. The initial set (_list) is translated into a new set of messages (_send) according the value of the glossary.
- The same computation is used to determine _wait, the set of instance references whose the response will determine the satisfaction of the transaction.

`whengpe` is a built-in predicate for computing the set of reception patterns which corresponds to an agent group.

Description of the behavior

Transition Order:
Transition `order` describes the reception of a list of orders. The instance is idle, that is no transaction occurs. The received message (_list) emitted by instance (_class,_ref) is treated by means of procedure compute_mes. The set of emission patterns (_send) is emitted and the entity evolves in a state `collect` indicating that a transaction is in progress. This predicate records the identity of the requester (_class,_ref), the initial order (_list) and the set of instance references involved in the transaction (_wait).

```
...
desk_couvert([idle,glossary([(couvert , [knife,fork])])])
desk_fork([idle,stock(2)])
desk_knife([idle,stock(2)])
desk_seat([glossary([(seat,[glass,plate,couvert])]),
          collect((desk , table),[1],[couvert,plate,glass])])
desk_couvert([idle,glossary([(couvert , [knife,fork])])])
input(desk_couvert(req(desk,seat),[[1]]))
...
```

Table 1: State enabling transition desk_couvert_order

```
...
desk_seat([ glossary([(seat , [glass,plate,couvert])]),
          collect((desk , table),[1],[couvert,plate,glass])])
desk_couvert([ glossary([(couvert , [knife,fork])]),
          collect((desk , seat),[1],[fork,knife])])
desk_fork([ idle,stock(2)])
desk_knife([ idle,stock(2)])
input(desk_fork(req(desk,couvert),[[1]]))
input(desk_knife(req(desk,couvert),[[1]]))
...
```

Table 2: State after firing transition desk_couvert_order

Example Let us consider, instance `couvert` of the hierarchy depicted in Fig 4. Table 1 describes a fragment of a system state enabling transition `order`

The context of instance `couvert` is determined by the two following clauses:
- `desk_couvert([idle,glossary([(couvert , [knife,fork])])])` indicating the state
- `input(desk_couvert(req(desk,table),[[1]]))` indicating the presence of message 1 in queue req(desk,seat)

The computation of compute_mes leads to the following instantiations:
- _send :
`[(desk,fork,req(desk,couvert),[1]),`
`(desk,knife,req(desk,couvert),[1])]`
- _wait : `[ fork, knike ]`

After transition firing, the next system state is described by table 2.

Transition `collect` describes the end of the transaction. Builtin predicate `whengpe` computes the set of reception patterns associated with the set of instance references whose the response is expected. When the receptions are possible, an acknowledgement message is emitted to the requester (_class,_ref,@ref,_list). The communication queue associated with this communication is the reference of the emitter `@ref`.

```
...
desk_fork([idle,stock(1)])
desk_knife([idle,stock(1)])
desk_seat([glossary([(seat,[glass,plate,couvert])]),
                    collect((desk , table),[1],[couvert,plate,glass])])
desk_couvert([glossary([(couvert , [knife,fork])]),
                    collect((desk , seat),[1],[fork,knife])])
input(desk_couvert(fork,[1]))
input(desk_couvert(knife,[1]))
...
```

Table 3: transition desk_couvert_collect is ready

```
...
desk_couvert([idle,glossary([(couvert , [knife,fork])])])
desk_fork([idle,stock(1)])
desk_knife([idle,stock(1)])
desk_seat([glossary([(seat,[glass,plate,couvert])]),
                    collect((desk , table),[1],[couvert,plate,glass])])
input(desk_seat(couvert,[[1]]))
...
```

Table 4: After firing of transition desk_couvert_collect

Example Let us consider, instance `couvert` of the hierarchy depicted in Fig 4. Table 3 describes a fragment of a system state enabling transition `collect`.

Clause `collect((desk , seat),[1],[fork,knife])]` indicates that desk_couvert is waiting the responses of instances fork and knife. The set of reception patterns is computed by built-in predicate `whengpe`. Clauses `input(desk_couvert(fork,[1]))` and `input(desk_couvert(knife,[1]))` indicate respectively that the expected responses are arrived. A response message is sent to the initiator of the transaction desk_seat. Table 4 depicts the state system after transition firing.

5 Verification

In order to perform formal verification, the configuration space may be generated (even in the case of dynamic description). The configuration space may then be analysed either by algebraic approaches, i.e. observational equivalence [Mil89], or by temporal logic model checking [ES89]. A specification, that is the formal interpretation of an informal description, should produce two kind of outputs: an operational description and expected properties. The purpose of the verification is then to ckeck whether the expected properties are satisfied by the operational description, (see Figure 5).

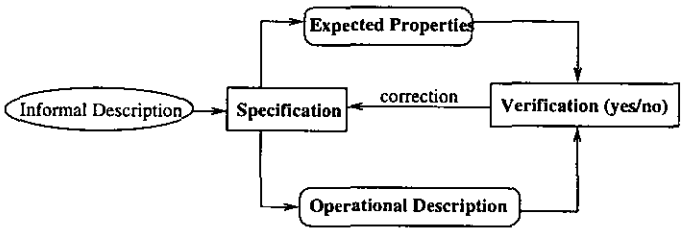


Figure 5: Verification.

Temporal Logic Temporal logic formula are statements on the reachability of global system states. Two basic modal operators are $INEV$itably or POTentially. Here are two examples of temporal assertions. Is it always true that any state for which condition *pending* is true is inevitably followed by a state for which *delivered* is true. $ALL(pending \Rightarrow INEV delivered)$
This statement may be false for a model for which the requests exceeds the resources, because a provider may refuse to serve an order.

Bisimulation This approach allows to derive a reduced view by considering as observable only a subset of the events. The observational equivalent automaton is an automaton where observationally equivalent states are merged into a single class, The behavior is then easier to analyze by considering this specific subset.

Application The system associated with the hierarchy depicted is analysed. This system corresponds to the causal diagram depicted by Figure 5 . More precisely, we investigate the processing of a table request. As we consider a single consumer request, the system has to reach a final state in which the request has been fulfilled, that is all the components of the table have been collected.

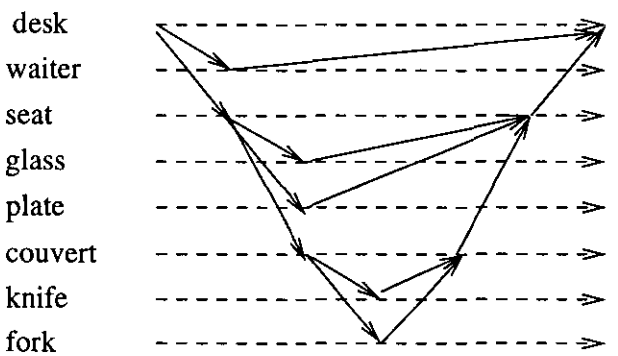


Figure 6: Causal diagram of the query tree.

The complete state system consists in 58 states and 132 edges. This graph

admits a single deadlock corresponding to the expected final state.

Temporal Logic allows to verify that this final state is still inevitable.
$ALL(Table.ordered \Rightarrow INEVtable.delivered)$

Bissimulation techniques allows us to visualize some details of the request processing. Different points of view may be considered by selecting specific set of observed events. Two kinds of observation are investigated:

In the first case, the bottom of the hierachy is considered: interactions between knife, fork and couvert are observed. The way for desk.couvert to collect knife and fork is observed. The obtained quotient automaton is depicted by Fig 7. A sample analysis allows us to verify that after a couvert.order is inevitably followed by a couvert.collect.

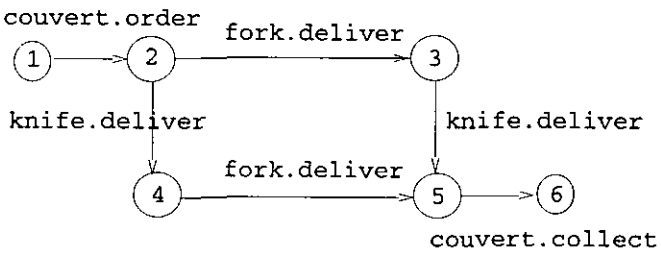


Figure 7: Observational service of interactions between knife, fork and couvert

The second observation focusses on the main steps to follow in order to collect a table. To keep the size of the quotient automaton manageable, the observation is reduced to the first level of the hierachy. The details relative to the processing of a seat or a couvert are not observed.

The obtained quotien automaton is depicted by Fig 8.

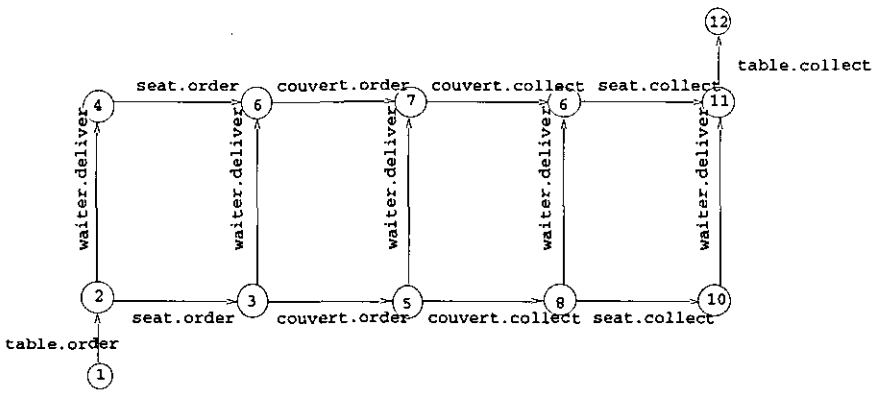


Figure 8: Main steps needed for .....

A sample analysis allows us to verify that after a table.order is inevitably followed by a table.collect.

6 Conclusion

Rapid prototyping and modularity are standard requirements for specification environments. The dynamic aspect seems particularly relevant for workflow analysis. The ability to describe dynamic systems, i.e. agent creation and suppression, and mobile processes, i.e. addressing by name and logic unification, offers a great flexibility to the designer.

For debugging purpose, a specification may be step by step interpreted. The simulator and the associated graphical user interface provide standard facilities : display of the global state and of the communication queues, list of the enable transitions, ...

The proposed approach appears useful during the first design phases [Hol96], in order to rather quickly elaborate a behavioral model, and to determine preliminary requirements. At this level, a crucial aspect concerns the identification of a consistent set of actions, or methods, and the decomposition of a complex system into easier subproblems.

Within the framework of cooperative systems, an agent oriented programming style is particularly adapted [VA96].

A current study deals with the following question: how to take explicitely into account regular architectures during a verification process.

References

[Agh86] G. Agha. *Actors: a model of concurrent computation in distributed systems.* MIT Press, 1986.

[CCI92] CCITT. *SDL.* Z100,Z104 Recommendations, 1992.

[ES89] E.A. Emerson and J. Srinivasan. *Branching Time Temporal Logic.* LNCS, vol.354, 1989.

[Gen91] H. J. Genrich. Predicate/transition nets. In *High-Level Petri Nets: Theory and Application.* Springer Verlag, 1991.

[Hol96] G. Holtzmann. Early fault detection tools. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–13. Springer Verlag, LNCS 1055, 1996.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[VA96] F. Vernadat and P. Azema. Prototyping of communicating agent system. In *ECOOP'96, Workshop on Proof Theory of Concurrent Object-Oriented Programming, European Conference on Object-Oriented Programming*, Linz, Austria, July 1996.

[VAL95] F. Vernadat, P. Azéma, and A. Lanusse. Actor Validation by means of Petri Nets. In *Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, june 1995.

Object – Oriented and Net-Based Modelling of Business Processes

Angela Mölders, Martin Wolf, Wolfgang Fengler
Technical University of Ilmenau, Germany
Rainer Burkhardt, OWiS Software GmbH

Abstract

In the field of modelling enterprises and business processes the classical structured modelling techniques are dominating; there are only a few object-oriented approaches. A method for creating a business process model will be proposed and the Object – Process – Net (OPN) will be presented, which can be used for describing the dynamic aspects of systems and which can be added to the family of diagrams combined within the UML (Unified Modelling Language). The OPN can be considered as an object-oriented Petri Net and can be transformed into a high-level Petri Net conform to the standard.

Keywords: Relationships between net theory and other approaches, Applications of nets to workflows, System design and verification using nets, Object Oriented Petri Nets, UML (Unified Modelling Language)

1 Introduction

Modelling of business processes attains increasing importance since it is an indispensable basis for successful business process reengineering as well as for the inauguration and the use of workflow management systems.

The business perspective focuses increasingly on an integrated, process oriented view on the enterprise. Hence, it appears to be advantageous to apply theoretical methods of computer science on business management issues. Over the last years an increasingly fruitful co-operation between business management and information theory has developed although there was not much common ground between both sciences in the past. Many issues of today's management reality result in highly complex demands on the development of information systems, particularly in terms of user friendliness, complexity, safety and adaptivity. The information theory point of view, a rather formal one, supports the process oriented examination of business flows.

Although object orientation has become a standard in the field of computer science there are only a few approaches for object oriented modelling of business processes so far while this domain is still dominated by classical structured methods [FeSi96], [Pres97], [PMK97]. Even the standardisation efforts of the WfMC (Workflow Management Coalition) have not been included object orientation yet.

Another aspect, which is disregarded from our perspective is the verification of models of business processes. Due to the complexity of models which arises from the underlying domain of applications it is inevitable to choose a multi-stage, incremental modelling approach in order to make verification possible in each modelling stage. Here, verification should not only be understood as evaluation by simulation but as a formal, analytic method for model verification.

Our way of business process modelling assumes that the UML ([UML1], [UML2]) will establish as the standard method for the object oriented paradigm. The advantage of this approach is the step-by-step evolution of a rather informal model to a formal one. The OPN, developed by the authors, is to be understood as an additional diagram of the UML for the description of dynamic aspects of a system.
The following chapter gives a short introduction to some UML-diagrams and after then the Object Process Net will be explained in chapter 3. In chapter 4 our method for creating a model is presented followed by some notes to tool support and model-checking methods.

2 The UML as the standard technique for object oriented modelling

The UML combines a set of diagrams for object oriented modelling of systems on different levels and from various views onto the systems. There are diagrams for the static, dynamic and architectural aspects . For special fields of system modelling only a subset of this diagrams will be used because some of them are redundant. A method for model-building is lacking within the UML.
Within our modelling method described in chapter 4 we use Use Case Diagrams, Class Structure Diagrams, Activity Diagrams and the Object Process Net. All this diagrams will be shortly explained and illustrated by a common example. Due to the limited scope of this paper only a clipping of the whole model can be presented. Within the framework of co-operation between a furniture and toy producing factory and the Technical University of Ilmenau various methods for modelling the business processes of this enterprise have been applied and evaluated. We modelled the process of handling orders for furniture within the enterprise.
The **Use Case** approach was developed by Jacobson [Jaco+92] [Jaco+95]. A use case diagram represents the external functionality of a system or a class as visible to an actor external to the system. It only shows *which* services a system gives to the environment but not *how* this service will be done by the system. Different scenarios of a use case are usually described textual.
The actors are humans as well as computers or other systems. Actors communicate with the use cases. Between use cases two types of relationships are possible: an *extends* and a *uses* relationship.
A **Class Structure Diagram** (CSD) gives a graphic view of the static structural model. Within a CSD the classes with their attributes and methods are described as well as role-associations and inheritance relationships between several classes.
The role-associations will be considered as attributes. With respect to Figure 1 for instance an object of the class *order* has the attribute *user* from type *people*.

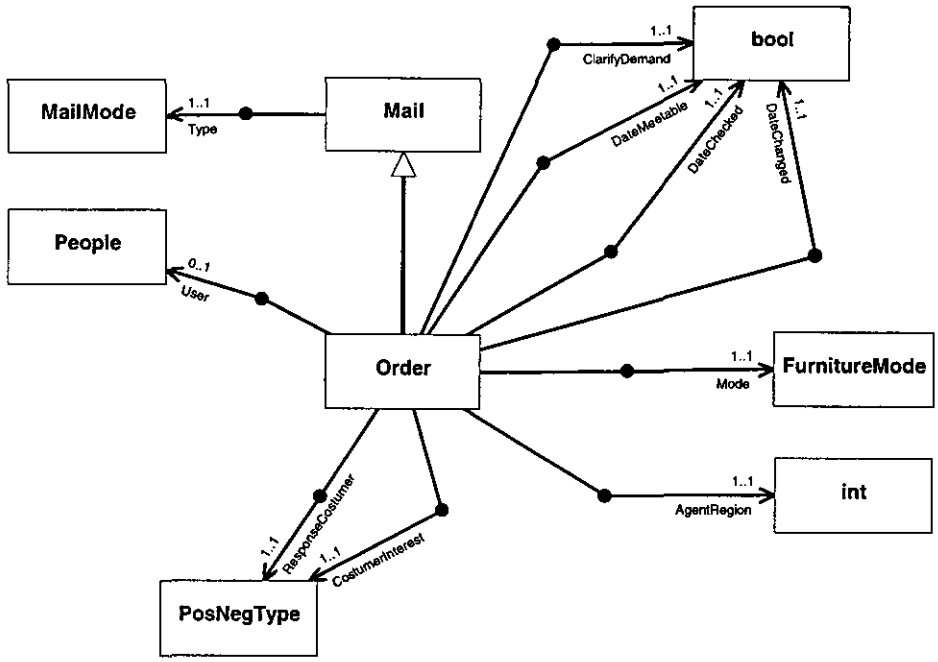


Figure 1: CSD for the class *order*

An **Activity Diagram (AD)** is a special case of a state diagram combined with some Petri Net ideas. Within Activity Diagrams the internal dynamic behaviour of use cases can be described. An AD captures the sequential and/or parallel processing order of the activities which produce the performance of a system for the environment. The model components of an Activity Diagram described within the UML allow the complete representation of all elementary structures for modelling business processes denoted in the glossary of the Workflow Management Coalition [WfMC97]. Activity Diagrams can be designed in a rather abstract manner as well as enriched with formal details. Therefore, they are a good entrance for the stepwise formalisation of the use cases. An activity is a state of the system with an inner action which is connected to other activities by transitions. An incoming transition initiate an activity. If there are more than one outgoing transition for an activity than they have to be distinguished by logical expressions. It is possible to split or synchronise transitions. The division of an Activity Diagram into socalled *swim lanes* gives the assignment between activities and the responsible objects.

The following Figure 2 shows a part of the order handling process and is self-explanatory. The first steps like incoming and sorting of orders are omitted. Here, only the time-dependent scheduling of orders will be illustrated.

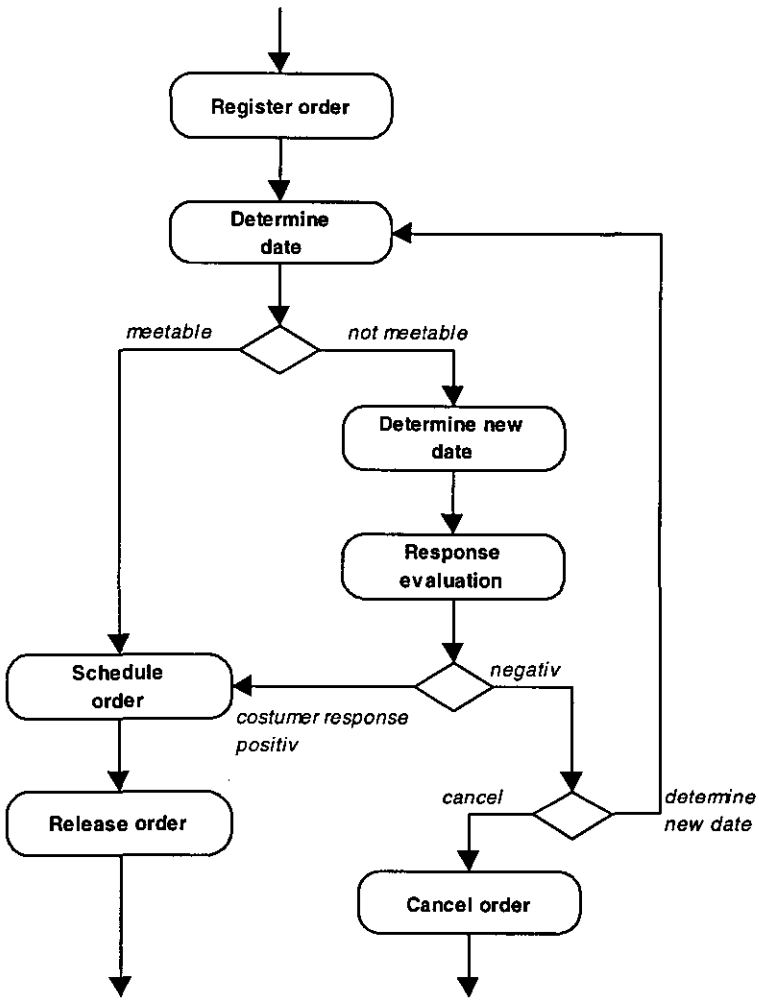


Figure 2 : Part of the Activity Diagram for order handling

In Figure 2 decisions within the flow of processing are represented by diamond shapes. If an activity has more than one incoming transitions (*Schedule order* in Figure 2) it represents an OR-Join conform to the WfMC-definitions for basic structures of business processes [WfMC96].
Another diagram for describing the dynamic aspects of systems is the Object Process Net developed by the authors. The individual business activities are connected via pre- and post-conditions. Therefore the procedural order of them is given implicitly. The simulation of an OPN gives various scenarios for the related use case. OPN is well suited for detailed modelling of parts from the system.
Both Activity Diagram and OPN are dealing with the classes and their methods defined in the Class Structure Diagram.

3 The Object Process Net (OPN)

3.1 Basic Ideas

The OPN carries on the idea of the Object Process Model (OPM) [Burk94], which was especially developed for software engineering starting by analysing the dynamic aspects of systems [Schm97]. The goal is the creation of a system model, which will be directly transformed into source code of the destination language [OTW97]. When using the OPN for modelling business processes this feature is not the most important one but the ability of describing the dynamic behaviour of system.

According to the object oriented paradigm the abstract objects represent the classes and the processes represent the methods defined for the classes. The graphical notation of an OPN is a bipartite, directed graph the nodes of which are the objects and processes. A complete model describing a real-world system consists of a *system of OPN diagrams*. Every OPN belonging to such a system shows a delimited part; therefore the models are rather clear.

The state of the modelled system is described by the values of attributes belonging to the objects. Therefore, a change of these values describes a variation of the system's state. Attribute values can be changed by activating and running processes.

On the one hand, the OPN is considered as an additional diagram within the family of diagrams forming the UML [UML1], [UML2]. Therefore it is possible to enhance an OPN by OCL-statements (Object Constraint Language) [OCL]. This makes the models more understandable and clear.

On the other hand, the OPN can be understood as an object oriented Petri Net. It is possible to transform an OPN into high level Petri Nets conform to the standard [Conc97]. This results in the ability to find analysis techniques directly for the OPN.

3.2 Objects

Objects are described exclusively as abstract objects. Hence, the properties modelled within an OPN hold for all instantiated objects of this class and of derived classes. In this way, the inheritance relationships are captured by an OPN.

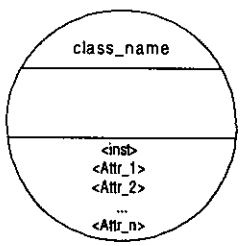


Figure 3: Abstract Object

The graphical representation of an object is a circle, which is divided into three parts. In the upper part the name of the abstract object has to be denoted mandatory. The middle part can be used for showing role-associations between various objects. This is a useful feature for modelling.

In the lower part of a pattern for an object a set of attributes can be specified. Every abstract object possesses an instance list *<inst>*, which can be considered as a special attribute and which contains all the instantiated objects of the related class. This list can be – particularly in the initial state - empty. Every instantiated object possesses all the attributes, which are defined for the abstract object to which it belongs. That is, the instance list of abstract objects having derived objects is the union of all instance lists of the derived abstract objects.

It is only necessary to declare such attributes in the lower part of an object pattern which are essential within the context of this OPN, i.e. the same object can be de-

picted with different subsets of attributes within various OPN belonging to a system of OPN's. In the following, a certain attribute is denoted by <*attr*> and its value by *val(attr)*. The value of an attribute can be *undef*, that means it has no defined value. According to the object-oriented paradigm the attributes are instances of four pre-defined colour classes or of a user-defined structured data type. For every colour class there exists a defined set of operations for dealing with the attributes.
Now a description of the pre-defined colour classes ENUM, INT, SET, MULTISET follows.
ENUM – comparable to an enumeration type. The finite set of values has to be defined by enumeration of all elements.

$$< attr > = ENUM\ \{value_1, value_2, .., value_n\},\ n \in \mathrm{N},\ value_i \neq value_j\ \forall\ i \neq j$$

$$\text{with } val(attr_{ENUM}) = \begin{cases} value_i & wert_i \in ENUM \\ undef \end{cases}$$

An ENUM-attribut can be considered as a finite sequence with pair-wise different elements.
INT – comparable to integers in programming languages.

$$val(attr_{INT}) = \begin{cases} n \in \mathrm{N} \\ undef \end{cases}$$

With respect to the finite domain by implementation an INT-attribute always has a finite value.
SET – comparable to a container class, which can only contain one copy of each element, i.e. it is comparable to a mathematical set. Declaring a SET-attribute requires the definition of a basic set B which define the domain of the attribute.

$$val\left(attr_{SET(B)}\right) = \begin{cases} VAL & with \quad VAL \subseteq B \\ undef \end{cases}$$

The value of a SET-attribute is a subset of the basic set B. The set B can be a previous defined ENUM.Type.
MULTISET – comparable to a container class, which can contain more than one copy of each element, e.g. comparable to a mathematical bag. Defining a MULTISET-attribute includes the declaration of the underlying basic set B.

$$MULTISET: B \rightarrow \mathrm{N}$$

It is possible to use a defined ENUM as basic set. The multiplicity of an element e of the underlying set B within the bag is denoted by *m(e)*. The value of such an attribute is a bag itself and can be denoted as a weighted sum:

$$val(attr_{MULTISET(ENUM)}) = \sum_{attr \in ENUM} m(attr)\, val(attr)$$

Using these four elementary colour classes it is possible to declare structured types, called CLASS.

$$CLASS = (e_1, e_2, \ldots, e_k) \quad with\ e_i \in (ENUM, INT, SET, MULTISET)$$

A CLASS-attribute describes the merger of some elements belonging to various elementary colour classes and can be called by the name of the attribute. However, access to the individual elements is possible too. The value of such a CLASS-attribute is the combination of the individual element values.

$$val(attr_{CLASS}) = (val(e_1), val(e_2), \ldots, val(e_k)) \quad and \quad val(attr_{CLASS}.e_i) = val(e_i)$$

Another special kind of attributes are role-attributes. Roles between different classes can be declared within the class structure diagram belonging to the system's model and can be used within the OPN.

3.3 Processes

Processes represent the instances of methods of individual classes and describe the dynamic behaviour of objects. When they are activated they can change the values of attributes of the corresponding objects and thus, depict state modifications within the system. A process always belongs to a class and can only be activated and run with objects of this class. For objects of other classes processes may only cause changes to the attributes by sending messages. Hereby, it is ensured that an object can only call a different object or communicate with it out of a method.

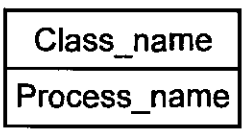


Figure 4: Process

The graphical representation of a process in an OPN is a divided rectangle. In the upper region the name of the class to which the process belongs has to be inscribed and in the lower part the name of the process itself is denoted. None of these inscriptions can be omitted. A process can be drawn in a system of OPN's only once (in opposition to objects, which can appear arbitrarily often).

A process can be activated only under certain circumstances: preconditions determine the individual system state, which is necessary for starting the process. Therefore, preconditions check certain values of the attributes of objects. Postconditions determine the state of the system after termination of the process. They change the attribute's values, i.e. they describe the changes of attribute values by running a process. Before starting a process it has to be checked whether these changes are possible.

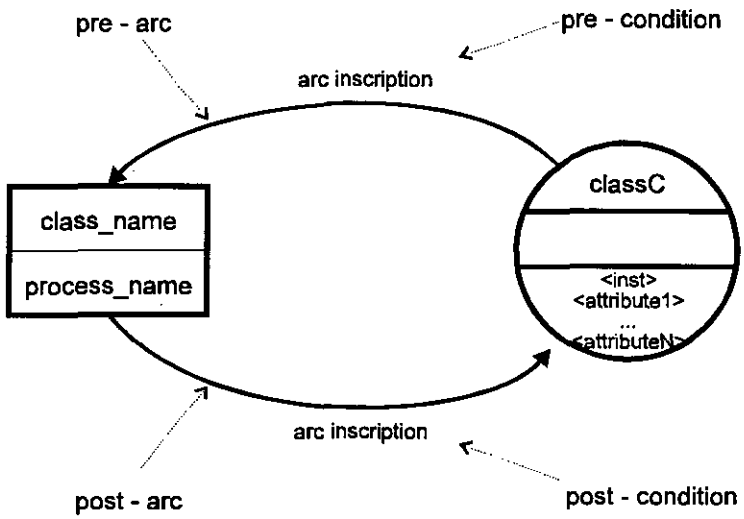


Figure 5 : Graph of an OPN

Arcs connect objects and processes and vice versa, but never net elements of the same type. Pre-and postconditions of processes are represented as arc inscriptions. Arc inscriptions are logical expressions built with attributes of related objects using suited operators. A condition is fulfilled if the related term will be evaluated to the value TRUE. Related to their directions arcs denote pre- or postconditions of processes: an arc directed from an object to a process is called pre-arc, an arc from a process to an object is a post-arc.

Processes can subjoin new objects to the list of instances of an object via post-conditions (*create* functionality). It is possible to instantiate new objects from the class to which the process belongs or from other classes. The values of the attributes of this instantiated objects can be undefined or have defaults or certain values. According to the create mechanism there exists a *destroy* functionality, i.e. instantiated objects can be destroyed. Furthermore, objects can be displaced by processes within the static class hierarchy.

Processes can be assigned a priority in order to solve conflicts during simulation. Furthermore a process can have a time attribute which determines the running time of this process.

3.4 Refinements of Processes

Processes can be refined. If a process has a refinement this will be shown by denoting a "+" in the upper right corner of the depiction of a process. The refinement of a process is an OPN as well, which is called subnet.

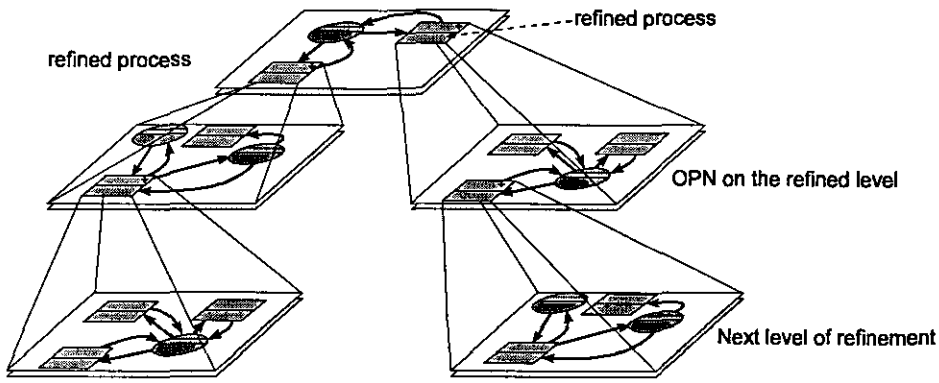


Figure 6: Refinement of processes

Using this mechanism of hierarchy realises inheritance and polymorphism within the system model. The preconditions of a process having a refinement will be inherited to all the processes on the refined level. Subnets can not be connected directly. Only indirect, invisible links between different subnets are possible because processes belonging to different subnets can affect the same object.

3.5 Arc inscriptions

Arc inscriptions are terms built of attributes by using the defined set of operations for each colour class. According to the actual value of the appropriate attributes the term results in a Boolean value. If the value of a term is TRUE, than the related condition is fulfilled.

There are two types of operations/operators which are defined for the several colour classes. Detailed description of each operation is not possible here because of the limited scope of this article.

1. *Value-changing operations* modify the values of an attribute. The resulting value belongs to the same colour class as the attribute.

$$op : C_i \times C_i \rightarrow C_i \quad with\ C_i \in (ENUM, INT, SET, MULTISET)$$

Examples for such operations are increment or decrement of INT-attributes.
These operations are used for building post-conditions.

2. *Testing operations* check the value of an attribute and result in a Boolean value.

$$op : C_i \times C_i \rightarrow BOOLEAN$$

Typical examples of such operations are test for equality or inequality.
Testing operations are used within pre-conditions exclusively.

A pre-arc inscription is a term built from one or more testing expressions related to attributes, which can be combined by logical AND (&&) and/or logical OR (||). Post-arc inscription terms can only contain value-changing expressions connected by logical AND (&&). The use of the OR-operator is prohibited within post-arc inscriptions.

3.6 Dynamics

Simulating an OPN is comparable to playing the token game in Petri Nets: if all the pre- and post-conditions of a process are fulfilled it will be activated and run[1]. All the objects contained in the instance list of an object can be considered like the structured tokens in Petri Nets. Therefore, a process can be activated several times if there is more than one instantiated object fulfilling the conditions.
A process can be activated if the pre-condition as well as the post-condition is fulfilled, i.e. if the related terms result in the Boolean value *true*. This checking has to be done for every instantiated object contained in the instance list of the abstract object.
Concerning pre-condition it is clear how to determine the Boolean value of the corresponding term because the testing operations result in Boolean values.
Determining the Boolean value of a post-arc inscription is a two-stage procedure:
First one has to check whether the several operations contained within the term are executable. For instance the decrementation of an INT-Attribute with an actual value of 0 is not executable. This results in the Boolean value *false* for the related term. A value-changing post-condition is fulfilled if at the moment of starting the process the operation(s) result in a value different from *undef*.
Next it has to be checked whether the attributes, which have to be changed by the term, are not *locked*. This lock mechanism is realised for avoiding problems by multiple access to the same attribute of the same instantiated object. If a process has a post-arc inscription containing an attribute, this attribute is locked during the running of this process. Other processes can not be activated if they want to change the value of the same attribute.
If conflicts occur they can be solved by interpreting the priorities given to the processes.

3.7 Example for an OPN

The following two figures show the OPN-diagram for the same facts modelled with an AD as shown in Figure 2 – the determination of a deadline for the order and the release for further activities.
Figure 7 illustrates that the processes *Order::Archive* and *Order::DetermineDate* can be activated in parallel (for a single order). However, each of the two processes is activated only once for each of the instantiated objects. Precondition for both processes is the accomplished registration *(State==registered)* and that the archiving/deadline determination has not yet been performed *(State!=archived / State!=dated)*. As postcondition the respective attribute values are added to the SET *state (State+=archived / State+=dated)*. The performed deadline determination is a pre-condition for the release of the order for further activities which is represented in the model by *Order::Release*.
Figure 8 shows the refinement of the process *Order::DetermineDate* from Figure 7.

---

[1] For activating processes we introduce a discrete time step and running a process (without a refinement) takes one step. This is combined with a maximum or stochastic firing rule. Stochastic firing rule means that the process will run under a given probability if it is activated. Because all simple processes (without a refinement) take one step for running time is introduced implicitly within the model.

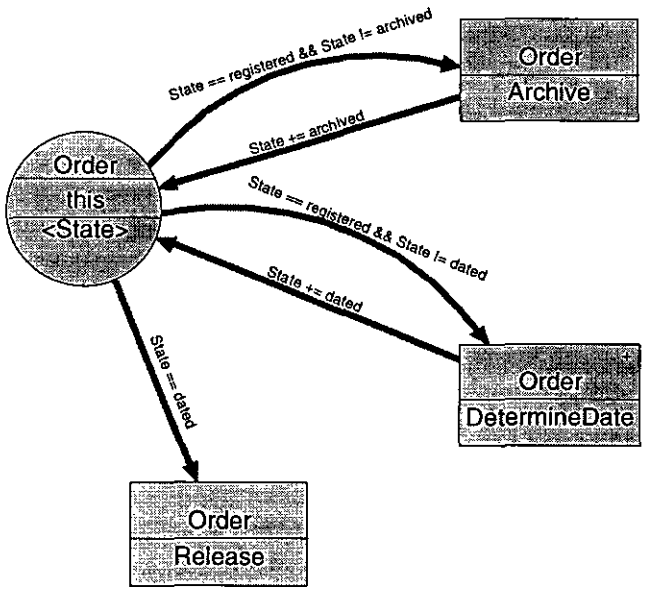


Figure 7: Final order processing

The process *Order::CalculateDate* determines the calendar week the order can be finished in. After completion of the calculation the order is assigned a value *Date* which has not yet been tested. An unchecked deadline is tested by the process *Order::TestDate* in order to verify its achievability. Then, the attribute *DateMeetable* has a defined value (yes/no). In case the deadline can not be met it has to be adjusted by the process *Order:AdaptDate.* In the postcondition *DateChanged=yes* the change of the deadline is registered such that it becomes achievable. If the deadline can be met for sure *(DateMeetable==yes / DateChanged==yes)* the order is scheduled for production by the process *Order::Schedule.* A multiple scheduling of the order is prevented by the supplementary precondition *State!=scheduled* since the value *scheduled* is added to the attribute *state* as a postcondition of *Order::Schedule.*

When a change of date has been performed reconfirmation with the customer becomes necessary *(Order::CustomerEnquiry).* His positive or negative answer is registered in the attribute *ResponseCustomer.* In case the response is negative *(ResponseCustomer==negative)* this reaction has to be evaluated in the process *Order::ResponseEvaluation* which is further refined in Figure 9. In consequence of this process the attribute *Date* is reset because either the order has been cancelled or a new deadline needs to be calculated.

The accomplishment of the deadline determination *(Order::DetermineDateFinish)* is reached when the order has been scheduled, the deadline can be achieved or a change of date has occurred and was confirmed by the customer.

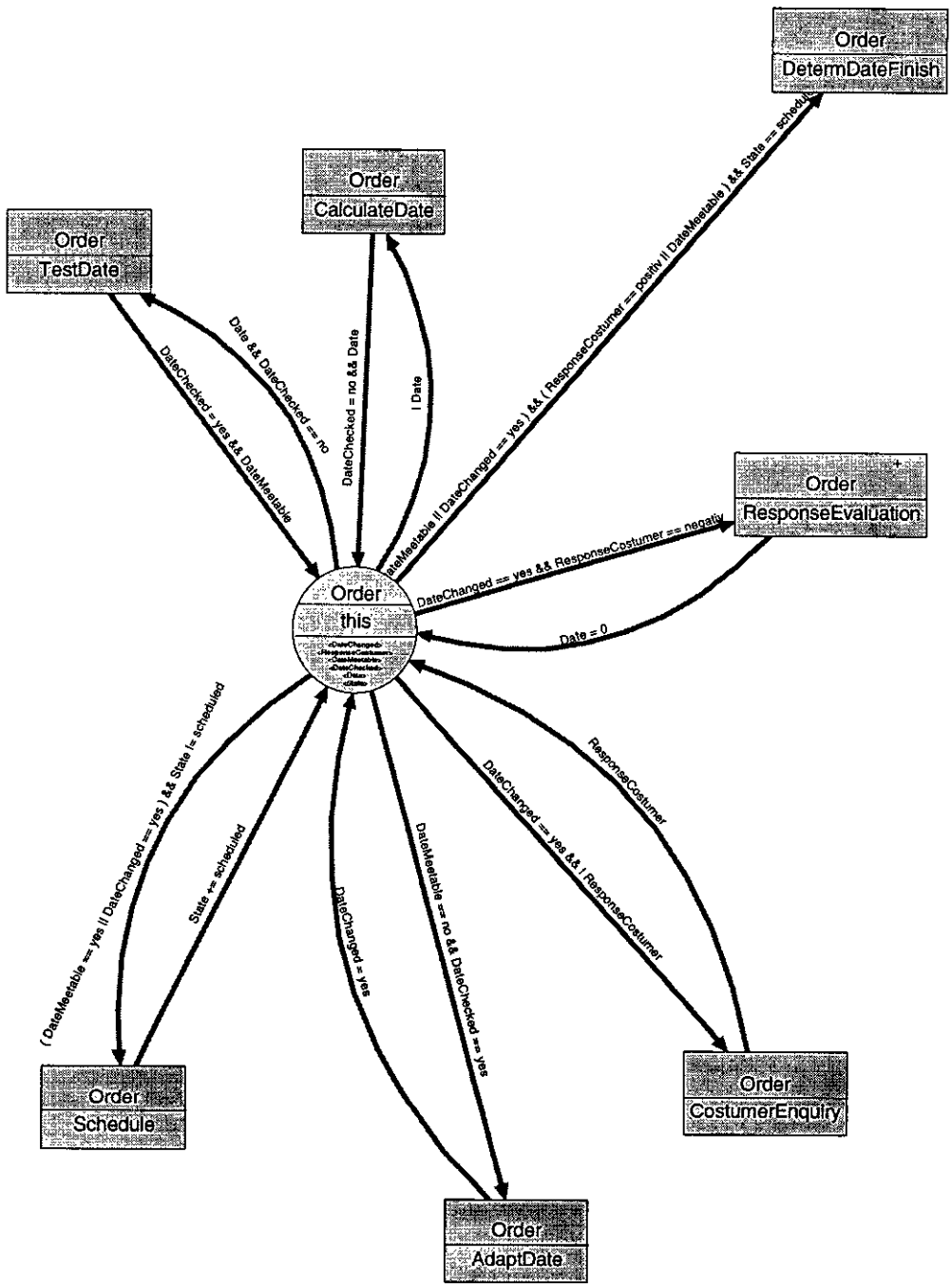


Figure 8: Refinement of the process *Order::DetermineDate*

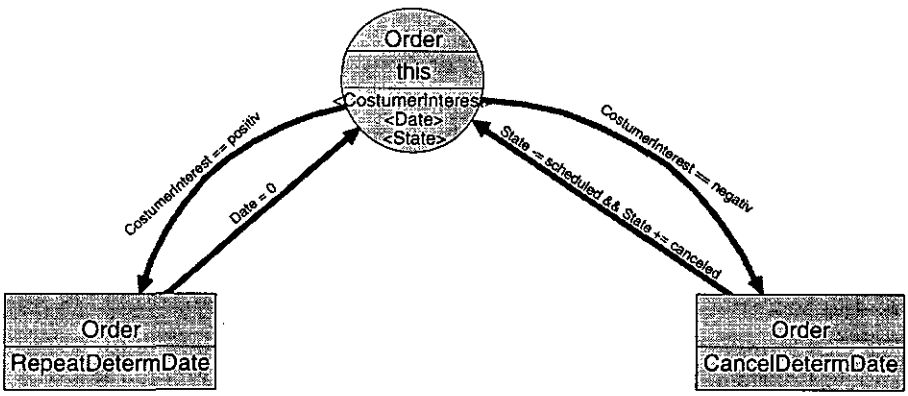


Figure 9: Refinement of process *Order::ResponseEvaluation*

In case the customer has not accepted a changed deadline two alternatives have to be distinguished: one the one hand the chance is that the customer is no longer interested in a further processing of his order, i.e. the order is getting cancelled. On the other hand he still may want his order to be processed if an earlier deadline can be met. Both cases are registered in the refinement of the process *Order::ResponseEvaluation* in Figure 9. When the customer does not want to retain his order *(CustomerInterest==negaitve)* the order has to be cancelled.
Otherwise, only the post-condition of *RepeatDetermDate* is reset to the date of the order such that a recalculation can be done.

4 Procedure of creating a business process model

In cooperation between the TU Ilmenau and OWiS Software Ltd. a framework for the object oriented software development process has been developed, called SEPP/OT (Software Engineering Process for Professional Projects based on the Object Technology) [WBP98]. SEPP/OT is divided into four "Use Cases" which stay active throughout the intire duration of the project and describe the main columns of the actions. These use cases are: Requirements capture, modelling, conversion and introduction. SEPP/OT is not a concrete instance of an activity flow model but rather a framework, which can be completed specific to a project and an enterprise.
The main characteristic of SEPP/OT is the stepwise, incremental enhancing of a model starting from use cases. For business process modelling we adapted SEPP/OT. Figure 10 illustrates our method for creating business process models. The spiral arrow symbolises that the modelling process is iterative and incremental. During the cyclic passing through the several phases the model will be completed and enhanced. There are interactions between all types of diagrams within Figure 10. For instance, the CSD diagram gives the information about the swim-lanes to the AD. Vice versa during creating an AD the model-builder obtains ideas about the static class structure.
Starting point for creating a business process model is a Use Case Diagram. In the meaning of business process modelling a use case diagram describes the perform-

ances of an enterprise for the environment and the interaction between them. Use cases facilitate the dialogue between the user and the developer. Starting from the Use Case Diagram there are some possibilities for enhancing the model as can be seen in Figure 10. The Use Case Model itself can be refined during the next modelling steps as well. But mostly it only gives a first informal approach for modelling the system.
Within the Class Structure Diagram the structural makeups of an enterprise can be specified as the static aspects of the system. Additionally, this diagram can be used for declaring the attributes of the objects, the role associations between objects and inheritance relations between classes. The CSD captures the actors as well as the infirm objects. It is possible to get an organisation chart for an enterprise directly from the CSD. This is a good aid for the dialogue between the staff of the enterprise and the UML-expert, who is creating the model.

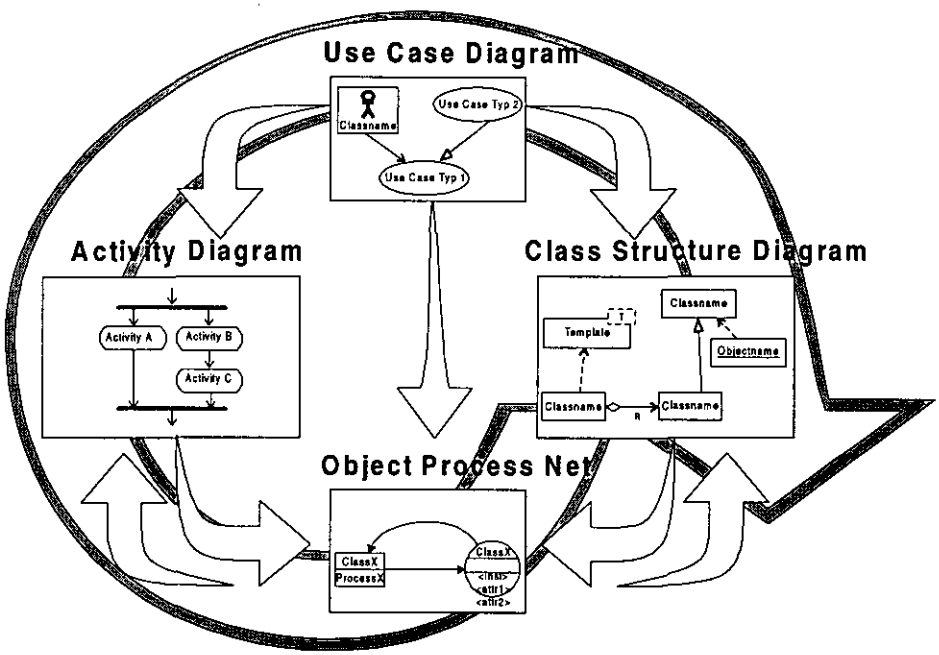


Figure 10: Procedure of creating a business model

For a more detailed description of the dynamic behaviour as the use cases do we use the Activity Diagram. The AD gives a clear view onto the logical flow of procedural order and it is easy to understood for non-informaticans. Outgoing from the AD the instantiation of an *activity tree* is possible, which shows the several pieces of work forming a business activity. This tree only shows the sub-activities which have to be done for fulfilling an activity but not the logical order of this sub-activities.
For getting a more formal model which is ready for simulating an OPN can be created.

5 Tool support and model checking

UML-based modelling requires tool support for ensuring consistency between the several diagrams building the model. We use the *OTW®2* (Object Technology Workbench) for modelling, development and model checking. The OTW is a modelling tool conform to the UML, which supports the development beginning from the requirements analysis to the source-code generation.
This tool for UML-based modelling provides several diagrams of the UML and the OPN. Furthermore, the *OTW®2* includes some integrated tools like Source Code Generators (C++, Java), Reengineering tools, Documentation Generators, Conversion Tools and a Petri Net tool. This Petri Net tool provides an editor and a simulator for Coloured Petri Nets and some features for statistical evaluation of simulation. Additionally an interface to the net analysing tool INA [INA98] is available.

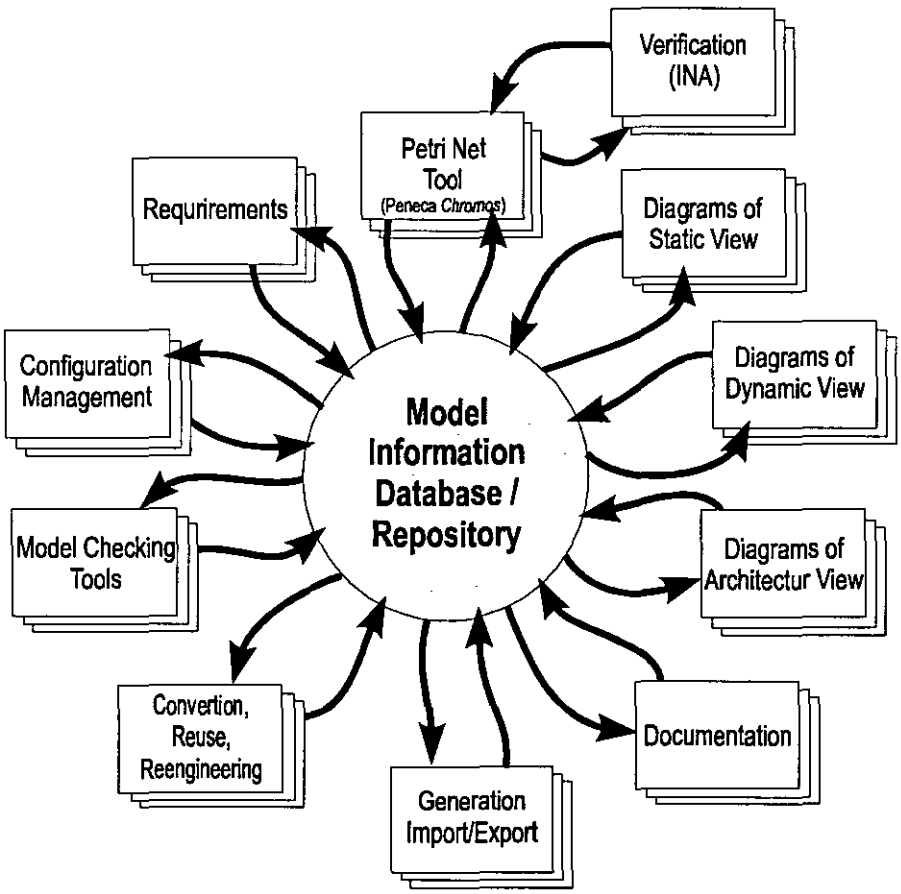


Figure 11 : The tool family combined within the OTW®2

The *OTW®2* bases on a repository. Inside this repository all necessary information is stored about modelling and development. The repository contains the whole model and hence it is the base for the internal consistency of the various parts of the model.

All the tools integrated in the *OTW®2* have access to the data stored in this repository. The several diagrams define different views to the content of the repository. Other tools like scanner and generator read objects from the repository or write new objects into it.
The information about the static and dynamic aspects of the system which are contained in the several diagrams can be used for checking and verifying the model with tool support. With respect to the existence of a model or a model together with its application, there are four different possibilities for testing the model ([Wolf95]): Evaluation of diagrams, static checking of consistency, active model-checking and passive model-checking.

Evaluation of diagrams

This means static check of basic properties of the model's composition. The evaluation of ergonomic properties of diagrams bases on a set of fuzzy rules.

Static check of consistency

This comprises the consistency checking of the static structure between the model and the application software. The use of a CASE tool is recommended for this step.

Active model-checking

Simulating an OPN is comparable to playing the token game in Petri Nets: if all the pre- and post-conditions of a process are fulfilled it will run. All the objects contained in the instance list of an object can be considered like the structured tokens in Petri Nets. Therefore, a process can be activated several times if there is more than one object fulfilling the conditions. For avoiding problems by multiple access to the same instantiated object a lock-mechanism is realised. If conflicts occur they can be solved by interpreting the priorities given to the processes.

Passive model-checking

Passive model-checking deals with the checking of the consistency between the actual model and the application software during the execution of the application software. The coupling between model and the application software allows the control of the modelled restrictions during the execution of the application software [Burk94]. For the OPN the pre- and post-conditions can be observed. A possible approach for realising such a dynamic check of consistency contains the following four steps:

- Generation
  Insertion of a special checking-code into the application software
- Compilation
  Creation of an runable application with the integrated check-mechanism
- Run
  Observation of the fulfilment with the required restrictions
- Adjustment
  Detection of inconsistencies, change of model or application software

In addition to these checks a special checking of business process models based on the AD is possible. This can be done by testing the compliance with some modelling rules, which were established by the authors. Every modelling element of an AD has an adequate Petri Net construct. All this rudimentary nets form a construction kit for a Petri Net. Therefore, a created AD can be automatically transformed into an ordinary Place Transition Net. If there are no violations against the modelling rules the resulting nets are sound workflow nets [Aals97]. Checking the existence or absence of violations against the modelling rules can be done by applying some simple reduction

rules to the nets. We use INA for doing the net reduction [INA]. The net reduction releases PTP-sequences and parallel nodes from the net. Nonconformance in the design results in typical structures within the reduced Petri Net, hence fault detection is possible.

6 Summary and further work

The UML is the first promising approach for standardisation in the field of object oriented modelling techniques. The diagrams permit the construction of system models in an incremental way, whereas the level of formalisation can be staggered according to the progress of the modelling process. With the OPN an additional means of description for the dynamic aspects of a system is provided expanding the facilities for simulation and for model verification.
At the moment the implementation of a simulating tool for the OPN is going on. This tool can be enhanced by some possibilities for statistical evaluation of simulation in order to do business process cost accounting. Furthermore, some work has to be done for automatic transformation of OPN into high-level Petri Nets in order to achieve the opportunity to apply formal analysis techniques. Eventually it is possible to find out some analysing rules directly for the OPN.
Another piece of work which has to be done is examining whether it makes sense to use reduction rules directly for the AD without transforming it into Place Transition Nets. The goal is simplification of the modelling and verification process for the user without knowledge about Petri Nets. But maybe by providing a meta-model for business process modelling based on the UML this verification step is not necessary. Using this meta-model simplifies the modelling work for the user and increases the quality of created models.

7 References and Links

[Aals97] van der Aalst, W. M. P.: Verification of Workflow Nets. LNCS 1248, S. 407-426, Springer Verlag. 1997

[Burk94] Burkhardt, R.: Modellierung dynamischer Aspekte mit dem Objekt-Prozeß-Modell, Dissertation. TU Ilmenau 1994 (in German)

[Burk97] Burkhardt, R.: UML – Unified Modeling Language. Addison-Wesley 1997 (in German)

[Conc97] High-level Petri Nets – Concepts, Definitions and Graphical Notation, Committee Draft ISO7IEC 15909, October 2, 1997, Version 3.4

[FeSi96] Ferstl, O., Sinz, E.: Geschäftsprozeßmodellierung im Rahmen des Semantischen Objektmodells, Geschäftsprozessmodellierung und Workflow-Management - Modelle, Methoden, Werkzeuge (Herausgeber: Vossen, G.; Becker, J.); International Thomson Publishing; Bonn/Albany 1996 (in German]

[INA] Integrated Net Analyzer, 2.1, http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/ (in German)

[Jaco+92] Jacobsen, I., Christerson, M., Jonsson, P., Övergaard, G.: Object-Oriented Software Engineering, A Use Case Driven Approach. Addison Wesley. 1992.

[Jaco+95] Jacobsen, I., Ericcson, M., Jacobsen, A.: The Object Advantage. Business Process Reengineering with Object Technology. Addison-Wesley. 1995.

[Möld96] Mölders, A., Fengler, W., Burkhardt, R., Philippow, I.: Workflow Configuration Using the Object Process Model, in: Proceedings of the 1. International Conference PAKM, Basel Oktober 1996

[OCL] Booch, G., Jacobson, I., Rumbaugh, J.: Unified Modeling Language, Object Constraint Language Specification, Version 1.1, Rational Software Corporation, www.rational.com, 1. September 1997

[OTW97] Objekttechnologie-Werkbank, Benutzerhandbuch OTW 2, OWiS Software GmbH Martinroda/Ilmenau, 1997 (in German)

[PMK97] Petersen, U., Mölders, A., Köhler, T.: Optimierung von Geschäftsprozessen durch objektorientierte Modellierung und Simulation, in: Proceedings of "Workflow-Management in Geschäftsprozessen im Trend 2000", Schmalkalden, 15./16. Oct. 1997 (in German)

[Pres97] Prescher, G.: Prozeßmodelle für Workflow-Management – ein Ansatz mit Use Case, in: Proceedings of "Workflow-Management in Geschäftsprozessen im Trend 2000", Schmalkalden, 15./16. Oct. 1997 (in German)

[Schm97] Schmutterer, A.: Theoretische Formalisierungen verschiedener objektorientierter Workflow-Modellierungstechniken, Diplomarbeit (Master Thesis) TU Ilmenau, 1997 (in German)

[UML1] Booch, G., Jacobson, I., Rumbaugh, J.: Unified Modeling Language, Notation-Guide, Version 1.1, Rational Software Corporation, www.rational.com, 1. September 1997

[UML2] Booch, G., Jacobson, I., Rumbaugh, J.: Unified Modeling Language, UML-Semantics, Version 1.1, Rational Software Corporation, www.rational.com, 1. September 1997

[WBP98] Wolf. M., Burkhardt, R., Philippow I.: Software Engineering Process with the UML. in: Schader, M., Korthaus, A. (eds.): The UML. Technical Aspects and Applications. Physica-Verlag. 1998.

[WfMC96] Workflow Management Coalition: Terminology & Glossary; Document Number TC 1011; Issue 2.0; Brüssel, Juni 1996.

[Wolf95] Wolf, M.:Validierung und Optimierung dynamischer Modelle auf der Basis des Objekt-Prozess-Modells, Technische Universität Ilmenau, Diploma Thesis, 1995 (in German).

Dipl.-Math. Angela Mölders, Dipl.-Inf. Martin Wolf, Prof. Dr.-Ing. habil. Wolfgang Fengler
Technical University of Ilmenau, Faculty of Informatics and Automation,
Institute of Theoretical and Technical Informatics
P.O. Box 100 565, 98684 Ilmenau, Germany
eMail: [wfengler|moelders|mwolf]@theoinf.tu-ilmenau.de
URL: http://www.theoinf.tu-ilmenau.de/~moelders/

Dr.-Ing Rainer Burkhardt, OWiS Software GmbH –Society for object-oriented and knowledge-based systems Ltd.
Lindenstrasse 28, 98693 Ilmenau, Germany
eMail: rsb@otw.de
URL: http://www.otw.de

Reuse-oriented Workflow Modelling with Petri Nets

Gerrit K. Janssens, Jan Verelst, Bart Weyn

University of Antwerp - RUCA
Information Systems & Operations and Logistics Management
Middelheimlaan 1, B-2020 Antwerp, Belgium
E-mail: {gerritj, verelst, bweyn}@ruca.ua.ac.be

Abstract

Several authors propose their own technique based on Petri Nets to model Workflow processes. Most of them recognise the adaptability problem inherent to workflows, viz. the frequently and/or radically changing character due to changing business process rules, but suggest totally different solutions. Because the proposed techniques are fundamentally different, eleven of these techniques are briefly discussed and compared. Next, we survey approaches to reuse in the workflow field and we classify them in a framework derived from the information systems literature.

1. Introduction

Recently, both the domain of workflow modelling by using Petri Nets and the area of reuse in software engineering have gained much attention. We share the opinion that it could be opportune to take a closer look at the application of the reuse concept to workflow modelling. It is our intention to make an informal introduction to the subject and an attempt to make a broad outline of desirable future developments and some topics that need further research. First we summarise the different topics concerning this field that have already been covered and treated by several authors.

The domain of workflow management seems to be characterised by lacking precise definitions. Because of the wide variety of definitions used by various authors concerning workflow, tasks, procedures etc., we define the most important terms in section two.

2. Definition of Basic Workflow Concepts

Little agreement exists upon what workflow exactly stands for and which specific features a workflow management system must provide. For an overview of existing definitions and interpretations of workflow and workflow management systems we refer to Georgakopoulos et al. [21]

In response to the proliferation of definitions, the Workflow Management Coalition (WfMC) performs considerable efforts to standardise terminology. The WfMC [25] is a non-profit international organisation which formal mission is to promote the use of workflow through the establishment of standards for workflow terminology, interoperability and connectivity between workflow products. The WfMC has also specified a reference architecture for workflow technology. Since its establishment in August 1993 it has grown to over 100 members, consisting of workflow vendors, users and analysts.

The formal definition of workflow presented by the WfMC is as follows:

The computerised facilitation or automation of a business process, in whole or part.

The definition of a Workflow Management System (WFMS) is consequently:

A system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

As various definitions of workflow are based upon the concepts of tasks, activities, procedures and work steps, we look here at the most common definitions. We refer to the definitions provided by Ellis and Nutt [13]. A task or procedure can be defined as a predefined set of work steps, and partial ordering of these steps. A work step consists of a header and a body. The work step contains identification, precedence, etc. and the body represents the actual work to be done. In this way an activity can be defined as the body of the work step of a procedure. An activity can then be either a compound activity, containing another procedure, or an elementary activity [13].

However, in the following we use the definitions proposed by the WfMC.

3. Workflow Concepts Translated into Petri Nets

Since Zisman [51] used Petri Nets to model workflow processes for the first time in 1977, several authors have made attempts to model workflows in terms of Petri Nets, amongst which De Cindio et al. [12], Ellis and Nutt [13,14] [39], van der Aalst [3],

Ferscha [16], Wikarski [48], Li et al. [34], Adam et al.[7], Oberweis et al. [40], Badouel and Oliver [9], Merz et al. [36,37], and Schömig and Rau [44].

3.1 Why Petri Nets to Model Workflow?

The execution structure of a workflow specifies the ordering constraints for the executions of the tasks. (example: "a task may not begin before a particular previously started task commits"). Such constraints can be specified by triggers, by finite state automaton or by Petri Nets. [33]

Van der Aalst [2] identifies mainly three reasons for using Petri Nets for workflow modelling. The first reason is the fact that Petri Nets possess formal semantics despite their graphical nature. The second reason is that instead of being purely event-based, Petri Nets can explicitly model states. In this way, a clear distinction can be made between the enabling and execution of a task. The final reason lies in the abundance of available and theoretically proven analysis techniques.

Oberweis et al. [40] identify five different reasons to opt for using Petri Nets when modelling workflows. They are:

- Integration of data and behaviour aspects,
- Support for concurrent, cooperative processes,
- Different degrees of formality,
- Availability of analysis techniques,
- Flexibility.

By this last property the authors refer to the fact that Petri Net models are directly executable by an interpreter (which is the workflow engine of the Petri Net based WFMS), but they are not so-called "hard-wired" application programs. This guarantees a reasonable degree of flexibility, because of the "late-binding" between activities and objects. This way it becomes possible, according to the authors, to make adjustments to workflow processes at run-time.

Merz et al. [36] finally state that the combination of a mathematical foundation, a comprehensive graphical representation, and the possibility to carry out simulations and verifications is the main strength of Petri Nets when modelling workflows.

Han [23], however, warns and states that despite the popularity of Petri Nets to model workflows, he does not believe that Petri Nets are directly applicable for modelling workflows, mainly due to their fixed structures. The author criticises the lack of flexibility of most of the proposed net models and indicates the mechanisms to support abstraction and compositionality as the main reason.

3.2 High Level Versus Low Level Petri Nets

In this section we give a brief overview of the Petri Net classes proposed by various authors. Because of the problematic nature of modelling business processes, Petri Nets in their conventional form are not well suited as a modelling language. Common problems encountered when modelling workflows include high complexity when dealing with other than just toy models and lack of flexibility, especially where structural changes are necessary.

As already mentioned, the structure of workflows is extremely volatile as a consequence of changing business process rules. Business environment and conditions change very quickly. System evolution is unavoidable because business processes evolve continuously caused by internal organisational reforms, external environmental changes, etc.

Hence, business models are subject to mainly two types of changes: on the one hand changes in the data of the workflow systems and on the other hand changes in the rules of the workflow systems.

All the above resulted in various authors proposing their own developed Petri Net class in order to cope with those specific issues. We synthesised different approaches of various authors and drew up a comparison in Table 1.

Author	Petri Net class	Brief description
Van der Aalst W.M.P [1]	Workflow-nets (WF-nets)	Abstraction into P/T-nets of High Level Petri Nets with two special places *i* and *o*, indicating beginning and end of the modelled business procedure.
Ellis C.A., Nutt G.J. [13]	Information Control Nets (ICN)	High Level Petri Net variant intended to represent control flow and data flow.
Oberweis A., et al. [40]	INCOME/WF	High Level Petri Nets are used to describe the so-called core workflows on a relatively abstract level.
Adam N. R., et al. [7]	Temporal Constraint Petri Net (TCPN)	Ordinary Petri Net extended with an interval function and a timestamp function to model absolute as well as relative time.

Wikarski D. [48]	Modular Process Nets	Low Level Petri Nets provided with a hierarchic module concept and with constructions designed to realise communication between net instances and their environment and constructions to create and destroy the net instances.
Agostini A., et al. [8]	Subclass of Elementary Net Systems	A WFMS consists of two basic components: namely a WF model and a WF Execution model. Simplicity of the WF model is stressed because it enhances the flexibility and adaptability of the WF Execution Module.
Schömig A.K., Rau H. [44]	Coloured Generalised Stochastic Petri Nets (CGSPN)	CGSPN are used as an adequate tool to measure performance and to model dynamic behaviour.
Merz M., et al. [36]	Coloured Petri Nets (CPN)	CPN are used to introduce dynamic workflow modelling in the distributed systems architecture COSM
Ferscha A. [16]	Generalised Stochastic Petri Nets (GSPN)	GSPN are used to model and quantify WF (performance and structural analysis).
Badouel E., Oliver J. [9]	Reconfigurable Nets	Extension of the WF-nets of van der Aalst [1], intended to support dynamic changes in Workflow systems.
Wikarski D., Han Y., Löwe M. [49], [23]	Higher Order Object Nets (HOON)	In contrast to Modular Process Nets [48], which are used to model workflow processes, this approach is intended to model explicitly the structure of the organisation and the organisational resources.

Table 1: Overview of the proposed Petri Net classes

Li et al. [34] also use a Petri Net variant to model office work. However, they are not mentioned in the table because of the limited scope of their paper concerning the introduction of Petri Nets to model procedural knowledge. Moreover they do not really go into the inherent problems of modelling workflows. The authors propose the Activity Manager System (AMS), which is a domain-independent formalism designed to hierarchically represent procedural knowledge. Activity Networks (AN) are defined, which are used to model the activities in office procedures. These AN can be transformed into Petri Nets, in this way benefiting from the various levels of abstraction and being capable of handling notions such as sequence, choice and concurrency, as offered by Petri Nets.

3.3 High Level Petri Nets

Ellis and Nutt [13] as well as van der Aalst [6] make a resolute choice in favour of High Level Petri Nets. They both state that High Level Petri Nets are an indispensable necessity when modelling real world applications because Low Level Petri Net models tend to become extremely complex and very large. Moreover, Ellis and Nutt [13] state that, when modelling large sets of office procedures, Low Level Petri Nets lead to "an exponential explosion" of the model.

The Workflow nets (WF-nets) proposed by van der Aalst [1] are an abstraction into P/T-nets of High Level Petri Nets with two special places i and o, indicating the beginning and the end of the modelled business procedure. These WF-nets are suitable not only for the representation and validation but also for the verification of workflow procedures.

The question: "Given a marked Petri Net graph, what structural changes can or cannot be applied while maintaining consistency and correctness", is an important and pressing problem which has also been recognised by Ellis and Nutt [13]. However, van der Aalst [4] provides an answer to this question for Workflow nets in the shape of transformation rules. These rules should not be confused with the more common reduction rules. Eight basic transformation rules allow the designer to modify sound WF-nets while preserving their soundness.

Badouel and Oliver [9] extend the WF-net formalism of van der Aalst [1] and propose the Reconfigurable Nets. These Reconfigurable Nets intend to support dynamic changes in Workflow systems. A Reconfigurable Net consists in fact of several Petri Nets which constitute the different possible configurations of the system. Each configuration gives a description of the system for some mode of operation. The authors denote that Reconfigurable Nets are self-modifying nets, meaning generalisations of P/T-nets where the flow relation between a place and a transition depends on the marking. The authors conclude by stating that it might be interesting to extend a Reconfigurable net with a control part to regulate the flow in the system.

Ellis and Nutt [13], [14], [39] propose Information Control Nets (ICN), derived from High Level Petri Nets to represent office workflows. By adding a complementary data flow model, generalising control flow primitives and simplifying semantics, ICN are in fact a generalisation of Coloured Petri Nets. ICN represent control flow as well as data flow. The authors provide an exception handling mechanism. They note, however, that the mechanism allows users to escape the model, rather than helping them to analyse and cope with the exceptions.

Finally, Merz et al. [36,37] use Coloured Petri Nets in order to enhance the distributed systems architecture Common Open Service Market (COSM), with concurrent workflow modelling. The authors introduce Coloured Petri Nets as a modelling and simulation technique for concurrent activity management and control. They use the definition of Coloured Petri Nets of Jensen [27], which is probably the most common one. The tokens of Coloured Petri Nets are typed (coloured) entities or data values. Places, transitions, edges, etc. are typed by their respective signature. Places represent data stores containing an arbitrary number of data values (tokens) of their respective type.

3.4 Stochastic Petri Nets

Ferscha [16] proposes Generalised Stochastic Petri Nets (GSPN) to model workflows. Stochastic Petri Nets (SPN) are in the classical approach based on Place/Transition (P/T) Petri Nets but stochastic extensions are added. With SPN, the set of transitions only contains stochastic timed transitions where the firing delay random variable is exponentially distributed.

Generalised Stochastic Petri Nets are a class of SPN and allow to model timed transitions with exponentially distributed delays as well as immediate transitions. Transition priorities are used to avoid confusion and undesirable conflicts. Timed transitions have the lowest priority level and immediate transition weights define how transitions from extended conflict sets should be fired. Ferscha [16] exploits the natural correspondence between the GSPN enabling and firing rules and the dynamic behaviour of workflow systems. With respect to quantitative analysis, the Markovian framework is used within the GSPN formalism to derive the performance metrics. For qualitative analysis the author refers to the available broad body of Petri Net structural analysis techniques.

Schömig and Rau [44] propose a variant of the above GSPN, i.e. the Coloured Generalised Stochastic Petri Nets (CGSPN). CGSPN are based upon Coloured Petri Nets as pure Petri Net formalism instead of Place/Transition Petri Nets. Compared to the classical approach which is based on P/T Petri Nets, this approach requires more sophisticated analysis techniques.

The authors [44] suggest an approach to model the dynamic behaviour of workflow in which resources are explicitly modelled as separate places. They state that CGSPN are

appropriate to model the typical characteristics of workflows, e.g. dynamic routing, simultaneous resource allocation, forking/joining of process control threads and random-order or priority based queueing disciplines. The advance of the control flow is modelled by a separate token, a so-called job token, which traverses the complete Petri Net, enabling activities according to the business rules.

In order to cope with dynamic routing, all possible routing paths through the process model are exhaustively determined. All jobs following the same path are collected in so-called job classes. Each of the paths can be modelled by a separate Petri Net. By associating a colour or an attribute to the job tokens, the job class membership can be specified. The distinct Petri Nets can be mapped into one single Petri Net model, which then also captures the routing constraints accordingly. The synchronisation problem of split/join constructs is solved by attributing an identifier to a job token. In this way, job tokens belonging to different jobs can be prevented from being erroneously synchronised to a single job. The authors also generate and solve the Markov chain from the underlying random process to derive the performance data by a Petri Net analysis tool.

3.5 Low Level Petri Nets

Wikarski [48] on the contrary, starts from the observation that in the recent past a vast variety of Petri Net classes came into existence that aimed at increasing the expressiveness of the net models. This gave rise to the existence of more complex Petri Net classes with various sorts of tokens, arc or place inscriptions and many other extensions of the classic Petri Net.

He further states that the arrival of High Level Petri Nets has created a number of problems the first of which is the reduction of the intuitive aspect when modelling by means of Petri Nets. An important characteristic of classic Petri Nets is that they can be learnt easily and that they are very communicative with respect to persons with no specific knowledge of Petri Nets. Two other problems are: (1), the impossibility to describe dynamic or changing behaviour and, (2), the communication of the active nets (i.e. those enacted by an interpreter) with each other and the rest of their environment.

To counter the above stated problems, Wikarski proposes Modular Process Nets, which can be described as Elementary Net Systems with minimal syntactic extensions. Elementary Net Systems (EN-systems) have originally been introduced by Rozenberg and Thiagarajan [43].

When using the three level subdivision of basic net models proposed by Bernardinello et al. [10], EN-systems can be catalogued as first level Petri Nets. The authors use the nature and the number of tokens in one place as a discriminating characteristic. Following this line they define a third level net system as a system in which places are marked by structured tokens. These nets are often denoted by the term High Level Petri Nets. The second level concerns nets with places that are marked by several

unstructured tokens. In this way, places represent in fact counters. Places of first level net systems are marked by one unstructured token at the most: hence places actually represent conditions.

Modular Process Nets can be characterised best as a generalisation of safe EN-systems, i.e. they have untyped tokens with a maximum of one token in any one place. One of the main aims of Modular Process Nets is that the formalism should be simple, easy to learn and comprehensible in order to be used as a widespread but formally precise means of communication. For this purpose, the author has developed a whole range of node types. The author defines three different kinds of places: (ordinary) places, fusion places and channels. Fusion places are used to improve the graphical representation of large nets and channels are used to support communication between nets via token passing. Moreover, the author defines seven transition types which deal with the handling of events. Wikarski states that the main and most innovative points of the nets are the introduction of a hierarchical module concept for nets and the definition of "elementary process nets".

Like Wikarski [48], Agostini et al. [8] plead for simplicity of workflow modelling and also opt for Elementary Net Systems. Their final objective is to create a workflow model that allows its users to design workflows having little or no experience with computer science, programming or formal languages. For this purpose, they define a subclass of these Elementary Net Systems. The authors stress that EN-systems possess adequate mathematical properties which allow the modeller to generate a large class of behaviours. They state that a WFMS consists of two basic components: a WF model and a WF Execution model. Simplicity of the WF model is stressed because it enhances the flexibility and adaptability of the WF Execution Module.

Adam et al. [7] state that an ordinary Petri Net fulfils the basic needs to model the control flow and value dependencies of a workflow system. In order to model the temporal dependencies between two tasks in a workflow, however, the authors propose a Temporal Constraint Petri Net (TCPN). According to the authors, existing Timed Petri Nets are not capable of modelling both relative and absolute time. Their functionality is limited to modelling relative time. The definition of a TCPN states that each place and each transition is associated with a time interval and a token with a time stamp.

The authors develop a notion of equivalence in order to be able to check whether the Petri Net model represents the specified workflow correctly. Concerning workflow analysis, the authors provide three types of analyses. A first analysis which can be performed aims to identify whether inconsistent dependency relations among tasks exist. A second type of analysis aims to test for workflow safety, which means to check whether the workflow terminates in an acceptable state. And finally the third type of analysis aims to test the temporal feasibility. This means to check, for a given starting time, whether a workflow is schedulable with the specified temporal constraints.

3.6 Petri Nets extended with object-oriented concepts

Modular Process Nets proposed by Wikarski [48], have sensor transitions which can detect triggering signals from the external environment. These signals, however, only contain control information in a predefined context. In contrast to Modular Process Nets, resource management is explicitly embodied in Higher-Order Object Nets (HOON), the other formalism proposed by Wikarski et al. [49]. The central idea of HOON is to arrange net models and their surrounding environments in a client/server manner and to model the client/server interfaces explicitly [23].

In HOON, each transition can own an interface place, which is called a control place. Each control place may have several control tokens and through the exchange of control tokens, a specified net can interact with the external world. Tokens can be put on control places by software tools, either manually or by another net, depending on the control policy. For modelling and controlling workflows, the authors propose two different classes of nets, i.e. working nets and control nets. The interfaces between those two classes of nets are the control places. Working nets can be compared to Coloured Petri Nets.
In the next paragraph we focus on the question in which of the Petri Net classes the reuse concept fits best.

4. The Reuse Concept for Workflow Modelling

4.1 Real World Workflow Modelling

In contrast to the field of software engineering where the concept of reuse is widely explored, few authors have developed a theoretical framework to reuse in workflow modelling by Petri Nets. Nevertheless, the concept of reuse is definitely encountered or applied in practice by many modellers of real world workflows. This is due to the fact that specifying and modelling real world workflows is highly complicated and complex and that they are usually not developed in a single step.

As Oberweis [41] stated, there are mainly two potential strategies for the development of large, real world workflow models. A first strategy is incremental construction by iteratively refining, evaluating and formalising net fragments. This strategy can be based upon composing certain elementary net building blocks from an existing Petri Net library. Another strategy is adapting application-specific reference process models and reference object models to the requirements of a specific case. These application-specific reference models are sometimes denoted as generic models because they have always captured a certain generic process knowledge.

In both cases, the importance of a well-documented library in which the reference models or the Petri Net fragments are stored cannot be underestimated in any way. For the whole concept of both approaches is based upon the library, the quality of the library is a discriminating factor between failure or success of both systems.

Van der Aalst [1] also states that when dealing with the high complexity of real world workflows, designers can refer to reuse on the basis of hierarchical decomposition, especially in communicating with end-users.

4.2 Approaches to Reuse of Petri Nets for Workflow Modelling

In this section, the existing approaches to reuse of Petri Nets for workflow modelling are discussed and classified into a framework used in the information systems (IS) literature (see Table 2).

The classification used in this paper is a summarised version of the classification framework by Krueger [28], which was later adopted and refined by Mili et al. [38]. We chose this framework because, as Mili explicitly states, it focuses on the « paradigmatic differences between the various reuse methods ». A classification according to fundamental differences allows us to explore to what extent current approaches to Petri Net-reuse cover the whole reuse-spectrum.

Krueger's [28] framework distinguishes between two main types of reuse: the building blocks approach (compositional approach) and the generative approach. The building blocks approach is further subdivided into the reuse of software patterns and into software architectures.

4.2.1 Patterns

A (software) pattern is a proven solution to a certain standard type of problem. It is described by four essential elements:

- a pattern name
- a problem description, which clarifies in which situations the pattern can be used
- the solution to the problem
- the consequences and trade-offs involved in applying the pattern.

A limited amount of design patterns [19] and analysis patterns [18] [24] have been published. An example of an analysis pattern is an Object-Oriented(OO)-conceptual model for the concept of a 'customer' or a 'bookkeeping account'. Although these patterns tend to be domain-specific, many of them can be used outside of their original domains [18]. For example, a pattern of a bill of material can also form the basis for modelling an organisation's hierarchy. Design patterns are situated at a lower level of abstraction. An example is the observer-pattern [19]. An observer is an object which

monitors the state of a certain 'subject'. When the subject changes state, the observer notifies all interested objects of this change. A typical application of the observer pattern is found in spreadsheets. When a graph is produced based on data in a spreadsheet, it is important that the graph is notified of any changes in the underlying data. The observer-pattern describes how an observer can be built to achieve this goal.

Many of the existing approaches to reuse of Petri Nets for workflow modelling can be interpreted as reuse of a pattern. Especially approaches discussing compositionality of Petri Nets fall into this category: these authors implicitly assume that some existing elements (for instance, workflows) are composed. We interpret these existing elements as patterns.

However, before we enumerate which authors fall into this category, we add a level in the classification: black-box vs. white-box reuse of patterns.

Black-box reuse is defined as the reuse of existing software components without any modification. White-box reuse does allow adaptation of the components, usually using the mechanism of inheritance.

In the IS-literature, a preference for black-box reuse has developed over the years. For instance, Fayad [15] claims that black-box reuse leads to systems that are easier to use and extend. Zweben [52] provides experimental evidence: his experiments show that black-box reuse is superior to white-box reuse in terms of required effort and correctness of the resulting system. The main disadvantage of white-box reuse is that the inheritance mechanism violates the encapsulation-principle. The aim of this principle is to minimise interdependencies between modules by defining strict interfaces. A subclass, however, has access to some of the data and code of its superclass. The subclass is allowed to change the values of these data items, to call functions of the superclass etc. As a consequence, several dependencies between the super- and subclass are introduced. These dependencies compromise reusability, as changes in a superclass frequently induce changes in the subclass. [19, 46].

In the context of Petri-Nets, white-box reuse is discussed by Lakos [30] [31], who defines the notion of inheritance for Object Petri Nets. Black-box reuse, through the notion of compositionality of Petri Nets, is discussed by Christensen [11], Han [23], Holvoet [26], Kruke [29], Wikarski [48] and van der Aalst [1,3].

For example, van der Aalst [1] briefly draws attention to reuse of WF-nets on the basis of 'task refinement' which is the refinement of a task by a subflow. In this way it becomes possible to decompose a complex workflow into subflows which, in their turn, can be built up from other subflows. In other words, one achieves a hierarchical decomposition.

Compositionality is an important property for hierarchical construction of WF-nets and more in particular for the reuse of subflows. The author [1] proves seven characteristics

about compositionality with regard to verifying the correctness of subflows in the same way as verifying the entire workflow on a more abstract level.

4.2.2 Software Architectures

A Software Architecture is a high-level design of a software system, i.e. the subsystems and their interactions [45]. Examples of architectures are compiler architectures (consisting of analysers, parsers and code generators), database architectures and rule-based architectures for expert systems. Software architectures are similar to very large-scale patterns. However, patterns tend to focus on a small part of a system whereas an architecture contains the overall structure of the system.

In the context of reuse of Petri Nets for workflows, both Han [23] and van der Aalst [3] define software architectures for workflow management systems.

4.2.3 Application generators and very high-level languages

Application generators and very high-level languages constitute the class of generative reuse. Forms of generative reuse are based on reusing the process of previous software development, rather than reusing existing products (such as patterns or software architectures) [38].

Application generators and very high-level languages allow the user to specify the requirements at a very high level of abstraction. From these requirements, code is automatically generated. This approach to reuse is considered, in the long term, to have the highest potential payoff. However, at the current moment, it remains very difficult to build generators that scale up to industrial production [42].

In the context of Petri Nets for workflow modelling, only van der Aalst [3] describes a number of Petri Net tools that belong to this category.

4.2.4 Evaluation

By far the most common approach to reuse of Petri Nets for workflow modelling is black-box reuse of patterns. Most authors discuss this kind of reuse implicitly through the notion of compositionality.

However, very few authors, if any, discuss the notion of reusability explicitly and/or in great detail. In other words, questions such as which advantages exactly can be achieved or which type of reuse leads to these advantages, remain unanswered.

Reuse Type		Authors
Software Patterns	Black-box reuse	Christensen [11], Han [23], Holvoet [26], Kruke [29], Wikarski [48], van der Aalst [3]
	White-box reuse	Lakos [30]
Software Architectures		Han [23], van der Aalst [3]
Application Generators and very high level languages		van der Aalst [3]

Table 2: A classification of techniques for reuse of Petri Nets for workflow modelling

4.2.5 A critical remark concerning reuse in the IS-literature

The idea of building software by assembling reusable components dates back to 1968 when Doug McIlroy proposed the idea of libraries of shared components at the NATO Conference of Software Engineering [35].

Since then, most programmers have continued to informally reuse their own code, but in an ad hoc way. Up to now, it has remained extremely difficult to realise a *systematic* approach to reuse [42][32][17]. Also, Prieto-Diaz [42] observes that the state-of-the-practice is still source code reuse, in spite of numerous claims that reuse at the design- or even analysis-level would have higher payoffs. Finally, Szyperski [47], for example, observes that at this moment, relatively few catalogues of reusable objects actually exist.

The literature contains a wide variety of potential reasons for the lack of systematic reuse: some technical, but many are managerial (relating to management commitment, organisational issues etc.) [50]. We now focus on one of the fundamental technical problems that underlie reuse.

4.2.6 Hidden assumptions

A fundamental problem of software reuse is the problem of the hidden assumptions. This problem refers to the fact that software components make assumptions about their intended environment which are implicit and either don't match the actual environment or conflict with those of other parts of the system. Such conflicting assumptions make reuse extremely difficult or even impossible [20].

For example, Garlan [20] describes an example in which several software packages were combined in order to build a software engineering tool. However, the assumptions that the different packages made about which program held the main thread of control, were incompatible, which drastically complicated building the new system. As these assumptions tend not to be documented, they are extremely difficult to detect when deciding which software components could be reused.
Glass [22] provides an example in which a sort program was reused. However, the program performed extremely slowly when sorting strings. The undocumented assumption was that the structure of the sort program was far more appropriate for sorting numbers than strings.

Garlan [20] suggests possible solutions for the hidden assumptions-problem: amongst others, make architectural assumptions explicit, provide techniques for bridging mismatches between assumptions and develop sources of architectural design guidance. Although we agree with these suggestions, it is clear that these solutions are more workarounds to the problem than an elimination of it.

4.2.7 Final remarks

We have briefly shown in this paragraph that, in the IS-field, systematic reuse has been pursued for up to 30 years, but that the practical state-of-the-art is still rather disappointing. Realising a systematic form of reuse has proven to be a very ambitious goal with a wide variety of problems (technical and managerial) along the way. Good modelling constructs alone (such as objects) have been insufficient to reach this goal.

It is our impression that the field of workflow modelling with Petri Nets is currently making quick progress towards deciding which modelling constructs are most appropriate. In order to determine whether this will be sufficient to realise systematic reuse in the workflow field, empirical and experimental studies are required. We have yet to find these in the literature.

5. Conclusion

In this paper we have tried to identify the existing Petri Net formalisms proposed by various authors used for modelling workflow. We found that, at the moment, there is not yet unanimity about which class of Petri Nets suits best the specific needs of workflow modelling. Especially the different approaches between Low Level Petri Nets and High Level Petri Nets can in this view be mentioned as exemplary.

An interesting remark, however, is that even a very good Petri Net formalism for modelling workflows is not worth much if there are no Workflow Management Systems or other computer tools based on it. This has also been stated by van der Aalst [5]

concerning the usability of High Level Petri Nets. The author [5] notes that the availability of adequate computer tools is a critical factor in the practical use of High Level Petri Nets and related analysis methods.

Compared to database models, workflow models are far from being mature. In response to the need of coming to a standard in workflow modelling (like in the field of conceptual modelling with Entity Relationship Modelling which has also been formulated by van der Aalst [1],) we would like to make a remark. When speculating about the best potential formalisms to serve as a possible standard, it is likely that the Petri Net formalism, which is best supported by computer tools turns out to become the standard.

With respect to reuse, much progress is being made towards developing an adequate modelling construct for modelling workflows using Petri Nets. However, we have the impression that adequate modelling constructs alone were not sufficient to achieve systematic reuse in the IS-field. Whether the same applies to the workflow field should be further investigated.

References

[1] W. M. P. van der Aalst, "Structural Characterizations of Sound Workflow Nets", Eindhoven University of Technology, Computing Science Reports 96/23, 1996.

[2] W. M. P. van der Aalst, "Three Good Reasons for using a Petri Net based Workflow Management System", in Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC '96), T. Wakayama, S. Kannapan, C. M. Khoong, S. Navathe and J. Yates, Eds., Cambridge, Massachusetts, pp.179-201, 1996.

[3] W. M. P. van der Aalst, "The Application of Petri Nets to Workflow Management", The Journal of Circuits, Systems and Computers, pp. 1-53, 1998.

[4] W. M. P. van der Aalst, "Verification of Workflow Nets", in Proceedings of 18th International Conference, ICATPN'97; Toulouse, France; 23-27 Jun 1997, P. Azema and G. Balbo, Eds., vol. 1248 of Lecture notes in Computer Science, Application and theory of Petri nets 1997, Springer-Verlag, pp. 407-426, 1997.

[5] W. M. P. van der Aalst, and K. van Hee, "Framework for Business Process Redesign" in Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 95), J. R. Callahan, Ed., IEEE Computer Society Press, Berkeley Springs, pp. 36-45, 1995.

[6] W. M. P. van der Aalst and K. van Hee, "Business Process Redesign: A Petri-net-based approach", Computers in Industry, vol. 29, no. 1-2, pp. 15-26, 1996.

[7] N. R. Adam, V. Atluri, and W. K. Huang, "Modeling and Analysis of Workflows Using Petri Nets" Journal of Intelligent Information Systems: Special Issue on Workflow and Process Management, M. Rusinkiewicz and S. H. Abdelsalam, Eds., vol. 10, no. 2, pp. 1-29, 1998.

[8] A. Agostini, G. De Michelis and K. Petruni, "Keeping Workflow Models as Simple as Possible", in Proceedings of the Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms within the 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain, June 21st, pp. 11-29, 1994.

[9] E. Badouel and J. Oliver, "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems", Publication Interne IRISA PI 1163, 1998.

[10] L. Bernardinello and F. De Cindio, "A survey of Basic Net Models and Modular Net Classes", G. Rozenberg, Ed., vol. 609 of Lecture Notes in Computer Science, Advances in Petri Nets 1992, Springer-Verlag, pp.304-351, 1992.

[11] S. Christensen and L. Petrucci, "Towards a Modular Analysis of Coloured Petri Nets", in Proceedings of the 13th International Conference Sheffield, UK, June 1992, K. Jensen, Ed., vol. 616 of Lecture notes in Computer Science, Application and Theory of Petri Nets 1992, Springer-Verlag, pp. 113-133, 1992.

[12] F. De Cindio, C. Simone, R. Vassallo and A Zanaboni, "CHAOS: a Knowledge-based System for Conversing within Offices", Office Knowledge Representation, Management and Utilization, W. Lamersdorf, Ed., Elsevier Science Publishers B.V., North-Holland, pp. 257-275, 1988.

[13] C. A. Ellis and G. J. Nutt, "Modeling and Enactment of Workflow Systems", in Proceedings of the 14th International Conference Chicago, Illinois, USA, June 1993, M. A. Marsan, Ed., vol. 691 of Lecture notes in Computer Science, Application and Theory of Petri Nets 1993, Springer-Verlag, pp. 1-16, 1993.

[14] C.A. Ellis and G. J. Nutt, "Workflow: The Process Spectrum", in Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, Athens, Georgia, pp. 140-145, 1996.

[15] M. A. Fayad and D. C. Schmidt, "Object-oriented Application Frameworks", Computers in Industry, vol. 40, no. 10, pp. 32-38, 1997.

[16] A. Ferscha, "Qualitative and Quantitative Analysis of Business Workflows using Generalized Stochastic Petri Nets" Oesterreichische NationalBank Austria Project No. 5096, 1997.

[17] R. Fichman, C. Kemerer, "Object Technology and Reuse : lessons from early adopters", IEEE Computer, pp. 47-59, October 1997.

[18] M. Fowler, Analysis Patterns: Reusable Object Models. Addison-Wesley, 1997.

[19] E. Gamma and R. Helm, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[20] D. Garlan, "Architectural mismatch: why reuse is so hard", IEEE software, pp 17-26, November 1995.

[21] D. Georgakopoulos, M. Hornick and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, no. 3, pp. 119-153, 1995

[22] R. Glass, "A word of warning about reuse", ACM SIGMIS Database, vol. 28 no. 2, pp. 19-21, spring 1997.

[23] Y. Han, "HOON - A Formalism Supporting Adaptive Workflows", Technical Report #UGA-CS-TR-97-005, Department of Computer Science, University of Georgia, November 1997.

[24] D. Hay, "Data model patterns: conventions of thought", Dorset House Publishers, pp. 268, 1996.

[25] D. Hollingsworth, "Workflow Management Coalition: The Workflow Reference Model" 4-29-1994, Brussels, Belgium.

[26] T. Holvoet and P. Verbaeten, "Petri Charts, An Alternative Technique for Hierarchical Net Construction" in Proceedings of the 1995 IEEE Conference on Systems, Man and Cybernetics (IEEE-SMC'95), pp. 1-19, 1995.

[27] K. Jensen, "Coloured Petri Nets": Vol. 1, Springer-Verlag, 1992.

[28] C. W. Krueger, "Software Reuse", ACM Computing Surveys, vol. 24, no. 2, pp. 131-183, 1992.

[29] V. Kruke, "Reuse in Workflow Modelling", Diploma Thesis, Information System Group, Department of Computer Systems, Norwegian University of Science and Technology, 1996.

[30] C. Lakos, "From coloured Petri Nets to Object Petri Nets", Technical Report TR94-9, Computer Science Department, University of Tasmania, 1994.

[31] C. Lakos, "The Consistent Use of Names and Polymorphism in the Definition of Object Petri Nets", in Proceedings of the 17th International Conference on Application and Theory of Petri Nets, Osaka, Japan, June 1996, vol. 1091 of Lecture Notes in Computer Science, , J. Billington and W. Reisig, Eds., Springer-Verlag, pp. 380-399, 1996.

[32] N-Y, Lee, C.R. Litecky, "An empirical study of software reuse with special attention to ada", IEEE Trans SE, vol. 23 no. 9, pp 537-549, September 1997.

[33] Y. Lei and M. P. A. Singh, "A Comparison of Workflow Metamodels", in Proceedings of the ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, 6-7 November 1997, UCLA, Los Angeles, California, S. Liddle, Ed., pp. 1-12, 1997.

[34] J. Li, J.S.K. Ang, X. Tong and M. Tueni, "AMS: A Declarative Formalism for Hierarchical Representation of Procedural Knowledge", IEEE Transactions on Knowledge and Data Engineering, vol. 6, no. 4, pp. 639-643, 1994.

[35] M. McIlroy, "Mass-Produced Software Components", 1968 NATO Conference on Software Engineering, pp. 138-155, 1968.

[36] M. Merz, D. Moldt, K. Müller and W. Lamersdorf, "Workflow Modeling and Execution with Coloured Petri Nets in COSM", In Proceedings of the Workshop on Applications of Petri Nets to Protocols within the 16th International Conference on Application and Theory of Petri Nets, pp. 1-12, 1995.

[37] M. Merz, K. Müller-Jones and W. Lamersdorf, "Petrinetz-basierte Modellierung und Steuerung unternehmensübergreifender Geschäftsprozesse", in Proceedings of the GI/SI Jahrestagung 1995, Tagungsband der GISI 95 Herausforderungen eines globalen Informationsverbundes für die Informatik, F. Huber-Wäschle, H. Schauer and P. Widmayer, Eds., Springer-Verlag, Zürich, pp. 1-8, 18-20 Sept. 1995.

[38] H. Mili, F. Mili, and A. Mili, "Reusing software: issues and research directions", IEEE Transactions on Software Engineering, vol. 21 no. 6, pp. 528-561, 1995.

[39] G. J. Nutt, "The Evolution towards Flexible Workflow Systems", Distributed Systems Engineering, no. 3 4, pp. 276-294, 1996.

[40] A. Oberweis, R. Schätzle, W. Stucky, W. Weitz and G. Zimmermann, "INCOME/WF- A Petri-net Based Approach to Workflow Management", H. Krallmann, Ed. Wirtschaftsinformatik '97, Springer-Verlag, pp. 557-580, 1997.

[41] A. Oberweis, "An Integrated Approach for the Specification of Processes and Related Complex Structured Objects in Business Applications", Decision Support Systems, no. 17, pp. 31-53 ,1996.

[42] R. Prieto-Diaz, "Status Report: Software Reusability", IEEE Software, pp. 61-66, may 1993.

[43] G. Rozenberg, P.S. Thiagarajan, "Petri Nets: Basic Notions, Structure, Behaviour", in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg, Eds., Current Trends in Concurrency, Lecture Notes in Computer Science, vol. 224, Springer-Verlag, 1986.

[44] A.K. Schömig and H. Rau, "A Petri Net Approach for the Performance Analysis of Business Processes", University of Würzburg, Report n° 116 Seminar at IBFI, Schloss Dagstuhl, May 22-26, 1995.

[45] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Addison-Wesley, 1996.

[46] A. Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Languages", in Proceedings of the International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), 1986.

[47] C. Szyperski, "Component software : beyond object-oriented programming", Addison-Wesley, 1997.

[48] D. Wikarski, "An Introduction to Modular Process Nets", International Computer Science Institute (ICSI) Berkeley, Technical Report TR-96-019, CA, USA, 1996.

[49] D. Wikarski, Y. Han and M. Löwe, "Higher-Order Object Nets and Their Application to Workflow modeling", Technische Universität Berlin, Forschungsberichte der FB Informatik 95-34, 1995.

[50] M. Zand, M. Samadzadeh, "Software reuse : current status and trends", Journal of systems and software, vol. 30, pp. 167-170, 1995.

[51] M. D. Zisman, "Representation, Specification and Automation of Office Procedures", University of Pennsylvania Wharton School of Business, PhD Thesis, 1977.

[52] S. H. Zweben, and S. H. Edwards, "The effects of layering and encapsulation on software development cost and quality", IEEE Transactions on Software Engineering, vol. 21, no. 3, pp. 200-208, 1995.

Finding Errors in the Design of a Workflow Process
A Petri-net-based Approach

W.M.P. van der Aalst

Department of Mathematics and Computing Science
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
telephone: -31 40 2474295, e-mail: wsinwawin.tue.nl

Abstract
Workflow management systems facilitate the everyday operation of business processes by taking care of the logistic control of work. In contrast to traditional information systems, they attempt to support frequent changes of the workflows at hand. Therefore, the need for analysis methods to verify the correctness of workflows is becoming more prominent. In this paper we present a method based on Petri nets. This analysis method exploits the structure of the Petri net to find potential errors in the design of the workflow. Moreover, the analysis method allows for the compositional verification of workflows.

Keywords: Petri nets; free-choice Petri nets; workflow management systems; analysis of workflows; business process reengineering; analysis of Petri nets; compositional analysis.

1 Introduction

Workflow management systems (WFMS) are used for the modeling, analysis, enactment, and coordination of structured business processes by groups of people. Business processes supported by a WFMS are *case-driven*, i.e., tasks are executed for specific cases. Approving loans, processing insurance claims, billing, processing tax declarations, handling traffic violations and mortgaging, are typical case-driven processes which are often supported by a WFMS. These case-driven processes, also called *workflows*, are marked by three dimensions: (1) the process dimension, (2) the resource dimension, and (3) the case dimension (see Figure 1). The process dimension is concerned with the partial ordering of tasks. The tasks which need to be executed are identified and the routing of cases along these tasks is determined. Conditional, sequential, parallel and iterative routing are typical structures specified in the process dimension. Tasks are executed by resources. Resources are human (e.g. employee) and/or non-human (e.g. device, software, hardware). In the resource dimension these resources are classified by identifying roles (resource classes based on functional characteristics) and organizational units (groups, teams or departments). Both the process dimension and the resource dimension are generic, i.e., they are not tailored towards a specific case. The third dimension of a workflow is concerned with individual cases which are executed according to the process definition (first dimension) by the proper resources (second dimension).

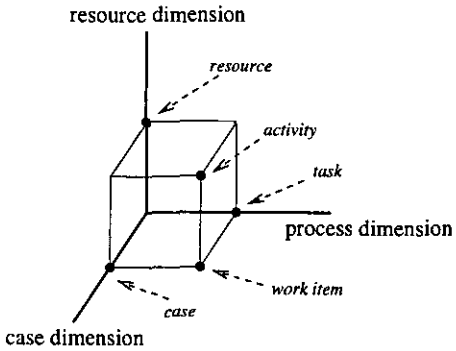


Figure 1: The three dimensions of workflow.

Managing workflows is not a new idea. Workflow control techniques have existed for decades and many management concepts originating from production and logistics are also applicable in a workflow context. However, just recently, commercially available generic WFMS's have become a reality. Although these systems have been applied successfully, contemporary WFMS's have at least two important drawbacks. First of all, today's systems do not scale well, have limited fault tolerance and are inflexible. Secondly, a solid theoretical foundation is missing. Most of the more than 250 commercially available WFMS's use a vendor-specific ad-hoc modeling technique to design workflows. In spite of the efforts of the Workflow Management Coalition ([20]), real standards are missing. The absence of formalized standards hinders the development of tool-independent analysis techniques. As a result, contemporary WFMS's do not facilitate advanced analysis methods to determine the correctness of a workflow.

As many researchers have indicated ([11, 16, 21]), Petri nets constitute a good starting point for a solid theoretical foundation of workflow management. In this paper we focus on the process dimension. We use Petri nets to specify the partial ordering of tasks. Based on a Petri-net-based representation of the workflow process, we tackle the problem of verification. We will provide techniques to verify the so-called *soundness property* introduced in [4]. A workflow process is sound if and only if, for any case, the process terminates properly, i.e., termination is guaranteed, there are no dangling references, and deadlock and livelock are absent.

This paper extends the results presented in [4]. We will show that in most of the situations encountered in practice, the soundness property can be checked in polynomial time. Moreover, we identify suspicious constructs which may endanger the correctness of a workflow process. We will also show that the approach presented in this paper allows for the compositional verification of workflow processes, i.e., the correctness of a process can be decided by partitioning it into sound subprocesses. To support the application of the results presented in this paper, we have developed a Petri-net-based workflow analyzer called Woflan ([5]). Woflan is a workflow management system independent analysis tool which inter-

faces with two of the leading products at the Dutch workflow market.

2 Petri nets

This section introduces the basic Petri net terminology and notations. Readers familiar with Petri nets can skip this section.[1]

The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 1 (Petri net) *A Petri net is a triple* (P, T, F)*:*

- P *is a finite set of places,*
- T *is a finite set of transitions* $(P \cap T = \emptyset)$*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation)*

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing p as an input place. Note that we restrict ourselves to arcs with weight 1. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \rightarrow \mathbf{N}$. We will represent a state as follows: $1p_1+2p_2+1p_3+0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$. To compare states we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

Given a Petri net (P, T, F) and a state M_1, we have the following notations:

[1] Note that states are represented by weighted sums and note the definition of (elementary) (conflict-free) paths.

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*} M_1$.

We use (PN, M) to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M \xrightarrow{*} M'$. Let us define some properties for Petri nets.

Definition 2 (Live) *A Petri net (PN, M) is live iff, for every reachable state M' and every transition t there is a state M'' reachable from M' which enables t.*

Definition 3 (Bounded, safe) *A Petri net (PN, M) is bounded iff, for every reachable state and every place p the number of tokens in p is bounded. The net is safe iff for each place the maximum number of tokens does not exceed 1.*

Definition 4 (Well-formed) *A Petri net PN is well-formed iff there is a state M such that (PN, M) is live and bounded.*

Paths connect nodes by a sequence of arcs.

Definition 5 (Path, Elementary, Conflict-free) *Let PN be a Petri net. A path C from a node n_1 to a node n_k is a sequence $\langle n_1, n_2, \ldots, n_k \rangle$ such that $\langle n_i, n_{i+1} \rangle \in F$ for $1 \leq i \leq k-1$. C is elementary iff, for any two nodes n_i and n_j on C, $i \neq j \Rightarrow n_i \neq n_j$. C is conflict-free iff, for any place n_j on C and any transition n_i on C, $j \neq i-1 \Rightarrow n_j \notin {}^\bullet n_i$.*

For convenience, we introduce the alphabet operator α on paths. If $C = \langle n_1, n_2, \ldots, n_k \rangle$, then $\alpha(C) = \{n_1, n_2, \ldots, n_k\}$.

Definition 6 (Strongly connected) *A Petri net is strongly connected iff, for every pair of nodes (i.e. places and transitions) x and y, there is a path leading from x to y.*

3 WF-nets

In Figure 1 we indicated that a workflow has (at least) three dimensions. The process dimension is the most prominent one, because the core of any workflow system is formed by the processes it supports. In the process dimension building blocks such as the AND-split, AND-join, OR-split, and OR-join are used to model sequential, conditional, parallel and iterative routing (WFMC [20]). Clearly, a Petri net can be used to specify the routing of cases. *Tasks* are modeled by transitions and causal dependencies are modeled by places. In fact, a place corresponds to a *condition* which can be used as pre- and/or post-conditions

for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs. Moreover, in [1, 3] it is shown that the Petri net approach also allows for useful routing constructs absent in many WFMS's.
A Petri net which models the process dimension of a workflow, is called a *WorkFlow net* (WF-net). It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation.

Definition 7 (WF-net) *A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net) if and only if:*

(i) PN has two special places: i and o. Place i is a source place: $\bullet i = \emptyset$. Place o is a sink place: $o\bullet = \emptyset$.

(ii) If we add a transition t^* to PN which connects place o with i (i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.

A WF-net has one input place (i) and one output place (o) because any case handled by the procedure represented by the WF-net is created if it enters the WFMS and is deleted once it is completely handled by the WFMS, i.e., the WF-net specifies the life-cycle of a case. The second requirement in Definition 7 (the Petri net extended with t^* should be strongly connected) states that for each transition t (place p) there should be a path from place i to o via t (p). This requirement has been added to avoid 'dangling tasks and/or conditions', i.e., tasks and conditions which do not contribute to the processing of cases.

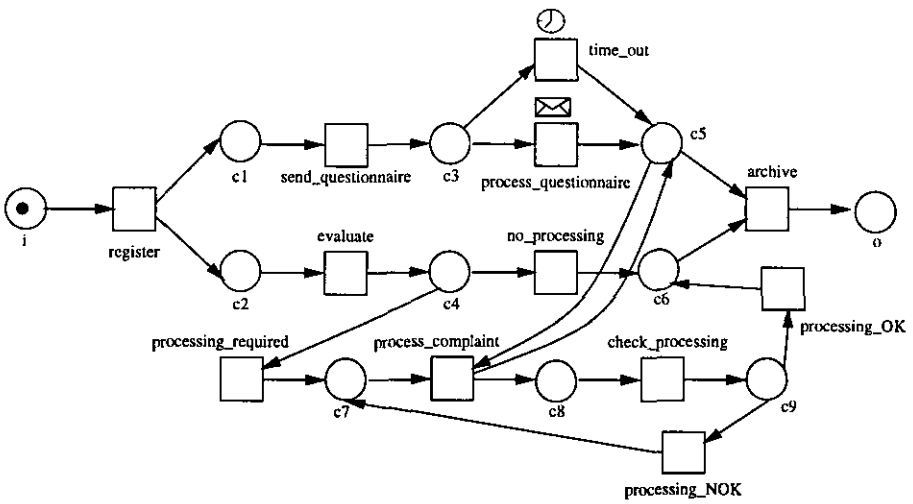


Figure 2: A WF-net for the processing of complaints.

Figure 2 shows a WF-net which models the processing of complaints. First the complaint is registered (task *register*), then in parallel a questionnaire is sent to the complainant (task

send_questionnaire) and the complaint is evaluated (task *evaluate*). If the complainant returns the questionnaire within two weeks, the task *process_questionnaire* is executed. If the questionnaire is not returned within two weeks, the result of the questionnaire is discarded (task *time_out*). Based on the result of the evaluation, the complaint is processed or not. The actual processing of the complaint (task *process_complaint*) is delayed until condition *c5* is satisfied, i.e., the questionnaire is processed or a time-out has occurred. The processing of the complaint is checked via task *check_processing*. Finally, task *archive* is executed. Note that sequential, conditional, parallel and iterative routing are present in this example.

The WF-net shown in Figure 2 clearly illustrates that we focus on the process dimension. We abstract from resources, applications and technical platforms. Moreover, we also abstract from *case variables* and *triggers*. Case variables are used to resolve choices (OR-split), i.e., the choice between *processing_required* and *no_processing* is (partially) based on case variables set during the execution of task *evaluate*. The choice between *processing_OK* and *processing_NOK* is resolved by testing case variables set by *check_processing*. In the WF-net we abstract from case variables by introducing non-deterministic choices in the Petri-net. If we don't abstract from this information, we would have to model the (unknown) behavior of the applications used in each of the tasks and analysis would become intractable. In Figure 2 we have indicated that *time_out* and *process_questionnaire* require triggers. The clock symbol denotes a time trigger and the envelope symbol denotes an external trigger. Task *time_out* requires a time trigger ('two weeks have passed') and *process_questionnaire* requires a message trigger ('the questionnaire has been returned'). A trigger can be seen as an additional condition which needs to be satisfied. In the remainder of this paper we abstract from these trigger conditions. We assume that the environment behaves fairly, i.e., the liveness of a transition is not hindered by the continuous absence of a specific trigger. As a result, every trigger condition will be satisfied eventually (if needed).

4 Soundness

In this section we summarize some of the basic results for WF-nets presented in [4]. The remainder of this paper will build on these results.
The two requirements stated in Definition 7 can be verified statically, i.e., they only relate to the structure of the Petri net. However, there is another requirement which should be satisfied:

> *For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there should be no dead tasks, i.e., it should be possible to execute an arbitrary task by following the appropriate route though the WF-net. These two additional requirements correspond to the so-called *soundness property.*

Definition 8 (Sound) *A procedure modeled by a WF-net* $PN = (P, T, F)$ *is sound if and only if:*

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state o. Formally:[2]

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

(ii) State o is the only state reachable from state i with at least one token in place o. Formally:

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in (PN, i). Formally:

$$\forall_{t \in T} \exists_{M,M'} \; i \xrightarrow{*} M \xrightarrow{t} M'$$

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 8 states that starting from the initial state (state *i*), it is always possible to reach the state with one token in place *o* (state *o*). If we assume fairness (i.e. a transition that is enabled infinitely often will fire eventually), then the first requirement implies that eventually state *o* will be reached. The fairness assumption is reasonable in the context of workflow management; all choices are made (implicitly en explicitly) by applications, humans or external actors. Clearly, they should not introduce an infinite loop. The second requirement states that the moment a token is put in place *o*, all the other places should be empty. Sometimes the term *proper termination* is used to describe the first two requirements [14]. The last requirement states that there are no dead transitions (tasks) in the initial state *i*.

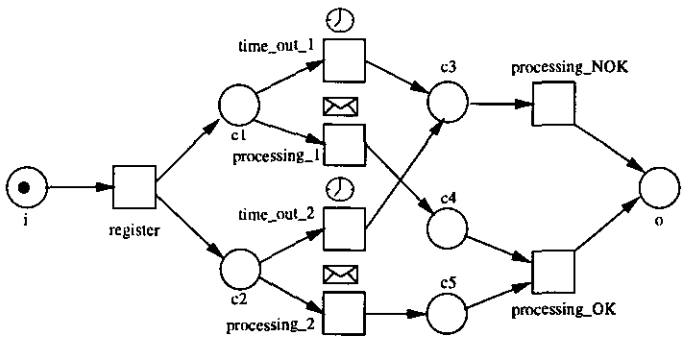


Figure 3: Another WF-net for the processing of complaints.

Figure 3 shows a WF-net which is not sound. There are several deficiencies. If *time_out_1* and *processing_2* fire or *time_out_2* and *processing_1* fire, the WF-net will not terminate properly because a token gets stuck in *c4* or *c5*. If *time_out_1* and *time_out_2* fire, then the task *processing_NOK* will be executed twice and because of the presence of two tokens in *o* the moment of termination is not clear.

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. In [4] we

[2]Note that there is an overloading of notation: the symbol *i* is used to denote both the *place i* and the *state* with only one token in place *i* (see Section 2).

have shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net obtained by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{(o, t^*), (t^*, i)\}$. The extended net allows for the formulation of the following theorem.

Theorem 1 *A WF-net PN is sound if and only if $(\overline{PN}, i)$ is live and bounded.*

Proof.
See [4] or [2]. □

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness.

5 Structural characterization of soundness

Theorem 1 gives a useful characterization of the quality of a workflow process definition. However, there are a number of problems:

- For a complex WF-net it may be intractable to decide soundness. (For arbitrary WF-nets liveness and boundedness are decidable but also EXPSPACE-hard, cf. Cheng, Esparza and Palsberg [8].)
- Soundness is a minimal requirement. Readability and maintainability issues are not addressed by Theorem 1.
- Theorem 1 does not show how a non-sound WF-net should be modified, i.e., it does not identify constructs which invalidate the soundness property.

These problems stem from the fact that the definition of soundness relates to the dynamics of a WF-net while the workflow designer is concerned with the static structure of the WF-net. Therefore, it is interesting to investigate structural characterizations of sound WF-nets. For this purpose we introduce three interesting subclasses of WF-nets: free-choice WF-nets, well-structured WF-nets, and S-coverable WF-nets.

5.1 Free-choice WF-nets

Most of the WFMS's available at the moment, abstract from states between tasks, i.e., states are not represented explicitly. These WFMS's use building blocks such as the AND-split, AND-join, OR-split and OR-join to specify workflow procedures. The AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. Because these systems abstract from states, every choice is made *inside* an OR-split building block. If we model an OR-split in terms of a Petri net, the OR-split corresponds to a number of transitions sharing the same set of input places. This means that for these WFMS's, a workflow procedure corresponds to a *free-choice Petri net*.

Definition 9 (Free-choice) *A Petri net is a free-choice Petri net iff, for every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.*

It is easy to see that a process definition composed of AND-splits, AND-joins, OR-splits and OR-joins is free-choice. If two transitions t_1 and t_2 share an input place ($\bullet t_1 \cap \bullet t_2 \neq \emptyset$), then they are part of an OR-split, i.e., a 'free choice' between a number of alternatives. Therefore, the sets of input places of t_1 and t_2 should match ($\bullet t_1 = \bullet t_2$). Figure 3 shows a free-choice WF-net. The WF-net shown in Figure 2 is not free-choice; *archive* and *process_complaint* share an input place but the two corresponding input sets differ.

We have evaluated many WFMS's and just one of these systems (COSA [18]) allows for a construction which is comparable to a non-free choice WF-net. Therefore, it makes sense to consider free-choice Petri nets. Clearly, parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property. Another reason for restricting WF-nets to free-choice Petri nets is the following. If we allow non-free-choice Petri nets, then the choice between conflicting tasks *may* be influenced by the order in which the preceding tasks are executed. The routing of a case should be independent of the order in which tasks are executed. A situation where the free-choice property is violated is often a mixture of parallelism and choice. Figure 4 shows such a situation. Firing transition *t1* introduces parallelism. Although there is no real choice between *t2* and *t5* (*t5* is not enabled), the parallel execution of *t2* and *t3* results in a situation where *t5* is not allowed to occur. However, if the execution of *t2* is delayed until *t3* has been executed, then there is a real choice between *t2* and *t5*. In our opinion parallelism itself should be separated from the choice between two or more alternatives. Therefore, we consider the non-free-choice construct shown in Figure 4 to be improper. In literature, the term *confusion* is often used to refer to the situation shown in Figure 4.

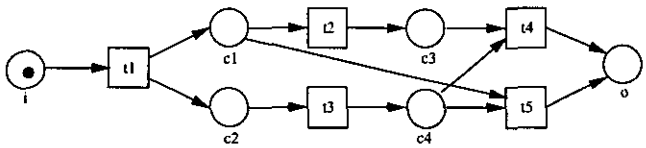


Figure 4: A non-free-choice WF-net containing a mixture of parallelism and choice.

Free-choice Petri nets have been studied extensively (cf. Best [7], Desel and Esparza [10, 9, 12], Hack [15]) because they seem to be a good compromise between expressive power and analyzability. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist. For example, the well-known Rank Theorem (Desel and Esparza [10]) enables us to formulate the following corollary.

Corollary 1 *The following problem can be solved in polynomial time.*
Given a free-choice WF-net, to decide if it is sound.

Proof.
Let PN be a free-choice WF-net. The extended net $\overline{PN}$ is also free-choice. Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial

time (Rank Theorem [10]). By Theorem 1, this corresponds to soundness. □

Corollary 1 shows that, for free-choice nets, there are efficient algorithms to decide soundness. Moreover, a sound free-choice WF-net is guaranteed to be safe.

Lemma 1 *A sound free-choice WF-net is safe.*

Proof.
Let PN be a sound free-choice WF-net. $\overline{PN}$ is the Petri net PN extended with a transition connecting o and i. $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is covered by state-machines (S-components, cf. [10]), i.e., each place is part of such a state-machine component. Clearly, i and o are nodes of any state-machine component. Hence, for each place p there is a semi-positive invariant with weights 0 or 1 which assigns a positive weight to p, i and o. Therefore, $\overline{PN}$ is safe and so is PN. □

Safeness is a desirable property, because it makes no sense to have multiple tokens in a place representing a condition. A condition is either true (1 token) or false (no tokens).

Although most WFMS's only allow for free-choice workflows, free-choice WF-nets are not a completely satisfactory structural characterization of 'good' workflows. On the one hand, there are non-free-choice WF-nets which correspond to sensible workflows (cf. Figure 2). On the other hand there are sound free-choice WF-nets which make no sense. Nevertheless, the free-choice property is a desirable property. If a workflow can be modeled as a free-choice WF-net, one should do so. A workflow specification based on a free-choice WF-net can be enacted by most workflow systems. Moreover, a free-choice WF-net allows for efficient analysis techniques and is easier to understand. Non-free-choice constructs such as the construct shown in Figure 4 are a potential source of anomalous behavior (e.g. deadlock) which is difficult to trace.

5.2 Well-structured WF-nets

Another approach to obtain a structural characterization of 'good' workflows, is to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. As shown in Figure 5, an AND-split should be complemented by an AND-join and an OR-split should be complemented by an OR-join.

One of the deficiencies of the WF-net shown in Figure 3 is the fact that the AND-split *register* is complemented by the OR-join *c3* or the OR-join *o*. To formalize the concept illustrated in Figure 5 we give the following definition.

Definition 10 (Well-handled) *A Petri net PN is well-handled iff, for any pair of nodes x and y such that one of the nodes is a place and the other a transition and for any pair of elementary paths C_1 and C_2 leading from x to y, $\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$.*

Note that the WF-net shown in Figure 3 is not well-handled. A Petri net which is well-handled has a number of nice properties, e.g. strong connectedness and well-formedness

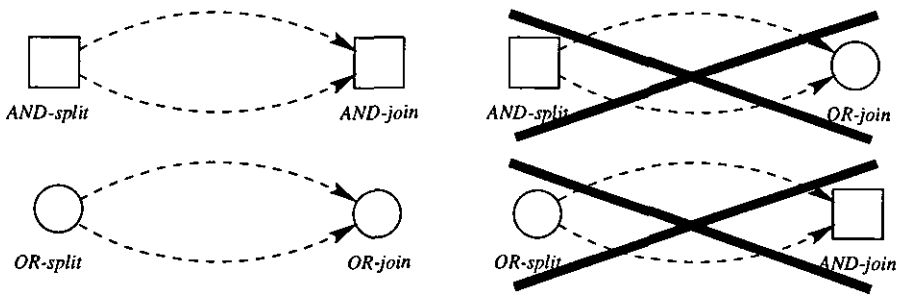


Figure 5: Good and bad constructions.

coincide.

Lemma 2 *A strongly connected well-handled Petri net is well-formed.*

Proof.
Let PN be a strongly connected well-handled Petri net. Clearly, there are no circuits that have PT-handles nor TP-handles ([13]). Therefore, the net is structurally bounded (See Theorem 3.1 in [13]) and structurally live (See Theorem 3.2 in [13]). Hence, PN is well-formed. □

Clearly, well-handledness is a desirable property for any WF-net PN. Moreover, we also require the extended $\overline{PN}$ to be well-handled. We impose this additional requirement for the following reason. Suppose we want to use PN as a part of a larger WF-net PN'. PN' is the original WF-net extended with an 'undo-task'. See Figure 6. Transition $undo$ corresponds to the undo-task, transitions $t1$ and $t2$ have been added to make PN' a WF-net. It is undesirable that transition $undo$ violates the well-handledness property of the original net. However, PN' is well-handled iff $\overline{PN}$ is well-handled. Therefore, we require $\overline{PN}$ to be well-handled. We use the term *well-structured* to refer to WF-nets whose extension is well-handled.

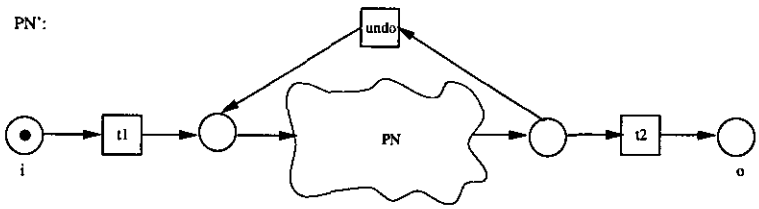


Figure 6: The WF-net PN' is well-handled iff $\overline{PN}$ is well-handled.

Definition 11 (Well-structured) *A WF-net PN is well-structured iff $\overline{PN}$ is well-handled.*

Well-structured WF-nets have a number of desirable properties. Soundness can be verified in polynomial time and a sound well-structured WF-net is safe. To prove these properties we use some of the results obtained for *elementary extended non-self controlling nets*.

Definition 12 (Elementary extended non-self controlling) *A Petri net PN is elementary extended non-self controlling (ENSC) iff, for every pair of transitions t_1 and t_2 such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, there does not exist an elementary path C leading from t_1 to t_2 such that $\bullet t_1 \cap \alpha(C) = \emptyset$.*

Theorem 2 *Let PN be a WF-net. If PN is well-structured, then $\overline{PN}$ is elementary extended non-self controlling.*

Proof.
Assume that $\overline{PN}$ is not elementary extended non-self controlling. This means that there is a pair of transitions t_1 and t_k such that $\bullet t_1 \cap \bullet t_k \neq \emptyset$ and there exist an elementary path $C = \langle t_1, p_2, t_2, \ldots, p_k, t_k \rangle$ leading from t_1 to t_k and $\bullet t_1 \cap \alpha(C) = \emptyset$. Let $p_1 \in \bullet t_1 \cap \bullet t_k$. $C_1 = \langle p_1, t_k \rangle$ and $C_2 = \langle p_1, t_1, p_2, t_2, \ldots, p_k, t_k \rangle$ are paths leading from p_1 to t_k. (Note that C_2 is the concatenation of $\langle p_1 \rangle$ and C.) Clearly, C_1 is elementary. We will also show that C_2 is elementary. C is elementary, and $p_1 \notin \alpha(C)$ because $p_1 \in \bullet t_1$. Hence, C_2 is also elementary. Since C_1 and C_2 are both elementary paths, $C_1 \neq C_2$ and $\alpha(C_1) \cap \alpha(C_2) = \{p_1, t_k\}$, we conclude that $\overline{PN}$ is not well-handled. □

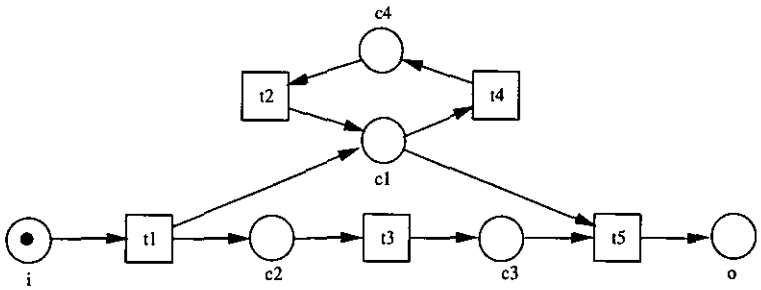


Figure 7: A well-structured WF-net.

Consider for example the WF-net shown in Figure 7. The WF-net is well-structured and, therefore, also elementary extended non-self controlling. However, the net is not free-choice. Nevertheless, it is possible to verify soundness for such a WF-net very efficiently.

Corollary 2 *The following problem can be solved in polynomial time.*
Given a well-structured WF-net, to decide if it is sound.

Proof.
Let PN be a well-structured WF-net. The extended net $\overline{PN}$ is elementary extended non-self controlling (Theorem 2) and structurally bounded (see proof of Lemma 2). For bounded elementary extended non-self controlling nets the problem of deciding whether a given marking is live, can be solved in polynomial time (See [6]). Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time. By Theorem 1, this corresponds to soundness. □

Lemma 3 *A sound well-structured WF-net is safe.*

Proof.
Let $\overline{PN}$ be the net PN extended with a transition connecting o and i. $\overline{PN}$ is extended non-self controlling. $\overline{PN}$ is covered by state-machines (S-components), see Corollary 5.3 in [6]. Hence, $\overline{PN}$ is safe and so is PN (see proof of Lemma 1). □

Well-structured WF-nets and free-choice WF-nets have similar properties. In both cases soundness can be verified very efficiently and soundness implies safeness. In spite of these similarities, there are sound well-structured WF-nets which are not free-choice (Figure 7) and there are sound free-choice WF-nets which are not well-structured. In fact, it is possible to have a sound WF-net which is neither free-choice nor well-structured (Figures 2 and 4).

5.3 S-coverable WF-nets

What about the sound WF-nets shown in Figure 2 and Figure 4? The WF-net shown in Figure 4 can be transformed into a free-choice well-structured WF-net by separating choice and parallelism. The WF-net shown in Figure 2 cannot be transformed into a free-choice or well-structured WF-net without yielding a much more complex WF-net. Place *c5* acts as some kind of milestone which is tested by the task *process_complaint*. Traditional workflow management systems which do not make the state of the case explicit, are not able to handle the workflow specified by Figure 2. Only workflow management systems such as COSA ([18]) have the capability to enact such a state-based workflow. Nevertheless, it is interesting to consider generalizations of free-choice and well-structured WF-nets: *S-coverable WF-nets* can be seen as such a generalization.

Definition 13 (S-coverable) *A WF-net PN is S-coverable iff the extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ satisfies the following property. For each place p there is subnet $PN_s = (P_s, T_s, F_s)$ such that: $p \in P_s$, $P_s \subseteq \overline{P}$, $T_s \subseteq \overline{T}$, $F_s \subseteq \overline{F}$, PN_s is strongly connected, PN_s is a state machine (i.e. each transition in PN_s has one input and one output place), and for every $q \in P_s$ and $t \in \overline{T}$: $(q,t) \in \overline{F} \Rightarrow (q,t) \in F_s$ and $(t,q) \in \overline{F} \Rightarrow (t,q) \in F_s$.*

This definition corresponds to the definition given in [10]. A subnet PN_s which satisfies the requirements stated in Definition 13 is called an *S-component*. PN_s is a strongly connected state machine such that for every place q: if q is an input (output) place of a transition t in $\overline{PN}$, then q is also an input (output) place of t in PN_s.
The WF-nets shown in Figure 2 and Figure 4 are S-coverable. The WF-net shown in Figure 3 is not S-coverable. The following two corollaries show that S-coverability is a generalization of the free-choice property and well-structuredness.

Corollary 3 *A sound free-choice WF-net is S-coverable.*

Proof.
The extended net $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable (cf. [10]).
□

Corollary 4 *A sound well-structured WF-net is S-coverable.*

Proof.
$\overline{PN}$ is extended non-self controlling (Theorem 2). Hence, $\overline{PN}$ is S-coverable (cf. Corollary 5.3 in [6]). □

All the sound WF-nets presented in this paper are S-coverable. Every S-coverable WF-net is safe. The only WF-net which is not sound, i.e. the WF-net shown in Figure 3, is not S-coverable. These and other examples indicate that there is a high correlation between S-coverability and soundness. It seems that S-coverability is one of the basic requirements any workflow process definition should satisfy. From a formal point of view, it is possible to construct WF-nets which are sound but not S-coverable. Typically, these nets contain places which do not restrict the firing of a transition, but which are not in any S-component. (See for example Figure 65 in [17].) From a practical point of view, these WF-nets are to be avoided. WF-nets which are not S-coverable are difficult to interpret because the structural and dynamical properties do not match. For example, these nets can be live and bounded but not structurally bounded. There is no practical need for using constructs which violate the S-coverability property. Therefore, we consider S-coverability to be a basic requirement any WF-net should satisfy.

S-coverability can be verified in polynomial time. Unfortunately, in general it is not possible to verify soundness of an S-coverable WF-net in polynomial time. The problem of deciding soundness for an S-coverable WF-net is PSPACE-complete. For most applications this is not a real problem. In most cases the number of tasks in one workflow process definition is less than 100 and the number of states is less than 200.000. Tools using standard techniques such as the construction of the coverability graph have no problems in coping with these workflow process definitions.

The three structural characterizations (free-choice, well-structured and S-coverable) turn out to be very useful for the analysis of workflow process definitions. S-coverability is a desirable property any workflow definition should satisfy. Constructs violating S-coverability can be detected easily and tools can be build to help the designer to construct an S-coverable WF-net. S-coverability is a generalization of well-structuredness and the free-choice property (Corollary 3 and 4). Both well-structuredness and the free-choice property also correspond to desirable properties of a workflow. A WF-net satisfying at least one one of these two properties can be analyzed very efficiently. However, we have shown that there are workflows that are not free-choice and not well-structured. Consider for example Figure 2. The fact that task *process_complaint* tests whether there is a token in *c5*, prevents the WF-net from being free-choice or well-structured. Although this is a very sensible workflow, most workflow management systems do not support such an advanced routing construct. Even if one is able to use state-based workflows (e.g. COSA) allowing for constructs which violate well-structuredness and the free-choice property, then the structural characterizations are still useful. If a WF-net is not free-choice or not well-structured, one should locate the source which violates one of these properties and check whether it is really necessary to use a non-free-choice or a non-well-structured construct. If the non-free-choice or non-well-structured construct is really necessary, then the correctness of the construct should be double-checked, because it is a potential source of errors.

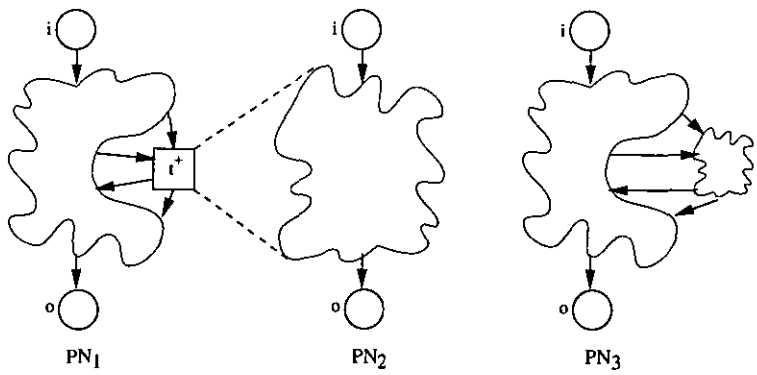


Figure 8: Task refinement: WF-net PN_3 is composed of PN_1 and PN_2.

6 Composition of WF-nets

The WF-nets in this paper are very simple compared to the workflows encountered in practise. For example, in the Dutch Customs Department there are workflows consisting of more than 80 tasks with a very complex interaction structure (cf. [3]). For the designer of such a workflow the complexity is overwhelming and communication with end-users using one huge diagram is difficult. In most cases hierarchical (de)composition is used to tackle this problem. A complex workflow is decomposed into subflows and each of the subflows is decomposed into smaller subflows until the desired level of detail is reached. Many WFMS's allow for such a hierarchical decomposition. In addition, this mechanism can be utilized for the reuse of existing workflows. Consider for example multiple workflows sharing a generic subflow. Some WFMS-vendors also supply reference models which correspond to typical workflow processes in insurance, banking, finance, marketing, purchase, procurement, logistics and manufacturing.

Reference models, reuse and the structuring of complex workflows require a hierarchy concept. The most common hierarchy concept supported by many WFMS's is *task refinement*, i.e., a task can be refined into a subflow. This concept is illustrated in Figure 8. The WF-net PN_1 contains a task t^+ which is refined by another WF-net PN_2, i.e., t^+ is no longer a task but a reference to a subflow. A WF-net which represents a subflow should satisfy the same requirements as an ordinary WF-net (see Definition 7). The semantics of the hierarchy concept are straightforward; simply replace the refined transition by the corresponding subnet. Figure 8 shows that the refinement of t^+ in PN_1 by PN_2 yields a WF-net PN_3.

The hierarchy concept can be exploited to establish the correctness of a workflow. Given a complex hierarchical workflow model, it is possible to verify soundness by analyzing each of the subflows separately. The following theorem shows that the soundness property defined in this paper allows for modular analysis.

Theorem 3 (Compositionality) *Let* $PN_1 = (P_1, T_1, F_1)$ *and* $PN_2 = (P_2, T_2, F_2)$ *be*

two WF-nets such that $T_1 \cap T_2 = \emptyset$, $P_1 \cap P_2 = \{i, o\}$ *and* $t^+ \in T_1$. $PN_3 = (P_3, T_3, F_3)$ *is the WF-net obtained by replacing transition* t^+ *in* PN_1 *by* PN_2, *i.e.,* $P_3 = P_1 \cup P_2$, $T_3 = (T_1 \setminus \{t^+\}) \cup T_2$ *and*

$$\begin{aligned} F_3 \;=\; & \{(x, y) \in F_1 \mid x \neq t^+ \wedge y \neq t^+\} \cup \{(x, y) \in F_2 \mid \{x, y\} \cap \{i, o\} = \emptyset\} \cup \\ & \{(x, y) \in P_1 \times T_2 \mid (x, t^+) \in F_1 \wedge (i, y) \in F_2\} \cup \\ & \{(x, y) \in T_2 \times P_1 \mid (t^+, y) \in F_1 \wedge (x, o) \in F_2\}. \end{aligned}$$

For PN_1, PN_2 *and* PN_3 *the following statements hold:*

1. *If* PN_3 *is free-choice, then* PN_1 *and* PN_2 *are free-choice.*
2. *If* PN_3 *is well-structured, then* PN_1 *and* PN_2 *are well-structured.*
3. *If* (PN_1, i) *is safe and* PN_1 *and* PN_2 *are sound, then* PN_3 *is sound.*
4. (PN_1, i) *and* (PN_2, i) *are safe and sound iff* (PN_3, i) *is safe and sound.*
5. PN_1 *and* PN_2 *are free-choice and sound iff* PN_3 *is free-choice and sound.*
6. *If* PN_3 *is well-structured and sound, then* PN_1 *and* PN_2 *are well-structured and sound.*
7. *If* $\bullet t^+$ *and* $t^+\bullet$ *are both singletons, then* PN_1 *and* PN_2 *are well-structured and sound iff* PN_3 *is well-structured and sound.*

Proof.

1. The only transitions that may violate the free-choice property are t^+ (in PN_1) and $\{t \in T_2 \mid (i, t) \in F_2\}$ (in PN_2). Transition t^+ has the same input set as any of the transitions $\{t \in T_2 \mid (i, t) \in F_2\}$ in PN_3 if we only consider the places in $P_3 \cap P_1$. Hence, t^+ does not violate the free-choice property in PN_1. All transitions t in PN_2 such that $(i, t) \in F_2$ respect the free-choice property; the input places in $P_3 \setminus P_2$ are replaced by i.
2. $\overline{PN_1}$ ($\overline{PN_2}$) is well-handled because any elementary path in $\overline{PN_1}$ ($\overline{PN_2}$) corresponds to a path in $\overline{PN_3}$.
3. Let (PN_1, i) be safe and let PN_1 and PN_2 be sound. We need to prove that $(\overline{PN_3}, i)$ is live and bounded. The subnet in $\overline{PN_3}$ which corresponds to t^+ behaves like a transition which may postpone the production of tokens for $t^+\bullet$. It is essential that the input places of t^+ in $(\overline{PN_3}, i)$ are safe. This way it is guaranteed that the states of the subnet correspond to the states of $(\overline{PN_2}, i)$. Hence, the transitions in $T_3 \cap T_2$ are live (t^+ is live) and the places in $P_3 \setminus P_1$ are bounded. Since the subnet behaves like t^+, the transitions in $T_3 \cap (T_1 \setminus \{t^+\})$ are live and the places in $P_3 \cap P_1$ are bounded. Hence, PN_3 is sound.
4. Let (PN_1, i) and (PN_2, i) be safe and sound. Clearly, PN_3 is sound (see proof of 3.). (PN_3, i) is also safe because every reachable state corresponds to a combination of a safe state of (PN_1, i) and a safe state of (PN_2, i).

Let (PN_3, i) be safe and sound. Consider the subnet in PN_3 which corresponds to t^+. X is the set of transitions in $T_3 \cap T_2$ consuming from $\bullet t^+$ and Y is the set of transitions in $T_3 \cap T_2$ producing tokens for $t^+\bullet$. If a transition in X fires, then it should be possible to fire a transition in Y because of the liveness of the original net. If a transition in Y fires, the subnet should become empty. If the subnet is not empty after firing a transition in Y, then there are two possibilities: (1) it is possible to move the subnet to a state such that a transition in Y can fire (without firing transitions in $T_3 \cap T_1$) or (2) it is not possible to move to such a state. In the first case, the places $t^+\bullet$ in PN_3 are not safe. In the second case, a token is trapped in the subnet or the subnet is not safe the moment a transition in X fires. (PN_2, i) corresponds to the subnet bordered by X and Y and is, as we have just shown, sound and safe. It remains to prove that (PN_1, i) is safe and sound. Since the subnet which corresponds to t^+ behaves like a transition which may postpone the production of tokens, we can replace the subnet by t^+ without changing dynamic properties such as safeness and soundness.

5. Let PN_1 and PN_2 be free-choice and sound. Since $(\overline{PN_1}, i)$ is safe (see Lemma 1), PN_3 is sound (see proof of 3.). It remains to prove that PN_3 is free-choice. The only transitions in PN_3 which may violate the free-choice property are the transitions in $T_3 \cap T_2$ consuming tokens from $\bullet t^+$. Because PN_2 is sound, these transitions need to have an input set identical to t^+ in PN_1 (if this is not the case at least one of the transitions is dead). Since PN_1 is free-choice, PN_3 is also free-choice.
   Let PN_3 be free-choice and sound. PN_1 and PN_2 are also free-choice (see proof of 1.). Since (PN_3, i) is safe (see Lemma 1), PN_1 and PN_2 are sound (see proof of 4.).

6. Let PN_3 be well-structured and sound. PN_1 and PN_2 are also well-structured (see proof of 2.). Since (PN_3, i) is safe (see Lemma 3), PN_1 and PN_2 are sound (see proof of 4.).

7. It remains to prove that if PN_1 and PN_2 are well-structured, then PN_3 is also well-structured. Suppose that PN_3 is not well-structured. There are two disjunct elementary paths leading from x to y in $\overline{PN_3}$. Since PN_1 is well-structured, at least one of these paths is enabled via the refinement of t^+. However, because t^+ has precisely one input and one output place and PN_2 is also well-structured, this is not possible.

□

Theorem 3 is a generalization of Theorem 3 in [19]. It extends the concept of a block with multiple entry and exit transitions and gives stronger results for specific subclasses.

Figure 9 shows a hierarchical WF-net. Both of the subflows (*handle_questionnaire* and *processing*) and the main flow are safe and sound. Therefore, the overall workflow represented by the hierarchical WF-net is also safe and sound. Moreover, the free-choice property and well-structuredness are also preserved by the hierarchical composition. Theorem 3 is of particular importance for the reuse of subflows. For the analysis of a complex

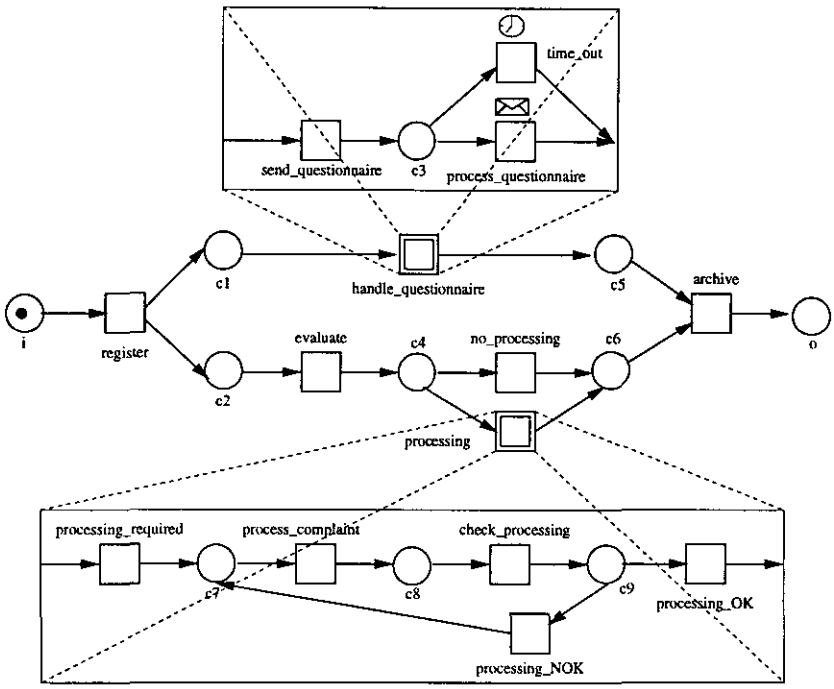


Figure 9: A hierarchical WF-net for the processing of complaints.

workflow, every safe and sound subflow can be considered to be a single task. This allows for an efficient modular analysis of the soundness property. Moreover, the statements embedded in Theorem 3 can help a workflow designer to construct correct workflow process definitions.

7 Woflan

To allow users of today's workflow management systems to benefit from the results presented in this paper we have developed Woflan, a tool which analyzes workflow process definitions specified in terms of Petri nets. Woflan (WOrkFLow ANalyzer) has been designed to verify process definitions which are downloaded from a workflow management system ([5]). Clearly, there is a need for such a verification tool, because today's workflow management systems do not support advanced techniques to verify the correctness of workflow process definitions. These systems typically restrict themselves to a number of (trivial) syntactical checks. Therefore, serious errors such as deadlocks and livelocks may remain undetected. This means that an erroneous workflow may go into production, thus causing dramatic problems for the organization. An erroneous workflow may lead to extra work, legal problems, angry customers, managerial problems, and ill-motivated employees. Therefore, it is important to verify the correctness of a workflow process definition before it becomes operational.

At the moment there are two workflow tools which can interface with Woflan: COSA (COSA Solutions/Software-Ley, Pullheim, Germany) and Protos (Pallas Athena, Plasmolen, The Netherlands). COSA (COSA Solutions) is one of the leading products in the Dutch workflow market. COSA allows for the modeling and enactment of complex workflow processes which use advanced routing constructs. However, COSA does not support verification. Fortunately, Woflan can analyze any workflow process definition constructed by using CONE (COSA Network Editor), the design tool of the COSA system. Woflan can also import process definitions made with Protos. Protos (Pallas Athena) is a so-called BPR-tool. Protos supports Business Process Reengineering (BPR) efforts and can be used to model and analyze business processes. The tool is very easy to use and is based on Petri nets. To facilitate the modeling of simple workflows by users not familiar with Petri nets, it is possible to abstract from states. However, Protos cannot detect subtle design flaws which may result in deadlocks or livelocks. Therefore, it is useful to download workflows specified with Protos and analyze them with Woflan.

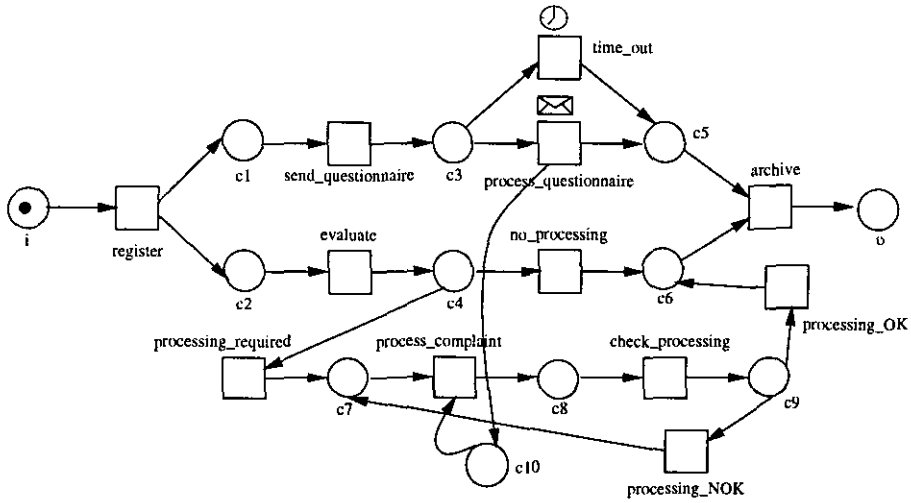


Figure 10: An alternative WF-net for the processing of complaints.

If the workflow process definition is not sound, Woflan guides the user in finding and correcting the error. Since a detailed description of the functionality of Woflan is beyond the scope of this paper, we will use the example shown in Figure 10 to illustrate the features of Woflan. For this particular workflow net, Woflan gives the following diagnostics:

- Woflan points out the fact that place *c10* is not bounded in the net extended with transition t^* which connects the output place *ready* with the input place *start*. This means that it is possible to terminate and leave a token in *c10* (i.e. a dangling reference).
- The OR-split *c3* is complemented by the AND-join *archive*, i.e., there are two disjunct paths (one via *c10*) leading from place *c3* to transition *archive*. Such a construct may lead to a potential deadlock. In this case it does!

- Woflan reports that the workflow net is not covered by state machines (S-components) i.e., the net is not S-coverable. In fact, Woflan indicates that *c10* is the only place not in any S-component.
- The fact that something is wrong with *c10* is also highlighted by the fact that place *c10* is not in the support of any of the semi-positive place invariants generated by Woflan.

The above diagnostics clearly show that the optional synchronization of the two parallel flows via place *c10* is the source of the error. Removing *c10* or replacing *c10* by the construct shown in Figure 2 solves this problem and results in a sound workflow process definition. For a small workflow with only 8 tasks these results may seem trivial. However, workflows encountered in practice may have up to a 100 tasks. Experience shows that for workflows with more than 20 tasks it is not easy to locate the source of the error if the workflow net is not sound. Therefore, the support offered by Woflan is of the utmost importance for the verification of workflow process definitions.

To assist the user in repairing the error, Woflan offers an on-line help facility. The on-line help is based on a step-wise approach to locate and remove constructs which violate the soundness property. This enables users without a background in Petri nets to operate the tool and repair an erroneous workflow process definition.

8 Conclusion

In this paper we have investigated a basic property that any workflow process definition should satisfy: the soundness property. For WF-nets, this property coincides with liveness and boundedness. In our quest for a structural characterization of WF-nets satisfying the soundness property, we have identified three important subclasses: free-choice, well-structured, and S-coverable WF-nets. The identification of these subclasses is useful for the detection of design errors.
If a workflow process is specified by a hierarchical WF-net, then modular analysis of the soundness property is often possible. A workflow composed of correct subflows can be verified without incorporating the specification of each subflow.
The results presented in this paper give workflow designers a handle to construct correct workflows. Although it is possible to use standard Petri-net-based analysis tools, we have developed a workflow analyzer which can be used by people not familiar with Petri-net theory. This workflow analyzer interfaces with existing workflow products such as COSA and Protos.

Acknowledgements

The author would like to thank Dr. M. Voorhoeve and Ir. T. Basten for their valuable suggestions and all the other people involved in the development of Woflan, in particular Ir. E. Verbeek and Dr. D. Hauschildt.

References

[1] W.M.P. van der Aalst. Petri-net-based Workflow Management Software. In A. Sheth, editor, *Proceedings of the NFS Workshop on Workflow and Process Automation in Information Systems*, pages 114–118, Athens, Georgia, May 1996.

[2] W.M.P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996.

[3] W.M.P. van der Aalst. Three Good reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, Nov 1996.

[4] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.

[5] W.M.P. van der Aalst, D. Hauschildt, and H.M.W. Verbeek. A Petri-net-based Tool to Analyze Workflows. In B. Farwer, D. Moldt, and M.O. Stehr, editors, *Proceedings of Petri Nets in System Engineering (PNSE'97)*, pages 78–90, Hamburg, Sept 1997. University of Hamburg (FBI-HH-B-205/97).

[6] K. Barkaoui, J.M. Couvreur, and C. Dutheillet. On liveness in Extended Non Self-Controlling Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag, Berlin, 1995.

[7] E. Best. Structure theory of Petri nets: the free choice hiatus. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 168–206. Springer-Verlag, Berlin, 1987.

[8] A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. In R.K. Shyamasundar, editor, *Foundations of software technology and theoretical computer science*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, Berlin, 1993.

[9] J. Desel. A proof of the Rank theorem for extended free-choice nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 134–153. Springer-Verlag, Berlin, 1992.

[10] J. Desel and J. Esparza. *Free choice Petri nets*, volume 40 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1995.

[11] C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993.

[12] J. Esparza. Synthesis rules for Petri nets, and how they can lead to new results. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 182–198. Springer-Verlag, Berlin, 1990.

[13] J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer-Verlag, Berlin, 1990.

[14] K. Gostellow, V. Cerf, G. Estrin, and S. Volansky. Proper Termination of Flow-of-control in Programs Involving Concurrent Processes. *ACM Sigplan*, 7(11):15–27, 1972.

[15] M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.

[16] G. De Michelis, C. Ellis, and G. Memmi, editors. *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, Zaragoza, Spain, June 1994.

[17] W. Reisig. *Petri nets: an introduction*, volume 4 of *Monographs in theoretical computer science: an EATCS series*. Springer-Verlag, Berlin, 1985.

[18] Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1996.

[19] R. Valette. Analysis of Petri Nets by Stepwise Refinements. *Journal of Computer and System Sciences*, 18:35–46, 1979.

[20] WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.

[21] M. Wolf and U. Reimer, editors. *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow*, Basel, Switzerland, Oct 1996.

Structural Analysis of Workflow Nets with Shared Resources

Kamel Barkaoui[1] and Laure Petrucci[2]

[1] CEDRIC-CNAM
292 rue St-Martin
F-75141 PARIS Cedex 03
barkaoui@cnam.fr
[2] CEDRIC-IIE
18, allée Jean Rostand
F-91025 EVRY Cedex
petrucci@iie.cnam.fr

Abstract. A *workflow* is the automation of *business processes* which describes activities in a business context. A *workflow management system* defines, creates and manages the execution of workflows. Petri nets have been shown to be a well-suited formalism to model and analyse business processes. The verification of *soundness* of a procedure, i.e. correct termination, using Petri nets, was considered in [vdA97]. In this paper, we extend this work by also taking into account resources shared by workflow procedures. We show how the soundness property can be proved efficiently by using structural Petri net techniques. These allow us to obtain parameterized results. Moreover, it leads to a tailoring of the system in order to enhance its performances.

1 Introduction

Workflow Management and *Business Process Reengineering* consider administrative tasks in large organisations, which interact and compete for shared resources. The key entity in such systems is the *business process* ([WFM96]). It consists in a set of linked activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships. A single enactment of a process is a *case*. A *workflow* is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules. A *Worflow Management System* (WFMS for short) is a system that defines, creates and manages the execution of workflows through the use of software which is able to understand the process definition, interacts with workflow participants. It is possible to distinguish two different categories of workflow software which support respectively structured and unstructured processes. *Structured processes* have a fixed behaviour, and they will not change during the time. On the contrary, *unstructured processes* are susceptible to be influenced by external actions.

Numerous WFMS are nowadays available. Unfortunately, there are few theoretical formalisms and tools to deal with these. The need to model and analyse the correctnesss of workflow procedures has lead to use Petri nets as a formalism. They are a well-suited formalism allowing to capture the business processes concepts and structure, i.e. splitting, synchronisation, conflicts, parallel and sequential routing, conditions, ... Moreover, the representation of business processes in terms of Petri nets supports automated manipulation and formal verification techniques. In particular, this last point is extremely important as the numerous WFMS nowadays available ([BPR97], [OSS+97], [Obe94]) generally offer only a simulation based partial verification. A significant work tackling the verification of workflow procedures using Petri nets theory was presented in [vdA97]. The soundness of a procedure, i.e. the correct termination (absence of dangling cases, deadlocks, livelocks, ...), can be checked in polynomial time under some conditions on the flow relation of the net model.

The work we present in this paper significantly extends these results. Indeed, we first take into account the general use of resources shared by workflow procedures. Second, we show how the soundness property, under this resource use constraint, can be proved efficiently by using structural Petri net techniques. The major advantage in using such techniques is to obtain parameterized results. Effectively, we charaterize families of sound models, where the number of cases and resources are considered as parameters. Another benefit of this approach is related to the performances of the whole WFMS. This is done by proposing another distribution of resources between tasks. It helps tailoring the workflow model, while preserving the soundness property.

2 Basic Notions of Petri Nets and Structural Analysis

In this section, we introduce the basic notions and notations used throughout this paper. We first define a Petri net.

Definition 1. *A* Petri net *is a tuple* $PN = < P, T, F, W >$ *where :*

(i) $P \neq \emptyset$ *is a set of* places *;*
(ii) $T \neq \emptyset$ *is a set of* transitions *;*
(iii) $F \subseteq P \times T \cup T \times P$ *is the* flow relation *;*
(iv) $W : P \times T \cup T \times P \rightarrow \mathbb{N} \wedge [W(x,y) = 0 \Leftrightarrow (x,y) \notin F]$ *is the* weight function.

In the following, we define the marking of a Petri net.

Definition 2. *A* marking *of a Petri net PN is a function* $M : P \rightarrow \mathbb{N}$. *The* initial marking *of PN is denoted by* M_0.

We then introduce the notations for pre-sets, post-sets and the incidence matrix.

Notation 1

$$\forall x \in P \cup T,\ {}^\bullet x = \{y \in P \cup T / (y,x) \in F\} \text{ and } x^\bullet = \{y \in P \cup T / (x,y) \in F\}$$

$$\forall (p,t) \in P \times T : C(p,t) = W(t,p) - W(p,t).$$

Then, we recall the firing rules.

Definition 3. *A transition $t \in T$ is* enabled *in a marking M (denoted by $M[t\rangle$) iff $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$.*

If transition t is enabled in marking M, it can be fired, *leading to a new marking M' such that: $\forall p \in P : M'(p) = M(p,t) + C(p,t)$. The firing is denoted by $M[t\rangle M'$.*

The set of all markings reachable from a marking M is denoted by $[M\rangle$.

We define the classical properties checked for Petri nets.

Definition 4. *Let PN be a Petri net and M_0 its initial marking.*

(i) *a marking M_h is a* home state *iff $\forall M \in [M_0\rangle,\ M_h \in [M_0\rangle$;*
(ii) *(PN, M_0) is* reversible *$\Leftrightarrow$ M_0 is a home state ;*
(iii) *(PN, M_0) is* bounded *$\Leftrightarrow \forall p \in P : [\exists k \in \mathbb{N} : \forall M \in [M_0\rangle,\ M(p) \leq k]$*
$\Leftrightarrow$ $[M_0\rangle$ is finite ;
(iv) *(PN, M_0) is* quasi-live *$\Leftrightarrow \forall t \in T : \exists M \in [M_0\rangle,\ M[t\rangle$;*
(v) *(PN, M_0) is* deadlock-free *$\Leftrightarrow \forall M \in [M_0\rangle : \exists t \in T,\ M[t\rangle$;*
(vi) *(PN, M_0) is* live *$\Leftrightarrow \forall t \in T : [\forall M \in [M_0\rangle : \exists M' \in [M\rangle,\ M'[t\rangle]$;*
(vii) *PN is* structurally live *$\Leftrightarrow [\exists M_0,\ (PN, M_0)$ is live$]$.*

Definition 5. *A function $\nu : [M_0\rangle \to \mathbb{N}$ is a* norm *(strict) for a marking $M_h \in [M_0\rangle$ iff:*

(i) *$\nu(M) = 0 \Leftrightarrow M = M_h$;*
(ii) *$\forall M \in [M_0\rangle : [\nu(M) > 0 \Leftrightarrow \exists t \in T : M[t\rangle M' \wedge \nu(M') < \nu(M)]$.*

In this paper, we use techniques from structure theory of Petri nets. Therefore, we introduce the basic notion of invariants.

Definition 6. *Let PN be a Petri net. An integer vector $f, f \neq 0$, indexed by P ($f \in \mathbb{Z}^P$) is a* place invariant *iff it satisfies ${}^t f \cdot C = 0$.*
The positive support *of f is the set of places $\|f\|^+ = \{p \in P : f(p) > 0\}$.*
The negative support *of f is the set of places $\|f\|^- = \{p \in P : f(p) < 0\}$.*
PN is conservative $\Leftrightarrow \exists f,\ f$ p-invariant, $\|f\|^+ = P$
$\Rightarrow \forall M_0,\ N$ is bounded.

A key concept in structural analysis is the siphon.

Definition 7. *Let PN be a Petri net and $S \subseteq P,\ S \neq \emptyset$. S is a* siphon *iff ${}^\bullet S \subseteq S^\bullet$. S is* minimal *iff it contains no other siphon as a proper subset.*

Now, we introduce the notion of controlled siphon.

Definition 8. *Let (PN, M_0) be a Petri net, and S a siphon of PN.*

(i) *S is* controlled *iff $\forall M \in [M_0\rangle, \exists p \in S : M(p) \geq \max_{p^\bullet} = \max_{t \in p^\bullet} W(p,t)$;*
(ii) *(PN, M_0) satisfies the* controlled-siphon property *(cs-property) iff each minimal siphon of PN is controlled.*

Two basic relations between liveness properties and the cs-property are stated in the following proposition.

Proposition 1. *Let (PN, M_0) be a Petri net. The following properties hold:*

(i) (PN, M_0) *live* $\Rightarrow$ (PN, M_0) *satisfies the cs-property ;*
(ii) (PN, M_0) *satisfies the cs-property* $\Rightarrow$ (PN, M_0) *is deadlock-free ;*

Two other properties useful to liveness analysis are recalled below.

Proposition 2. *Let (PN, M_0) be a Petri net and $M_h \in [M_0)$. Then:*

(i) (PN, M_0) *is quasi-live under home state* $M_h \Rightarrow$ (PN, M_0) *is live under* M_h
(ii) M_h *is a home state* $\Leftrightarrow \exists$ *a norm for* M_h.

3 WorkFlow Nets

Van der Aalst showed in [vdA97] that Petri nets modelling business processes generally satisfy some typical properties. They always have two special places i and o, which correspond to the beginning and termination of the processing of a case. They are respectively source and sink places.

Definition 9. *A Petri net PN is a* WF-net *iff:*

(i) PN has two special places: i and o. Place i is a source place: ${}^\bullet i = \emptyset$. Place o is a sink place: $o^\bullet = \emptyset$.
(ii) If we add a transition t^* to PN, connecting place o with i, i.e. ${}^\bullet t^* = \{o\}$ and $t^{*\bullet} = \{i\}$, the Petri net $\overline{PN}$ obtained is strongly connected. $\overline{PN}$ is called the augmented net *of PN.*

A key property of workflow procedures is the *soundness property.* It states that, for any case, the procedure will terminate eventually, and at the moment the procedure terminates, there is a token in place o and all other places are empty. This is a slight extension of the soundness definition given in [vdA97] which considers only one case.

Definition 10. *A WF-net $(PN, n.i)$, n being the number of cases to process, is* sound *iff:*

(i) $\forall M \in [n.i\rangle,\ n.o \in [M\rangle$;
(ii) $\forall M \in [n.i\rangle : M(o) \geq n \Rightarrow M = n.o$;
(iii) $\forall t \in T, \exists M \in [n.i\rangle : M[t\rangle$.

It was proved in [vdA97] that the soundness of a WF-net (PN, i) is equivalent to the liveness plus boundedness of the augmented net $(\overline{PN}, i)$. One can easily extend this property to n cases.

Proposition 3. *A WF-net $(PN, n.i)$ is sound iff $(\overline{PN}, n.i)$ is live and bounded, with $W(o, t^*) = W(t^*, o) = n$.*

Proof. The proof is similar to the one in [vdA97].

$\Leftarrow$: Let us suppose that $(\overline{PN}, n.i)$ is live and bounded. As $(\overline{PN}, n.i)$ is live, $\forall M \in [n.i\rangle : \exists M' \in [M\rangle,\ M'[t^*\rangle$.

Thus, $\forall M \in [n.i\rangle : \exists M' \in [M\rangle,\ M'(o) \geq n$.

Let us suppose that $M' = n.o + M''$, $M'' \neq 0$. Then $M'[t^*\rangle n.i + M''$, which contradicts the boundedness hypothesis. Therefore $M' = n.o$ and conditions (i), (ii) of definition 10 are ensured. Condition (iii) is guaranteed by liveness.

$\Rightarrow$: Let us assume that $(PN, n.i)$ is sound.

We first prove that $(\overline{PN}, n.i)$ is bounded. Suppose that $(PN, n.i)$ is not bounded. Then $\exists M_1 \in [n.i\rangle : \exists M_2 \in [M_1\rangle,\ M_2 > M_1$.

As $(PN, n.i)$ is sound, we know, by definition 10.(i) that $\exists \sigma \in T^* : M_1[\sigma\rangle n.o$. Thus, $\exists M,\ M_2[\sigma\rangle M : M > n.o$. This contradicts the soundness hypothesis (definition 10.(ii)). Thus $(PN, n.i)$ is bounded and therefore $(\overline{PN}, n.i)$ is bounded.

We now prove that $(\overline{PN}, n.i)$ is live. As $(PN, n.i)$ is sound, from definition 10.(i) $\forall M \in [n.i\rangle : n.o \in [M\rangle$. Then, by firing t^*, we obtain: $\forall M \in [n.i\rangle : n.i \in [M\rangle$, i.e. $n.i$ is a home state of $(\overline{PN}, n.i)$. Using definition 10.(iii), we conclude that all transitions are quasi-live and thus, by proposition 2.(i) that $(\overline{PN}, n.i)$ is live. $\diamond$

For a Free-Choice WF-net, proposition 3 can be checked in polynomial time using algorithms based either on the rank theorem ([KB92]) or on Commoner's property ([BM92]). This is due to the fact that checking soundness for one case is equivalent to checking it for any number of cases, as liveness is monotonic for Free Choice nets, contrary to the general case. However, a structural necessary and sufficient liveness condition for Asymmetric Choice nets was presented in [BP96]. This condition, namely *cs-property*, generalizes Commoner's property.

In [vdA97], it is said that most (all but one) of the existing WFMS use constructs corresponding to free-choice nets. In our opinion, non free-choice synchronization patterns can be encountered in practice.

4 Structural Soundness of WF-Nets

Our aim is to consider the number of cases as a parameter, and to monitor the number of cases which can be processed simultaneously, such that the soundness property is satisfied. As in our model liveness is not monotonic, the techniques used for Free Choice nets are not appropriate. Therefore, we introduce the notion of structural soundness.

Definition 11. *A WF-net PN is* structurally sound *iff* $\exists n$ *such that* $(PN, n.i)$ *is sound.*

In order to prove structural soundness of a model, we will use the cs-property. This technique will also allow us, when adding resources, to enhance their use.

In the general case, the cs-property guarantees the absence of deadlock apart from the final marking $M = n.o$. But the soundness is not ensured. In the particular case of Asymmetric Choice WF-nets (including Free Choice WF-nets), the soundness is verified iff the augmented net is bounded and satisfies the cs-property. Moreover, it is possible to characterise a set of initial markings for which soundness is guaranteed.

Now, we will characterise a more general class, abstracting from circuits which can be performed by a case. In fact, WF-nets with circuits can be changed

into WF-nets without circuits, using transformations presented in [vdA97]. From the soundness property, one can easily see that $n.o$ is a home state of $(PN, n.i)$. Using property 2.(ii), there exists a norm for $n.o$. In the particular case of circuit-free nets, this norm can easily be built, as shown in the proof of theorem 1, where the soundness property is structurally characterized.

Definition 12. *A net PN is a* Circuit Free WF-net *(CFWF) iff PN is a WF-net and has no circuit.*

For a bounded and quasi-live CFWF, the cs-property is a necessary and sufficient soundness condition. One can consider easily that boundedness and quasi-liveness properties are minimal requirements.

Theorem 1. *Let PN be a CFWF. PN is structurally sound iff $\exists n$ such that $(\overline{PN}, n.i)$ is bounded, quasi-live and satisfies the cs-property.*

Proof. $\Rightarrow$: Let us suppose that PN is structurally sound. This means (definition 11) that $\exists n$ such that $(PN, n.i)$ is sound. From proposition 3, $(\overline{PN}, n.i)$ is live and bounded, in particular bounded and quasi-live. Moreover, from proposition 1.(i), it satisfies the cs-property.

$\Leftarrow$: Let us suppose that $\exists n$ such that $(\overline{PN}, n.i)$ is bounded, quasi-live and satisfies the cs-property. We will first exhibit a norm ν for marking $n.o$. We construct function ν as follows: we number the places in reverse topological order, i.e. place o is numbered 0, place i has the highest number, and the other places are such that a successor p' of a place p in the graph of the Petri net has a lower number than place p. This can be done due to the absence of cycles. We call num this numbering function. Then we define $\forall M : \nu(M) = \sum_{p \in P} M(p) num(p)$.

We now prove that ν is a norm for $n.o$. By construction, $\nu(M) = 0 \Leftrightarrow M = x.o$. If $x < n$, then $\nexists t \in T : M[t\rangle$. As $(\overline{PN}, n.i)$ satisfies the cs-property, we deduce from proposition 1.(ii) that $(\overline{PN}, n.i)$ is deadlock-free, i.e. $\forall M \in [n.i\rangle : \exists t \in T,\ M[t\rangle$. Thus, there is a contradiction. If $x > n$, $x.o[t^*\rangle M' > n.i$. This contradicts the boundedness hypothesis. Thus, we have proved condition (i) of definition 5.

Let us suppose that $\nu(M) > 0$ for a marking M. As $(\overline{PN}, n.i)$ is deadlock-free, $\exists t : M[t\rangle M'$. If $t \neq t^*$, by construction of ν, $\nu(M') < \nu(M)$. Otherwise $(t = t^*)$, as $\nu(M) > 0$ and $\nu(M) = 0 \Leftrightarrow M = n.o$ (already proven), marking M must have the form $M = n.o + M''$ with $M'' \neq 0$. Then $M[t^*\rangle n.i + M'' > n.i$, which contradicts the boudedness hypothesis. Thus, $\Rightarrow$ of definition 5.(ii) is satisfied.

Let us now suppose that $\exists t \in T : M[t\rangle M' \wedge \nu(M') < \nu(M)$. The construction of function ν is such that $\forall M : \nu(M) \geq 0$. Then $\nu(M) > \nu(M') \geq 0$. Thus, $\Leftarrow$ of definition 5.(ii) is satisfied.

We deduce from all this that we found a norm function ν for $n.o$.

From proposition 2.(ii), $n.o$ is a home marking. Then, as $n.o[t^*\rangle n.i$, $n.i$ also is. As $(\overline{PN}, n.i)$ is quasi-live and its initial state is a home state, it is live. From proposition 3, $(PN, n.i)$ is sound. $\diamond$

In some subclasses of Petri nets, these last conditions can be lessened.

Definition 13. *A Petri net PN is an* Asymmetric Choice net *iff:*

$$\forall (p,q) \in P \times P : [p^\bullet \cap q^\bullet \neq \emptyset \Rightarrow p^\bullet \subseteq q^\bullet \vee q^\bullet \subseteq p^\bullet].$$

Corollary 1. *Let PN be an Asymmetric Choice CFWF-net. PN is structurally sound iff* $\exists n$ *such that* $(\overline{PN}, n.i)$ *is bounded and satisfies the cs-property.*

Proof. For Asymmetric Choice nets, the cs-property is a necessary and sufficient liveness condition ([BP96]). The result follows from the fact that liveness implies quasi-liveness and from theorem 1. ◇

In the next section, we will add resources to sound nets in order to study the adequation between the number of resources available and the number of cases to be handled.

5 Coping with Shared Resources

In this section, we first introduce the model of WF-nets with resources and combine them into a system where they share resources. Secondly, we show how to perform analysis of structural soundness.

5.1 Modelling Business Processes Competing for Shared Resources

We introduce the notion of WF-net with resources. It is basically a WF-net plus a set of places modelling the resources. We demand the WF-net (without resources) to be sound, and the resources to be preserved by the net, i.e. a resource requested will eventually be released and a resource released has previously been requested. Several resources can be requested/released at a same time. The resource preservation can be expressed by a place invariant of the system (definition 14.(v)). One can note that the subnet associated with this invariant is not necessarily a state machine.

Definition 14. *A* WFR-net *is a tuple* $PNR = < P \cup P_R, T, F \cup F_R, W \cup W_R >$ *where:*

(i) $PN = < P, T, F, W >$ *is a* structurally sound *WF-net.*
(ii) $P_R \neq \emptyset \wedge P \cap P_R = \emptyset$ *(set of resources)*
(iii) $F_R \subseteq (P_R \times T) \cup (T \times P_R)$ *(flow relation for resources)*
(iv) $\forall u \in F_R, W_R(u) \geq 1$ *(resource use)*
(v) $\forall r \in P_R, \exists f_r \geq 0 : {}^t f_r \cdot C = 0 \ \wedge \|f_r\| \cap P_R = \{r\}$ *(resource preservation)*

Then, we can compose several WFR-nets into a system where they share resources. This is obtained by fusion of the places representing the shared resources.

Definition 15. *A* WFRS *is recursively defined. A WFR-net is a WFRS.*

Let $PN_i = < P_i \cup P_{Ri}, T_i, F_i, W_i >, i \in \{1,2\}$, *be two WFRS such that* $P_1 \cap P_2 = T_1 \cap T_2 = \emptyset$. *We denote the set of shared resources by* $P_{R1R2} = P_{R1} \cap P_{R2}$. *The net* $PN = PN_1 \circ PN_2$ *resulting of the fusion of nets* PN_1 *and* PN_2 *over the set* P_{R1R2} *is a WFRS.*

Definition 16. *Let N be a WFRS. We denote by $\overline{N}$ the generalized Petri net constructed as N, where the WF-nets components (definition 14.(i)) are replaced by the corresponding augmented WF-nets.*

One can note that, compared to other classes presented in the literature, WFRS extend S^4R-nets ([BBA96]), which are a generalisation of S^3R-nets ([ECM95]) proposed to cope with deadlocks in flexible manufacturing systems.

5.2 Structural Analysis

The definition of structural soundness (definition 11) can easily be extended to WFRS.

Lemma 1. *Let N be a WFRS, and S a minimal siphon of $\overline{N}$. There exists an initial marking M_0 under which S is controlled.*

Proof. Let N be a WFRS and S a minimal siphon of $\overline{N}$.

Let us first suppose that $S \cap \bigcup P_{Ri} = \emptyset$. By construction, $\exists PN_i : S \subseteq P_i$. As PN_i is structurally sound, there exists an initial marking under which S is controlled.

Let us now consider the complementary case: $S \cap \bigcup P_{Ri} \neq \emptyset$. We suppose that siphon S is not controlled. We denote by $f(r)$ a flow of minimal support associated with a resource r, and by $f(p)$ a flow of minimal support associated with p in its WF-net. Let:

$$g_S = \sum_{r \in S \cap \bigcup P_{Ri}} f(r)$$

$$Out(S) = \|g\| \setminus S$$

$$h_S = \sum_{p \in Out(S)} f(p)$$

$$\lambda_S = \max_{p \in Out(S) \cap \|h_S\|} g(p)$$

$$z_S = g_S - \lambda_S . h_S$$

Siphon S is controlled as soon as:

$${}^t z_S \cdot M_0 > \sum_{p \in S} z_s(p).(\max_{p^\bullet} - 1)$$

Therefore, there exists a marking under which S is controlled. ◇

The following property holds for any WFRS.

Lemma 2. *Let N be a structurally sound WFRS. There exists an initial marking M_0 such that $(\overline{N}, M_0)$ satisfies the cs-property.*

Proof. Let N be a structurally sound WFRS. If it does not satisfy the cs-property, it cannot be live (proposition 1.(i)). Thus, it cannot be sound. ◇

In theorem 2, we extend lemma 2 in the particular case where the components PN_i are circuit free.

Theorem 2. *Let N be a WFRS where the PN_i are CFWF. N is structurally sound iff there exists an initial marking M_0^* under which $(\overline{N}, M_0^*)$ is bounded, quasi-live and satisfies the cs-property.*

Proof. $\Leftarrow$: The cs-property is ensured by lemma 2. The boundedness and quasi-liveness are deduced from soundness.

$\Rightarrow$: Let N be a WFRS where the PN_i are CFWF. Let us suppose that there exists an initial marking M_0^* under which $(\overline{N}, M_0^*)$ is bounded, quasi-live and satisfies the cs-property. We will now prove that $(\overline{N}, M_0^*)$ is live. To do that, we proceed as in the proof of theorem 1, i.e. we will exhibit a norm ν_R. This norm ν_R is an extension of norm ν where the resources are numbered 0. The proofs of the properties of a norm are similar to those in theorem 1, taking also into account resource preservation (definition 14.(v)). ◇

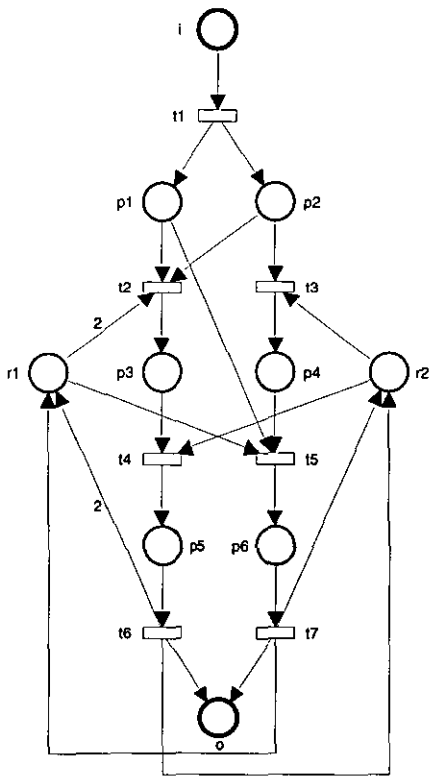


Fig. 1. A Circuit-Free WFRS.

We will now apply theorem 2 to the net of figure 1. The minimal siphons of the augmented net are:

$S_1 = \{i, p1, p3, p5, p6, o\}$
$S_2 = \{i, p2, p3, p4, p5, p6, o\}$
$S_3 = \{p3, p5, p6, r1\}$
$S_4 = \{p4, p5, p6, r2\}$
$S_5 = \{r1, r2, p5, p6\}$
$S_6 = \{i, p1, p5, p6, o, r2\}$

The WF-net composing the WFRS is structurallay sound: it is live and bounded for e.g. $M_0(i) = 1$. Thus, the two siphons without resource places, S_1 and S_2 are controlled as soon as $M_0(i) > 0$. Siphons S_3 and S_4 are the support of positive flows. Hence they are invariant-controlled. The control condition for $S3$ is $M_0(r1) > 1$, and for S_4, $M_0(r2) > 0$. We now have to examine siphons S_5 and S_6 more in detail in order to calculate their control condition.

$g_{S_5} = f(r1) + f(r2)$
$= r1 + 2.p3 + 2.p5 + p6 + r2 + p4 + p5 + p6$
$= r1 + r2 + 2.p3 + p4 + 3.p5 + 2.p6$
$Out(S_5) = \{p3, p4\}$
$h_{S_5} = f(p3) + f(p4)$
$= i + p1 + p3 + p5 + p6 + o + i + p2 + p3 + p4 + p5 + p6 + o$
$= 2.i + p1 + p2 + 2.p3 + p4 + 2.p5 + 2.p6 + 2.o$
$\lambda_{S_5} = 2$
$z_{S_5} = r1 + r2 + 2.p3 + p4 + 3.p5 + 2.p6$
$-4.i - 2.p1 - 2.p2 - 4.p3 - 2.p4 - 4.p5 - 4.p6 - 4.o$
$= r1 + r2 - 4.i - 2.p1 - 2.p2 - 2.p3 - p4 - p5 - 2.p6 - 4.o$
S_5 is controlled as soon as $M_0(r1) + M_0(r2) - 4.M_0(i) > 1$.
$g_{S_6} = f(r2)$
$= r2 + p4 + p5 + p6$
$Out(S_6) = \{p4\}$
$h_{S_6} = f(p4)$
$= i + p2 + p3 + p4 + p5 + p6 + o$
$\lambda_{S_6} = 1$
$z_{S_6} = r2 + p4 + p5 + p6 - i - p2 - p3 - p4 - p5 - p6 - o$
$= r2 - i - p2 - p3 - o$
S_6 is controlled as soon as $M_0(r2) - M_0(i) > 0$.

To conclude, the net of figure 1 is controlled as soon as the following inequalities are satisfied:

$$M_0(r1) > 0,\ M_0(r2) > 0,\ M_0(i) > 0,$$

$$M_0(r1) + M_0(r2) - 4.M_0(i) > 1,\ M_0(r2) - M_0(i) > 0$$

In the next section, we show how to enhance the performances of the system by allowing more cases to enter the system and still preserve the structural soundness property.

6 Enhancing Performances of WFRS

The control we have up to now is global, and a thinner control can only improve the concurrency, i.e. increase the number of concurrent cases in the system. The basic idea is to dissociate the control of the siphons from the input places of the WF-nets constituting the WFRS. In practice, these places can be considered as part of the environment.

The set of minimal siphons of a given WFRS can be partitionned into 3 classes. The first class (type 1) contains the minimal siphons without resource places. They are controlled since the WF-nets constituting the WFRS are sound. The second class (type 2) contains those which include resources and are invariant-controlled in the sense of [BP96]. The last class (type 3) contains the minimal siphons including resource places but not invariant-controlled.

We associate, with each siphon S of type 3, a local control place C_S with:

$$C_S{}^\bullet = {}^\bullet Out(S),\ {}^\bullet C_S = Out(S)^\bullet$$

$$\forall p \in Out(S),\ \forall t \in {}^\bullet p,\ \forall t' \in p^\bullet : W(C_S, t) = W(t', C_S) = g(p)$$

One can easily avoid self-loops introduced by the flow relation restricted to C_S, since this operation preserves the invariant and thus the future control. Adding place C_S has created a new flow:

$$f(C_S) = C_S + \sum_{p \in Out(S)} g(p).p$$

Let $z_{C_S} = g_S - f(C_S)$. For siphon S to be controlled, we must have:

$${}^t z_{C_S} \cdot M_0 > \sum_{p \in S} z_{C_S}(p).(\max_{p^\bullet} - 1)$$

New control places behave like resources, i.e. they satisfy the resource preservation condition of definition 14.(v). Hence the net with these new control places is a WFRS.

Let us now consider the simple example in figure 2 without the grey part (place C). The initial marking is parameterized by $M_0(i)$, $M_0(r1)$ and $M_0(r2)$. The initial marking is $M_0 = M_0(i).i + M_0(r1).r1 + M_0(r2).r2$. Our example presents one minimal siphon of each of the 3 types:

$S_1 = \{i, p1, p2, p3, p4, p5, p6, o\}$ (type 1)

$S_2 = \{r2, p2, p3, p5\}$ (type 2)

$S_3 = \{r1, p2, p3, p4, p5, p6\}$ (type 3)

Siphons S_1 and S_2 are the support of positive flows. S_1 is controlled as soon as $M_0(i) > 0$, S_2 is controlled for $M_0(r2) > 0$.

We now consider siphon S of type 3 for which the control is not guaranteed. We use the same notations for flows associated with places and for the calculus of the cs-property as in the proof of lemma 1. For S_3 in our example:

$g_{S_3} = f(r1)$

$= r1 + 2.p1 + 4.p2 + 3.p3 + 5.p4 + 3.p5 + 3.p6$

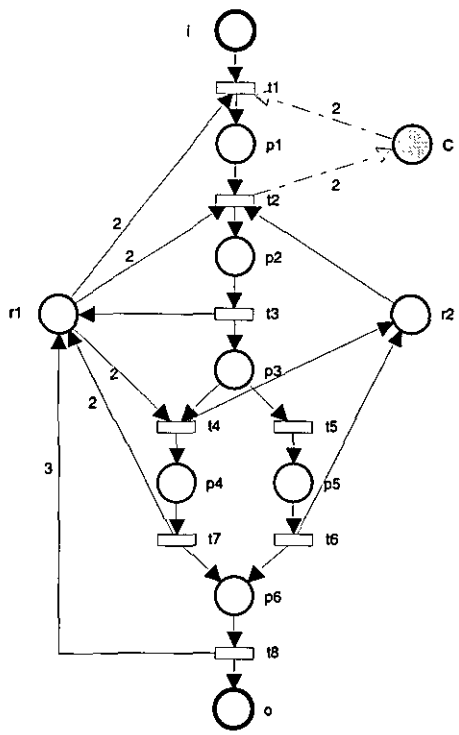


Fig. 2. A Free Choice Circuit-Free WFRS.

$Out(S_3) = \{p1\}$
$h_{S_3} = f(p1)$
$\quad = i + p1 + p2 + p3 + p4 + p5 + p6 + o$
$\lambda_{S_3} = 2$
$z_{S_3} = r1 + 2.p1 + 4.p2 + 3.p3 + 5.p4 + 3.p5 + 3.p6$
$\qquad -2.i - 2.p1 - 2.p2 - 2.p3 - 2.p4 - 2.p5 - 2.p6 - 2.o$
$\quad = r1 - 2.i + 2.p2 + p3 + 3.p4 + p5 + p6 - 2.o$

Thus, S_3 is controlled as soon as $M_0(r1) - 2.M_0(i) > 1$. We conclude, using theorem 2, that this net is structurally sound for any initial marking $M_0 = M_0(i).i + M_0(r1).r1 + M_0(r2).r2$ such that:

$$M_0(i) > 0,\ M_0(r2) > 0,\ M_0(r1) > 2.M_0(i) + 1$$

Hence, for example if $M_0(r1) = 7$, $M_0(r2) = 1$ then $M_0(i) \leq 2$, i.e. at most 2 cases can be simultaneously processed.

We now want to enhance the performances of the example of figure 2. S_3 is the only siphon of type 3. Thus, we will control it more locally. Since $Out(S_3) = \{p1\}$, the associated local control place C (see figure 2) satisfies: $C^\bullet = \{t1\}, {}^\bullet C = \{t2\}$, $W(C, t1) = W(t2, C) = 2$. We have: $f(C) = C + 2.p1$. Then:

$z_C = g_{S_3} - f(C)$
$\quad = r1 + 4.p2 + 3.p3 + 5.p4 + 3.p5 + 3.p6 - C$

Siphon S_3 is controlled for any marking satisfying $M_0(r1) - M_0(C) > 1$.

For example, if $M_0(r1) = 7$, $M_0(r2) = 1$, we must have $M_0(C) \leq 5$. The introduction of place C adds only one siphon $S_4 = \{C, p1\}$, which is of type 2, and thus controlled when $M_0(C) > 1$. The new WFRS is structurally sound for the initial marking we have exhibited. Without place C, we could only handle 2 cases at a time. Now, we can process 3 cases simultaneously. Thus the throughput, resources use and parallelism are better with a local control than with a global one. One could object that this new place can add siphons of type 3. If there are new uncontrolled siphons, they can be handled in the same manner, either locally or globally. It was proved in [Bar97] that this iterative process necessarily stops: we eventually reach a step where the role of the control place to be added can be played by an already existing one.

7 Conclusion

Many researchers have investigated properties related to the soundness property. In this work, we have shown that, for an important subclass of WF-nets sharing resources, called WFRS, the soundness property can be structurally characterized. The technique presented has a great advantage compared to other approaches based on the computation of the reachability set. Indeed, if the initial marking, i.e. the number of cases involved and resources availability, is modified, the soundness checking requires only to compute the initial markings of the control places.

We are currently applying our method in the workflow of an hospital ([Car97]) in order to enhance the performances of an operating theatre block by performing reengineering. This can be achieved by introducing a balance between specialisation and generalisation or between centralisation and decentralisation of resource classes (human and material) of the system.

References

[Bar97] K. Barkaoui. Control systems design for automated manufacturing systems. Technical report, CEDRIC, 1997.

[BBA96] K. Barkaoui and I. Ben Abdallah. Analysis of a resource allocation problem in FMS using structure theory of Petri nets. In *Proceedings of the* 1st *International Workshop on FMS and Petri Nets*, Osaka, June 1996.

[BM92] K. Barkaoui and M. Minoux. A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets. *LNCS*, 616, June 1992.

[BP96] K. Barkaoui and J-F. Pradat Peyre. On liveness and controlled siphons in Petri nets. *LNCS*, 1091, June 1996.

[BPR97] *BPR tools description.* http://www-a5.igd.fhg.de/reuse-m/bpr-description.html, 1997.

[Car97] P. Carvalho. Modelling, by queuing network, of hospital operating theatre block. Mémoire d'ingénieur CNAM, March 1997.

[ECM95] J. Ezpeleta, J.M. Colom, and J. Martinez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(2), April 1995.

[KB92] P. Kemper and F. Bause. An efficient polynomial-time algorithm to decide liveness and boundedness of free-choice nets. *LNCS*, 616, 1992.

[Obe94] A. Oberweis. Workflow management in software engineering. In *Proceedings of the 2nd International Conference on Concurrent Engineering and Electronic Design Automation*, Bournemouth, April 1994.

[OSS+97] A. Oberweis, R. Schätzle, W. Stucky, W. Weitz, and G. Zimmermann. INCOME/WF - A Petri net based approach to workflow management. In *Wirtscheftsinformatik'97*. Springer-Verlag, 1997.

[vdA97] W. M. P. van der Aalst. Verification of workflow nets. *LNCS*, 1248, June 1997.

[WFM96] *Workflow Management Coalition Specification: terminology and glossary.* http://www.aiai.ed.ac.uk/WfMC/DOCS/glossary/glossary.html, 1996.

Modeling and Verification of Workflow Nets

M. Voorhoeve (email: wsinmarc@win.tue.nl)

Eindhoven University of Technology

Abstract

A semantics for workflow processes is proposed, based on tasks that have a duration and may be executed concurrently. The semantics supports operators that can be used to compose processes from simpler ones. An important operator is *refinement*, replacing a task by a process. By abstracting from certain tasks, a related notion allows verifications based upon step by step reduction of the process. The approach is illustrated by means of an example Petri net model.

Keywords: Workflow, Concurrency, Validation, Verification.

1 Introduction

Workflow management is an important new development in the computerized support of human work. As such, it is an emerging market with scores of commercially available products, not to mention research prototypes at universities. A workflow management system (WFMS) focuses on *cases* flowing through the organization, while *tasks* are executed for them, needing *resources*.

A workflow management system needs models that describe the cases, tasks and resources, and the way they interact. This interaction is modeled by *stages* or states that cases may be in. The stage of a case determines the possible tasks that can be executed. After executing such a task, the case moves to a new stage. Each task has a set of resources required for its execution. The WFMS elicits the proper tasks at the proper stage of a case and keeps track of its progress. See [10] for an overview of workflow terminology.

In this paper we limit ourselves to the *process* aspect: determining which tasks or *actions* can be executed in which stage of a given case. This aspect of workflow can be aptly described by Petri Nets [7], [1].

In Figure 1, such a net is shown, modeling the process of travel arrangement. A travel request initially enters the process and three parallel activities are started. A budget check is performed and the hotel and travel requirements are studied. If necessary, hotel and travel information is obtained. After obtaining enough information and receiving a budget approval, travel and hotel accommodation is booked, the travel documents and budget approval are assembled and sent to the client.

Models such as the one in Figure 1 can be understood with a little training and are *formal*, which means they represent precisely defined mathematical

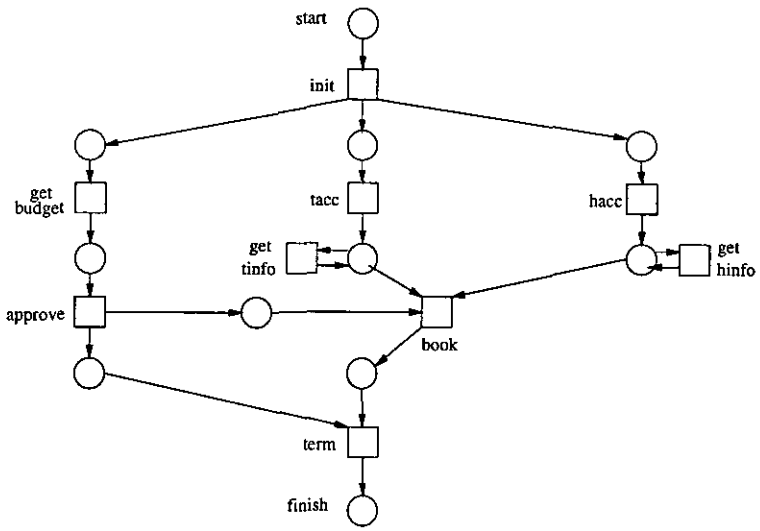


Figure 1: A workflow process model: travel department

objects. This allows rigorous *verification* of a model, e.g. by model checking. One states desired properties for the workflow process and checks whether the model satisfies them.

Related to modeling is the comparison of different model proposals for the same problem. All models may satisfy the stated requirements, however one would like to see whether and how they differ, in order to choose the most appropriate one.

A third issue addresses the maintenance of workflow processes. One must be able to modify a process on an ad-hoc basis (e.g. due to temporary absence of a resource) or permanently. Many workflow processes have an inherent *protocol* with some external party. In Figure 1 there will be a client wishing his trip to be arranged, who starts the process and receives the final trip documents. In between, the client must be prepared to answer requests from the `get info` actions, giving additional information. The client, however, is unaware of the execution of other (internal) actions. Often a modification may not affect the protocols with certain external parties. After *abstracting* from internal actions the old and new nets must be equivalent. This problem has been addressed in [2] and [9].

The last two issues have something in common. One is about (in)equality of processes in all respects, and the other about equivalence to a certain extent. This equality and equivalence is the subject of this paper. We define a *semantics* for processes, making it possible to conclude whether different nets model the same system. A related notion deals with equivalence. This last notion can speedup model checking, by checking a reduced model instead of the original one. The reduced model is obtained after abstracting from actions that the

requirement is not addressing and applying certain reduction rules that allow e.g. to remove nodes and arcs in the net.

In the remainder of the paper, we first define a workflow process as a class of equivalent nets. This equivalence class is the semantics of a given net model. We then define some operators for workflow nets, allowing the construction of workflow nets by composition and refinement. Replacing the composing nets by equivalent ones in a construction will result in net equivalent to the original one. This means that the operators have a definition for workflow processes, as intended. We show how the example process modeled by Figure 1 can be constructed from simple actions alone. We finally introduce a related equivalence notion that allows one to abstract from certain actions and give an example reduction.

2 Semantics

A workflow process - either modeled by a net or otherwise - is based on *actions*. A process can be represented by a graph where the nodes are states and the (directed) edges denote state changes. The graphs representing workflow processes will be called workflow graphs. To each state correspond a *bag* (or multiset) of actions that are busy executing and a set of actions that are *enabled.* [1] In addition, there are two special states: the initial and terminal state. In the initial and terminal state no actions are busy; in the terminal state no actions are enabled.

A process can move from state to state in various ways. An enabled action can *start*. The started action is added to the busy part of the state. The started action and other enabled actions may then become disabled. Starting an action does not enable new actions.

A busy action can be removed from the busy part of the state in three ways. It may *commit*, whereby new actions may become enabled. It may also *rollback*, whereby it may become enabled again, together with other actions that became disabled when starting it. Finally it may *abort*, disappearing from the state without further enabling or re-enabling any actions. Only rollback actions allow to reach the initial state.

Two workflow graphs are said to be *bisimilar* if there exists a *bisimulation* between their nodes. A bisimulation is a relation between the nodes of graphs such that if nodes, say, r and s are related,

i) r is initial iff s is initial,

ii) r is terminal iff s is terminal,

iii) r and s have the same bag of busy actions,

iv) to any state s' reachable from s by start, commit, rollback or abort corresponds a related state r' reachable from r in the same way,

[1] The set of enabled actions is in fact redundant.

v) to any state r' reachable from r by start, commit, rollback or abort corresponds a related state s' reachable from s in the same way.

Bisimilar workflow graphs are equivalent: there is no way to tell the processes apart. Workflow processes are defined as the *equivalence classes* of workflow graphs modulo bisimilarity. Notions about workflow processes are defined for graphs and must be proved to be preserved modulo bisimilarity, i.e. if the notion holds for a given graph, it must also hold for a graph bisimilar to it. One such notion is *soundness*. A workflow graph is *sound* iff every state reachable from the initial state can reach the terminal state in the same way. If two graphs are bisimilar and one is sound, the other is sound too. So soundness is a property of processes.

A process is modeled by a labeled Petri net like in Figure 1. The net must have special initial and terminal places and labeled transitions. In most cases there is a single initial and another single terminal place. Every transition label corresponds to an action in the workflow system. We describe how a net defines a graph, and thus models a process.

A *state* of the net is a bag (multiset) of both places and transitions. Given a state, every node (place or transition) has a finite nonnegative weight. The initial state of the process corresponds to the state where the initial places have weight one and the other nodes weight zero. The terminal state corresponds to the state where the terminal places have weight one and the other nodes weight zero.

Given a state, the bag of busy actions is obtained by summation of the labels of the transitions in the state. An action is enabled iff a transition with that label has all its input places marked. Starting an enabled action results in the state with the enabled action added to the state and its input removed. Committing it removes the action and adds its output. Rollback is the inverse of start. Aborting a busy action removes it from the state without changing the marking of the other nodes.

In Figure 2, the complete graph of a given workflow net is depicted, i.e. all the edges and nodes connected to the node representing the initial state. The start and rollback events are denoted by a solid line with two arrowheads. The commit event by a solid line with a single arrowhead The abort event with a dashed line. The depicted states are those that can be reached from the initial state by any kind of event. On the left four marked nets are depicted that correspond to states of the graph. For reasons of space, the other seventeen are not shown, but it is not hard to construct one from the information given. The correspondance between the nets and the node labels in the graph can be easily assessed. In e.g. the second net a b-labeled action is running and another b-labeled action is enabled; indeed, the corresponding node is labled with the pair (b, b).

In Figure 3, bisimilar graphs are depicted. The graph on the left is the smallest one (i.e. with the least number of states) that is bisimilar to the graph on the right. It is easy to ascertain that no nodes with the same label in the left-hand

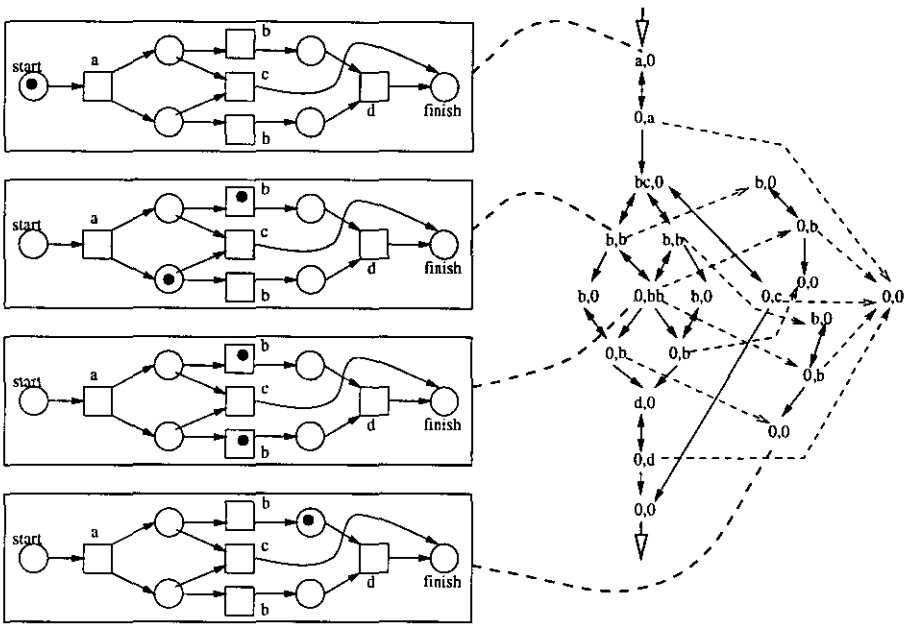


Figure 2: Nets and graph

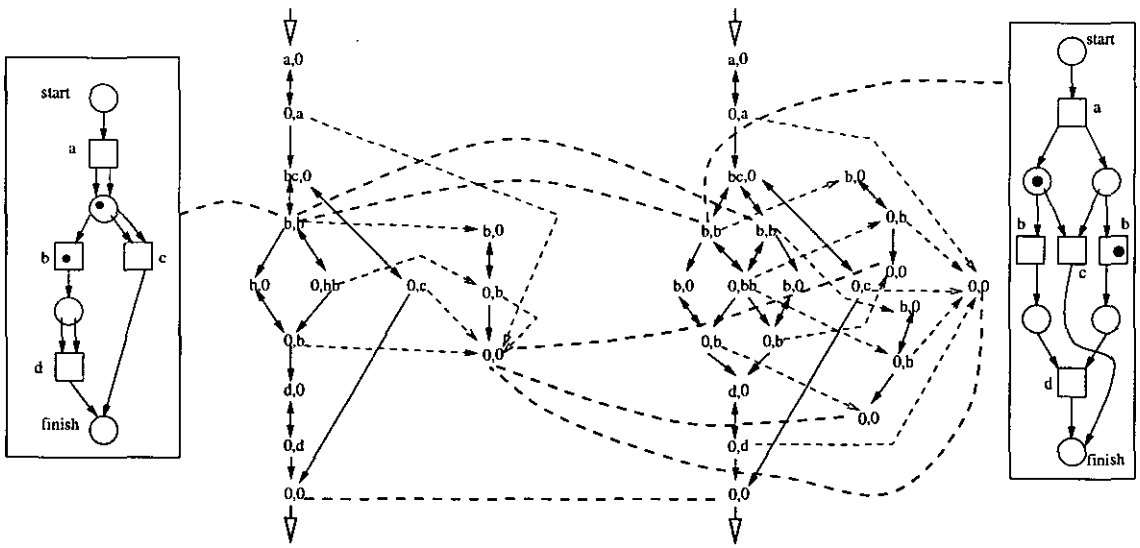


Figure 3: Bisimilar graphs

net can related by a bisimulation to one and the same node. For instance, there are two nodes labeled $(0,0)$ in the graph, but one of therm is terminal, whereas the other is not. The left-hand graph corresponds to a net with weighted (multiple) arcs as indicated. So the two nets depict the same process. Indeed, one can be seen as the *reduction* of the other. Several relations between states are given in the figure.

3 Composition of workflow processes

In this section we define some operators that allow to construct workflow processes. A construction involves an operator with one or more parameter graphs (or nets) giving a result graph (net). Bisimilarity is a *congruence* for these operators, which means that replacing parameters in a construction by bisimilar ones gives a bisimilar result. This means that the operators can be defined on processes (equivalence classes modulo bisimilarity) and thus are true operators on a semantical level.

The operators are sequencing, choice, iteration, free merge, synchronous communication, asynchronous communication, relabeling and refinement. We give a net-based description; formal net-based and graph based definitions can be given (as in [3] and [8]).

Sequencing, choice and iteration share the notion of place fusion. Fusing a set A of places to another set B of places means adding a new place for every pair (a,b) of places from $A \times B$, adding an edge to the place corresponding to (a,b) iff there exists a similar edge to a or b and then removing the places in A and B and edges from or to them. If A and B are singleton sets, this is equivalent to ordinary fusion. If A or B is empty, it becomes removal of places and edges. In other cases a kind of "weaving" occurs, as illustrated in Figure 4. In the figure, place identifiers are added to illustrate the correspondence between the original and the new places.

The sequencing and choice operators have two parameter nets. Sequencing fuses the terminal places of one net to the initial places of the other, so that the second net can start iff the first one has terminated. Choice fuses the initial and terminal places of its two parameter nets so that either net can start, disabling the other. Iteration has three parameter nets; the terminal places of the first are fused with the initial and terminal places of the second and the initial places of the third, thus creating a loop.

The merge operator juxtaposes two nets. The initial and terminal places of the merge result are obtained by taking the union of the initial, respectively terminal places of the parameter nets. These disjoint nets can be connected by communication. Synchronous communication fuses transitions, so that the actions they represent are executed simultaneously. This involves relabeling the synchronized transitions. Asynchronous communication adds places, so that some actions must wait until others have occurred. Relabeling means applying a function to the transition labels. Refinement means substituting a net for the

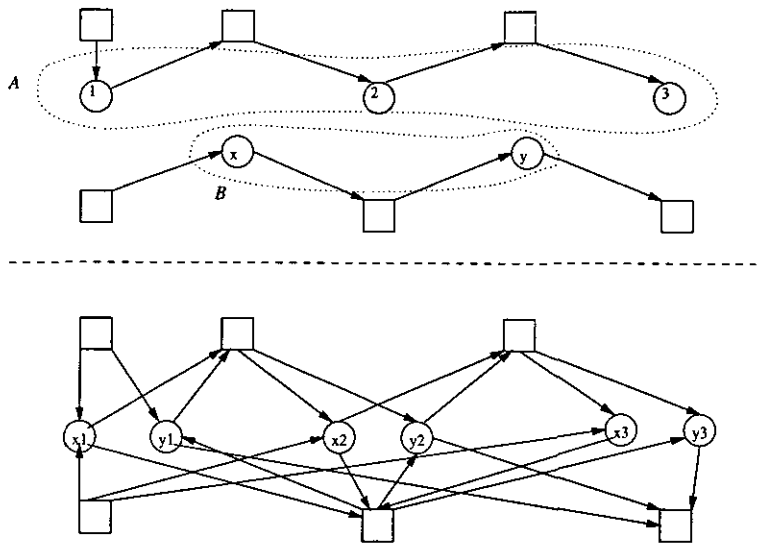


Figure 4: Place fusion

transitions with a given label. Here, another place fusion occurs: the initial places of the net refining the transition are fused with the input places of that transition. Likewise, the output places of the transition are fused to the terminal places of the net refining it.

In Figure 5, the operators defined above are illustrated. We see the nets A and B, the sequencing $A.B$ (A followed by B), the choice $A+B$ (A or B), their merge $(A||B)_{d\to c}$ with asynchronous communication c after d, their merge $A||B)_{d|c=f}$ with synchronous communication of d and c to f, the iteration $BA*B$ (B followed by iterated A, terminated by B) and the refinement $A[b \leftarrow B]$ (A with action b refined to B. The relabeling operator does not occur in the figure.

The synchronization operators must be used with great care; note that all nets in Figure 5 are sound, except for the two nets whose construction involves synchronization. Rather than formally defining the operators, we show the construction of our trip planning example, illustrated in Figure 6.

By sequential composition of three actions, the first net is arrived at. Now the middle action `do_work` is refined into the parallel composition (merge) of administration, travel and hotel actions. These actions on their turn are refined into nets involving sequencing and iteration. Asynchronous communication causes the hotel and travel booking to wait for budget approval. The hotel and travel booking is performed synchronously (as they influence one another). A reduction modulo bisimilarity can be performed, giving the net in Figure 1.

The construction operators can be incorporated in an editor for workflow processes. All operators except those involving communication preserve the soundness property: the construction is sound if all the parameters are sound.

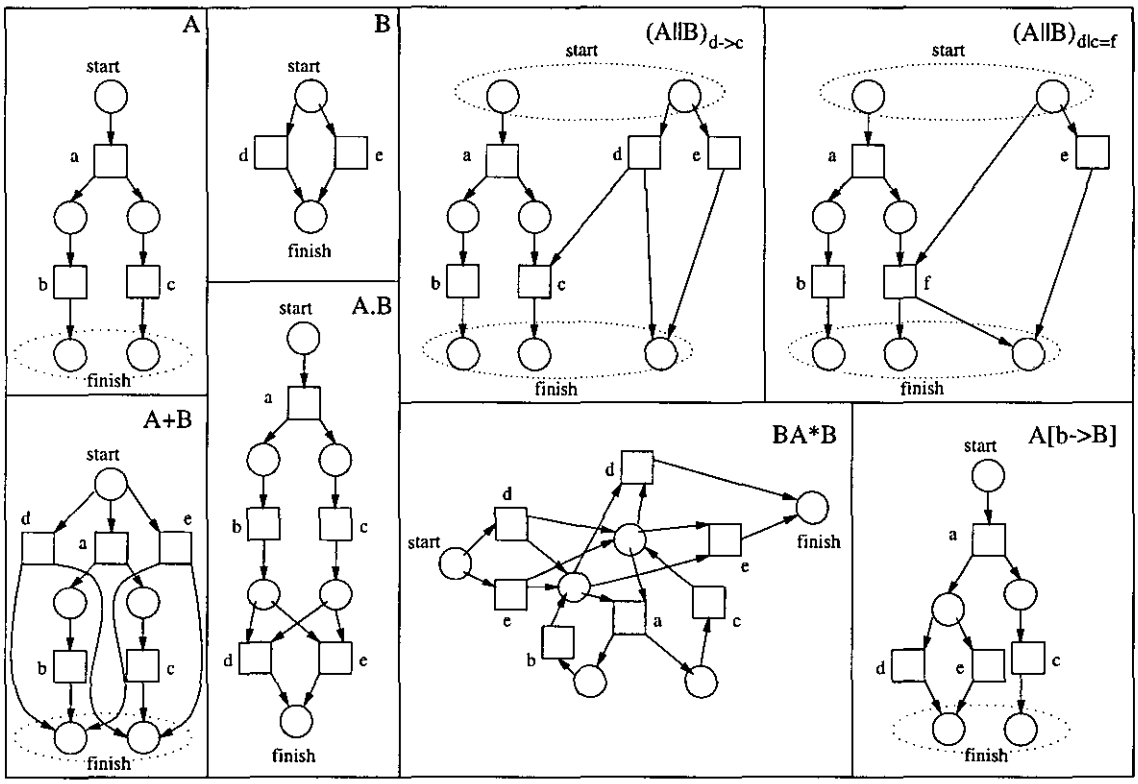


Figure 5: Workflow net operators

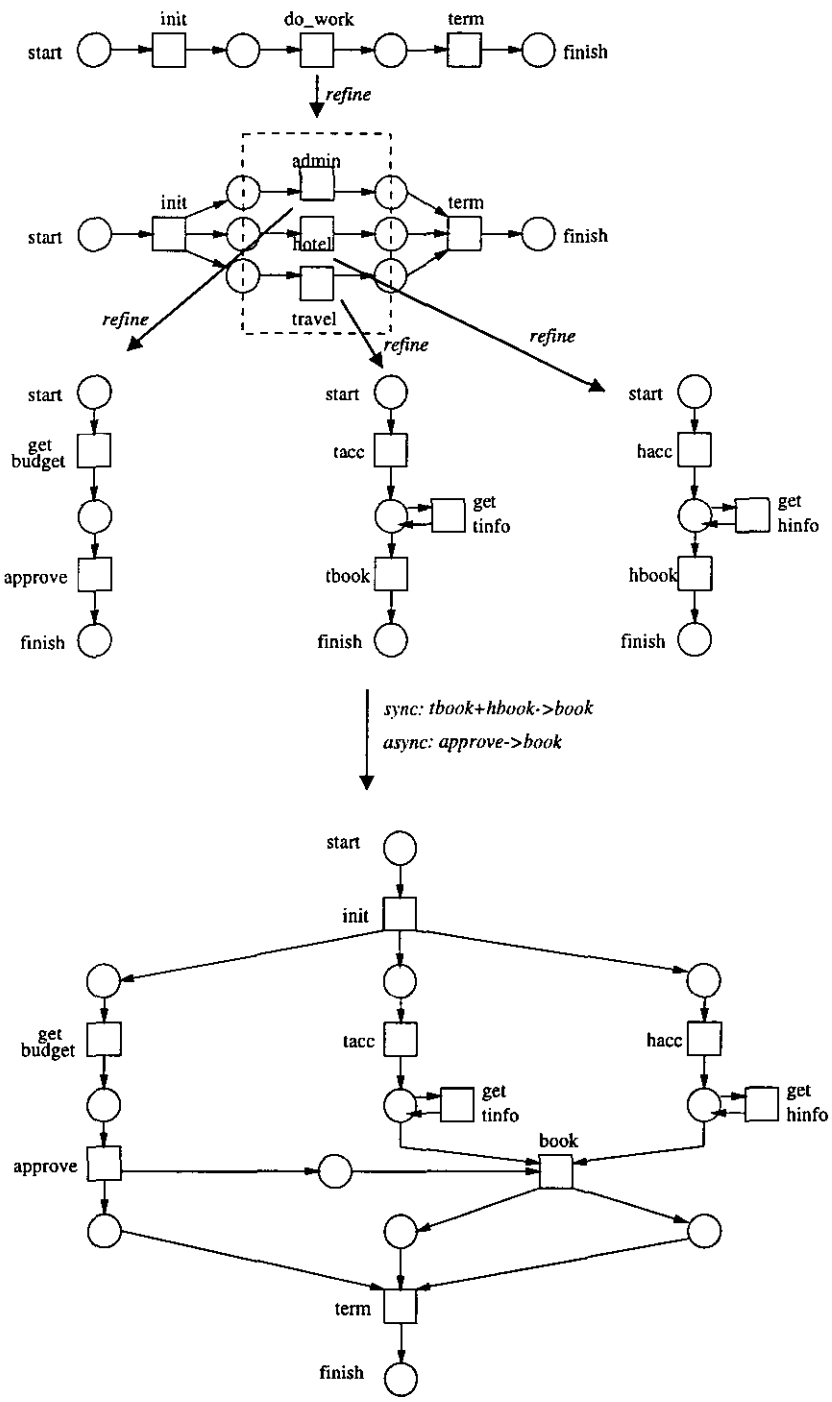


Figure 6: Top-down net composition

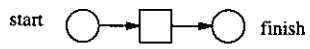


Figure 7: Prototype sound net

4 Verification

The primary reason for modeling workflow is the possibility to control and monitor the work by means of a WFMS. However, the existence of a formal model also allows for verification. As models become more complex, the need for verification will grow.

There are various properties that need verification. First of all, the modeled process must be sound. Next, the process may be a redefinition of an already existing process. In that case, one would like the interface to an external party of the old and new process to be the same. Generally, one would like to compare different models of different processes and see how they behave w.r.t. certain important actions.

One can relabel the net, so that the unimportant actions have a null label (are unlabeled) and use *weak* bisimilarity. This is an equivalence relation between graphs that abstracts from null actions. Two graphs are *weakly bisimilar* iff there exists a weak bisimulation between them. A weak bisimulation differs from an ordinary (or strong) bisimulation by the fact that one no longer observes null actions. So the correspondence between states is weakened as only the non-null busy actions must correspond. One also cannot observe the difference between directly starting an action and starting it after having started and committed some null actions. On the other hand, starting or committing a null action may correspond to not doing anything at all. Finally, the states that are only reachable from the initial state by aborting a null action do not need to be related by a weak bisimulation.

In [8], an equivalence relation is defined for general labeled nets that amounts to weak bisimilarity when applied to workflow nets. Strong bisimilarity implies weak bisimilarity, so the fact that two nets are weakly bisimilar after relabeling with null labels is a property of their whole equivalence classes, i.e. the processes. The operators in the previous section are congruences for both weak and strong bisimilarity.

Weak bisimilarity allows to reformulate the soundness property. A process being sound is equivalent to it being weakly bisimilar to the net in Figure 7 after relabeling all its actions to null. This makes sense, as a sound process from its initial state performs any number of actions before arriving in its final state. As all actions become null, it must correspond to the process modeled in Figure 7.

Nets can be *reduced* modulo weak bisimilarity, e.g. by the reduction rules from [8]. Reduction yields a simpler net (less places, transitions and/or arcs) for which will be easier to verify whether it behaves as expected w.r.t. the important actions.

We give an example for our travel department example, illustrated in Figure 8.

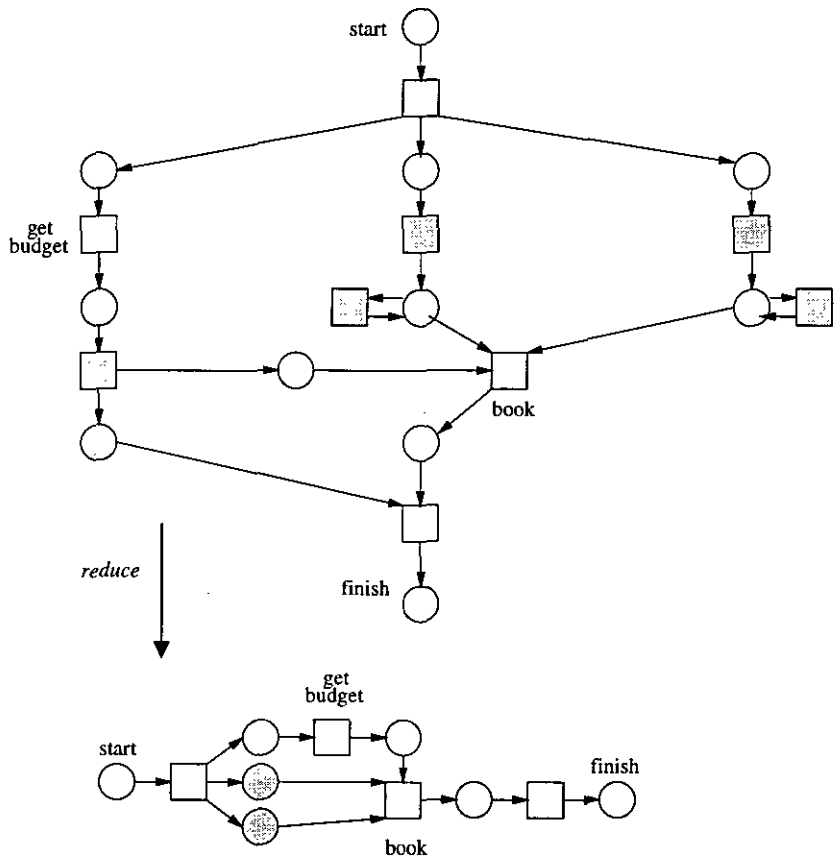


Figure 8: An example reduction

The requirement to verify, is that no trip can be booked without budget. The important actions for this requirement are **getbudget** and **book**. The top net in the figure shows the relabeling (or rather delabeling) result. The shaded place is *redundant* and can be removed. The shaded transitions are *inert* and can be removed after fusing their input and output places. The net thus reduced is shown below, where two more places are seen to be redundant. Transitions affecting initial and terminal places cannot be inert, so they are not removed. The stated requirement holds for the reduced net and thus also for the original one.

5 Conclusion and further work

The present paper proposes a semantics for workflow processes based on non-atomic actions. Actions have a duration and can be started, committed, rolled back and aborted. This dramatically increases the number of states when compared with atomic actions. However, when interfacing with a given workflow system, the above aspects of actions can become manifest. Most important, non-atomic actions allow refinement of processes and thus a hierarchical modeling strategy.

Labeled Petri (place-transition) nets are proposed as models for workflow processes. Models are equivalent if they are (strongly) bisimilar. Operators are defined for composing processes. The refinement operator allows a hierarchical approach to workflow modeling.

The increase in the number of states (due to actions being non-atomic), causes brute-force model checking verification to becomes less viable. Instead, a model can be verified *locally*, by examining small portions at a time, reducing it step by step. One can modify a workflow procedure and yet guarantee that it presents the same interface to its clients.

In [4], a non-atomic approach to actions has been proposed too. The ST bisimilarity defined there corresponds to ours in many respects. Markings comprise both transitions and places, and start and commit events are possible. There is also a notion corresponding to our abort event, which is essential for allowing refinement. The rollback possibility is new. Since actions are non-atomic and can be rolled back, the notions of weak bisimilarity [6] and branching bisimilarity [5] coincide.

We believe that the proposed semantics and equivalence are promising enough to justify further work. One direction is the development of reduction algorithms. Another one is theoretical, e.g. to investigate the kind of operators on labeled nets that strong/weak bisimilarity is a congruence for.

Acknowledgements

I wish to thank Wil van der Aalst for his help and advice.

References

1. W. van der Aalst. Verification of Workflow Nets. In *Application and Theory of Petri Nets 1997, 18th. International Conference, Proceedings*, volume 1248 of *Lecture Notes in Computer Science*, Toulouse, France, 1997. Springer–Verlag, Berlin, Germany.
2. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock and C. Ellis, editors, *Conf. on Organizational Computing Systems*, pages 10 – 21. ACM SIGOIS, ACM, Aug 1995. Milpitas, CA.
3. R.J. van Glabbeek and U. Goltz. Equivalence Notions for Concurrent Systems and Refinement of Actions. In A. Kreczmar and G. Mirkowska, editors, *Mathematical Foundations of Computer Science 1989, 14th. International Symposium, Proceedings*, volume 379 of *Lecture Notes in Computer Science*, pages 237–248. Springer–Verlag, Berlin, Germany, 1989.
4. R.J. van Glabbeek and F. Vaandrager. Petri Net Models for Algebraic Theories of Concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *PARLE: Parallel Architectures and Languages 1987, Vol II: Parallel Languages*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242, Eindhoven, Netherlands, 1987. Springer–Verlag, Berlin, Germany.
5. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
6. R. Milner. *Communication and Concurrency*. Prentice–Hall, London, UK, 1989.
7. W. Reisig. *Petri Nets*. Springer–Verlag, Berlin, Germany, 1985.
8. M. Voorhoeve. State Event Net Equivalence. Computing Science Reports 98/02, Eindhoven University of Technology, 1998. `http://wwwis.win.tue.nl/~wsinmarc/techreports.html`.
9. M. Voorhoeve and W. van der Aalst. Conservative Adaptation of Workflow. Computing Science Reports 96/24, Eindhoven University of Technology, 1996.
10. WFMC. Workflow Management Coalition Terminology and Glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, 1996.

Modeling Workflow Dynamic Changes Using Timed Hybrid Flow Nets

Clarence Ellis[1] and Karim Keddara[1] and Jacques Wainer[2]

[1] University of Colorado, CTRG Labs, Dept of Computer Science, Boulder CO 80309-0430, USA
[2] University of Campinas, Dept of Computer Science, Campinas Brazil

Abstract. Workflow Management Systems[26] are networked computer systems which enable the specification, analysis, coordination, and enactment of organizational procedures. Although some of these workflow management systems (we abbreviate to workflow systems) have been successful, many have failed to improve organizational processes. One reason for this failure is the dynamically changing nature of organizations and work which is not well supported by workflow systems.
In a previous paper[9], the authors defined notions of dynamic change in workflow systems by utilizing Petri net models. Some types of workflow change are safe, non-disruptive, and can be performed anytime. Other changes disturb ongoing transactions, and cause problems if they are attempted in a dynamic fashion. That previous paper also presented various definitions of dynamic change correctness. In this paper, we define the *timed flow nets* as a way to accommodate time issues into the design of workflow systems and the analysis of their structural changes. We also expand upon the issue of "safe" structural transformations which preserve the soundedness[20] properties. This paper introduces another new Petri net based model, the *timed hybrid flow net* model, that is suitable to address dynamic changes within workflow systems, their analysis. This model generalizes the notions of SCOC[9] and Extended SCOC[13].
This work is part of an ongoing research effort of the Collaboration Technology Research Group (CTRG) at the University of Colorado. Previous CTRG work introduced the many dimensions of workflow that can change, including process change, change of roles and actors, application data change, organizational structure change, and change of social structures.

1 Modeling Workflow Procedures

We assume the reader to have some basic understanding of the Petri net models, their firing semantics and basic properties including boundedness, safeness, liveness, and reachability graphs (the reader is refereed to [17, 15] otherwise.)

Many Petri-net based workflow models have been introduced in the literature[7, 8], but only few of them deal with time issues. Meeting commitments and deadlines has been identified for long as a key requirement in organization business models to achieve customer retention and expansion. Therefore, there is an urgent need to accommodate the temporal behavior of workflow systems. This

situation has been acknowledged but merely addressed by the WFMC[26]. On the other hand, many efforts have and are being put in place to address the issue in many other areas; including real time systems, communication protocols, process planning, work-force management systems etc...

In a previous work[9] (and similarly in [20],) workflow procedures are modeled by the so-called flow nets (workflow nets,) this modeling is carried out as follows: Activities that define the procedure are represented by transitions. Each transition has a label, a set of input places to mark the beginning of the modeled activity, and a set of output places to mark the end of the activity. The workflow procedure also specifies the order in which its activities ought to be carried out; activities may be mandatory or optional, they may be executed in sequence or in parallel. This partial ordering is modeled in the net by the so-called flow relation. Each flow net has a single entry place to reflect the start of the modeled procedure and a single exit place to mark the end of the modeled procedure.

Note 1. In the remainder of the paper, $\mathcal{AN}$ denotes a finite alphabet of *activity names*, $\mathcal{JN}$ denotes a finite alphabet of *job names*. $\mathcal{T}$ denotes the time domain each element of which is a non negative rational number, and $\mathcal{I}$ denotes the time interval domain. A time interval may be unbounded (e.g. $[2, \infty[$). $\mathcal{E} \subseteq (\mathcal{AN} \times \mathcal{T})$ denotes the alphabet of *event labels*. For a finite set A, A_{MS} denotes the class of multi-sets over A.

Definition 2. A **flow net** , $flow = (pSet, tSet, fRel, lab, s_{in}, s_{out})$ consists of:

- disjoint, finite and non empty sets $pSet$ of places and $tSet$ of transitions.
- the flow relation $fRel \subseteq (pSet \times tSet) \bigcup (tSet \times pSet)$ which is such that:

$$\underline{dom}(fRel) \bigcup \underline{ran}(fRel) = pSet \bigcup tSet$$
$$\forall t \in tSet,\ \exists p, p' \in PSet,\ [(t,p) \in fRel\ \&\ (p',t) \in fRel]$$

- the labelling function $lab : tSet \longrightarrow \mathcal{AN}$.

- $s_{in} \in pSet$ is the entry place, and $s_{out} \in pSet$ is the exit place, are such that:

$$\forall t \in tSet,\ [(t, s_{in}) \notin fRel\ \&\ (s_{out}, t) \notin fRel]$$

Moreover, $\underline{FNets}$ denotes the class of all flow nets.

Note 3. For $x \in pSet \bigcup tSet$, x is called an element of $flow$. The set of elements of $flow$ is denoted $\underline{Elem}(flow)$. The output set of an element x, denoted $\underline{out}_{flow}(x)$, is the set $\{y \mid (x,y) \in fRel\}$. The input set of x, denoted $\underline{inp}_{flow}(x)$, is the set $\{y \mid (y,x) \in fRel\}$. These notions are extended to sets in the usual manner. The subscript $flow$ will be dropped whenever it is clear from the context. The interface of $flow$, denoted $\underline{interface}(Flow)$, is the set $\{s_{in}, s_{out}\}$. $\underline{Interior}\ (flow)$ denotes the set of interior places (i.e. non bordering) of $flow$.

Since flow nets are Petri nets, some notions concerning Petri Nets carry over to flow nets. In particular, We use $\underline{1}$ (resp. $\overline{1}$ to denote the unique marking which

consists of a single token residing in the entry (resp. the exit) of a flow net. $\underline{Mark}(flow)$ denotes the class of all markings of $flow$.
For $m, m \in \underline{Mark}(flow)$ and $w \in tSet^*$, we write $m\,[flow\rangle\, m'$ to state that w is a firing sequence of $flow$ leading from m to m'. $\underline{Reach}(flow, m)$ denotes the class of all marking of $flow$ reachable from m. $mflow = (flow; m)$ is called a marked flow net.

In the remainder of the paper, we assume that flow nets do not use the symbol t_{exec}, be it as place, transition or label. $flow^*$ denotes the marked Petri Net obtained from $flow$, by adding a transition t_{exec} labeled t_{exec} connecting the exit place to the entry place and a single token in the entry place.

In [20], Van der Aalst introduces workflow nets and the notion of sound workflow nets. A flow net relaxes the strong connectivity property that a workflow net has to met due to our model of delayed change (to be explained later). Furthermore, the author shows that the soundedness property id decidable by linking it to the boundedness and liveness properties.

Definition 4. • $flow$ is **sound** iff the following conditions hold:

1. $flow^*$ is strongly connected.
2. $\forall m \in \underline{Reach}(flow, \underline{1}), \overline{1} \in \underline{Reach}(flow, m)$.
3. $mflow = (flow; \underline{1})$ has no dead transitions.

Adding time to flow nets

Different ways of accommodating time in Petri net models have been proposed by many researchers. These different proposals were influenced by the specific application domains, however there seem to be a commonly shared concern not to modify the basic behavior of the untimed model (parallelism and non determinism.) Three main-streams can be identified: Timing is associated with places [18], Timing is associated with transitions [16, 14, 27, 10, 22, 5] and Stochastic Petri Nets [2, 3]
Without claiming the superiority of any one with respect to others, we adopt the timed transition proposal as defined in [14, 22, 5]. Our choice is pragmatic and is driven by our concern to use a model which is in the middle of the complexity spectrum. It is well known that the modeling power of Timed Place Petri nets is equivalent to the limited modeling power of Timed Transition Petri nets with fixed durations. Although the analysis of Stochastic Petri Nets is possible under certain somewhat severe conditions, the behavior of these nets is better analyzed under simulation. This does not mean that Timed Transition Petri nets do not resist any kind of analysis. On the contrary, analysis is possible only if boundedness (in general undecidable, but always carries over from the underlying untimed net) holds [6] and this is the best result known to date (at least to us.) Luckily, boundedness is in general a well accepted requirement for workflow models. In our model, each transition will be associated with a *firing delay interval.* Formally,

Definition 5. A **timed flow net** , is a system $tflow =< flow, fdelay >$ which consists of:

- $flow \in \underline{FNets}$, the underlying flow of $tflow$.
- $fdelay : tSet \longrightarrow \mathcal{I}$, the firing delay function.

Moreover, $\underline{TFNets}$ denotes the class of all timed flow nets.

Note 6. For $t \in tSet$ and $[x, y] = fdelay(t)$, x is the early firing time of t and is denoted $\underline{ef\_time}\,(t)$, and y is the latest firing time of t and is denoted $\underline{lf\_time}\,(t)$.

Example 1. Consider an office procedure for order processing within a typical electronics company. When a customer requests by mail, or in person, an electronic part, this is the beginning of a job. A form is filled out by the order entry activity (abbreviated to *oe*) ; the job is sent to credit check Check (abbreviated to *cc*) , and then to inventory check (abbreviated to ic). After the evaluation (abbreviated to *ev*), the order is approved (abbreviated *ap*) and then sent to shipping (abbreviated *sh*) and billing (abbreviated to *bi*) and then to archiving (abbreviated to *ar*.) The shipping department will actually cause the part to be sent to the customer; the billing department will see that the customer is sent a bill, and that it is paid. Fig.1. depicts two versions of this procedure along, namely *oldNet* and *newNet*. The firing delays are expressed in minutes; for instance the firing delay of *oe* is between 2mn and 5mn. Whenever the activity labeling is injective, we will identify an activity with its label. The entry place of *oldNet* is p_0 and its exit place is p_6.

The question as to how to deal with marking extension has also given rise to at least two proposals. The original one [14, 6] extends the (untimed) marking with a set of dynamic firing intervals. The "new" current [22, 5, 10, 11] tends to lean toward the Coloured Petri net current[12]; a timed token contains time information (a time-stamp and/or a time interval) which in general is related to the creation of the token. In our model, a timed token is used to keep track of the creation and the availability time of a token. The creation time of a token is equal to the enabling time (see below for definition) of the activity which has produced the token. It does not have any bearing on the firing semantics, but it will become handy to carry out dynamic changes.

Definition 7. Let $tflow \in \underline{TFNets}$. A **timed token** over $tflow$ is a system tk, which consists of

- $loc \in pSet$, the location of tk.
- $c\_time \in \mathcal{T}$, the creation time-stamp of tk.
- $av\_time \in \mathcal{T}$, the availability time-stamp of tk.

Moreover, $\underline{Tks}\,(tflow)$ denotes the class of all tokens over $tflow$.

Definition 8. Let $tflow \in \underline{TFNets}$.
- A marking of $tflow$ is a distribution $m \subseteq \underline{Tks}\,(tflow)_{MS}$.
- $\underline{Mark}(tflow)$ denotes the class of all markings of $tflow$

Note 9. if m consists of a single token $tk = (s_{in}, 0, 0)$, then m is referred to as the *initial* marking of m and is denoted $\underline{1}_{tflow}$. Likewise, if m consists of a single token $tk = (s_{out}, \alpha, \tau)$, then m is referred to as a *terminal* marking. $\overline{1}_{tflow}$ denotes the class of all terminal markings. The subscript will be dropped whenever it is clear from the context.

We use a two-phase firing semantics:

wait phase: This phase begins the moment the activity is enabled and cannot go beyond the limits prescribed by the firing delay. During this phase, either the activity is disabled by the initiation of another (conflicting) activity or it *must* fire (*Strong Time Semantics or STS* [10].) However, the model can be extended to accommodate the *Weak Time Semantics or WTS* which relaxes the "*must* " into a "*may*. Traditionally, the WTS has being used in the modeling of *soft real-time systems*with soft deadlines whereas the STS has been applied for *hard real-time systems*in which deadlines are hard target to meet.

The wait phase may be necessary in some situations; for instance when the activity has to wait for some external events to happen (e.g. triggers), in this case a timeout may be set. Consider the case of a workflow specification which reads as " If the customer form arrives within 5 days, then activity A is executed, otherwise activity B". This situation can be easily modeled by using two conflicting transitions labeled A and B with the firing delay of A being $[0, 5]$ and the firing delay of B being $[5, 5]$.

firing phase: The activity fires by *consuming* one token from each of the input places and *producing* a new token in each output place. For the sake of simplicity, we assume that activity are instantaneous, however the model can be extended to include activities with fixed or variable duration.

The questions as to which tokens are selected for consumption, how to deal with *multiple enabledness* and *conflict resolution* are addressed within the context of the so-called *firing policy*. We choose (for the sake of simplicity) a policy based on *eager firing* with *infinite server* , *enabling memory* and *race-based conflict resolution*. The next definition formalizes the behavior of timed flow nets.

Definition 10. Let $tflow \in \underline{TFNets}$ and let $m \in \underline{Mark}(tflow)$

- An **event** over $tflow$, is a system $e = (tk_{in}, t, \tau)$, such that:

$$t \in tSet \;\&\; tk_{in} \subseteq \underline{Tks}\,(tflow) \;\&\; \underline{dom}(loc) = \underline{inp}(t)$$
$$loc\ is\ injective \;\&\; \tau \in (fdelay(t) + \underline{en\_time}\,(e))$$

where $\underline{en\_time}\,(e)$, called the enabling time e, denotes the the maximum availability time-stamp associated with the tokens of tk_{in}.
- $\underline{Evts}\,(tflow)$ denotes the class of all events over $tflow$.
- e is enabled under m, written $m\,[[e\rangle$, iff $b_{in} \subseteq m$.
- e is time enabled under m, written $m\,[e\rangle$, iff the following conditions hold:

$$m\,[[e\rangle \;\&\; \forall e' \in \underline{Evts}\,(tflow), m\,[[e'\rangle \Rightarrow \underline{en\_time}\,(e) \leq \underline{en\_time}\,(e')$$

In this case, the *enabling* of e under m *leads* to the marking m', written $m\,[e\rangle\,m'$, where

$$m' = (m - tk_{in}) \bigcup \{(s_{out}, \underline{en\_time}\,(e)\,, \tau) | s_{out} \in \underline{out}(t)\}$$

$\bullet \overset{tflow}{\longrightarrow} \subseteq \underline{Mark}(tflow) \times \underline{Evts}\,(tflow)^* \times \underline{Mark}(tflow)$ denotes the timed firing sequence relation associated with $tflow$ and is given by:

$(m, w, m') \in \overset{tflow}{\longrightarrow}$ iff $m = m'$ & $w = \lambda$ or the following condition is true:
$\exists m'' \in \underline{Mark}(tflow), \exists e \in \underline{Evts}\,(tflow)\,,$
$\left[w = w' \bullet e \;\&\; (m, w', m'') \in \overset{tflow}{\longrightarrow} \;\&\; m''\,[[e\rangle\,m'\right]$

Note 11. We will write $m\,[w\rangle_{tflow}\,m'$ instead of $(m, w, m') \in \overset{Flow}{\longrightarrow}$. $tflow$ will be dropped whenever it is clear from the context. w is called a (m, m')-timed firing sequence and the sequence w', obtained from w by dropping the information about consumed tokens, is called a (m, m')-firing sequence. $\underline{Fire}(tflow, m, m')$ will be used to denote the language of all (m, m') firing sequences.

These notions are lifted to the level of activity names. Thus, the sequence $w'' = lab(w')$ is called a (m, m')-labeled firing sequence. In particular, if $m = \underline{1}$ and $m' \in \overline{1}$, then w'' is called a schedule, the availability time-stamp of the terminal marking is called the completion time of sch and is denoted $\underline{cpl\_time}(sch)$. $\underline{LFire}(tflow, m, m')$ denotes the language of all (m, m')-labeled firing sequences, $\underline{Sched}(tflow)$ denotes the language of all schedules and $\underline{cpl\_time}(tflow)$ denotes the set of completion times of $tflow$.

Example 2. Consider the timed flow net *oldNet* introduced in Example 1 and the initial marking $m_0 = \underline{1} = \{tk_1 = (p_0, 0, 0)\}$.
Under m_0, the event $e_1 = (m_0, oe, 3)$ is enabled and $m_0\,[e_1\rangle\,m_1$ where $m_1 = \{tk_2 = (q_1, 0, 3), tk_3 = (q_2, 0, 3)\}$.
Under m_1, the events $e_2 = (tk_2, cc, 7)$ and $e_3 = (tk_3, ic, 5)$ are enabled, both have the same enabling time 3, so they can fire in any order. Thus, $m_1\,[e_2\rangle\,m_2\,[e_3\rangle\,m_3$ where $m_2 = \{tk_3, tk_4 = (q_3, 3, 7)\}$ and $m_3 = \{tk_4, tk_5 = (q_3, 3, 5)\}$.
Under m_3, the event $e_4 = (tk_4, tk_5, ev, 10)$ is enabled, its enabling time is 7, and $m_3\,[e_4\rangle\,m_5$ where $m_5 = \{tk_6 = (p_1, 7, 10)\}$.
Under m_5, the event $e_5 = (tk_6, ap, 13)$ is enabled and $m_5\,[e_5\rangle\,m_6$ where $m_6 = \{tk_7 = (p_3, 10, 13), tk_8 = (p_2, 10, 13)\}$.
Then, we could have $m_6\,[e_6\rangle\,m_7\,[e_7\rangle\,m_8\,[e_8\rangle\,m_9$, where $e_6 = (tk_7, sh, 15)$, $m_7 = \{tk_8, tk_9 = (p_5, 13, 15)\}$, $e_7 = (tk_8, bi, 16)$, $m_8 = \{tk_9, tk_{10} = (p_4, 13, 16)\}$, $e_8 = (tk_9, tk_{10}, ar, 20)$ and $m_9 = \{tk_{11} = (p_6, 16, 20)\}$
The sequence $sch = (oe, 3)(cc, 7)(ic, 5)(ev, 10)(ap, 13)(sh, 15)(bi, 16)(ar, 20)$ is a schedule, its completion time is 20mn. The completion time of *oldNet* is $[10, 34]$ it is the same as the completion time of *newNet* (we have readjusted the firing delays of *bi* in *newNet*.)

Definition 12. Let $tflow_1$ and $tflow_2$ be timed flow nets.

• $tflow_1$ is a **time approximation** of $tflow_2$, written $tflow_1 \sqsubseteq_T tflow_2$, iff $\underline{cpl\_time}(tflow_1) \subseteq \underline{cpl\_time}(tflow_2)$.

• $tflow_1$ is a **schedule approximation** of $tflow_2$, written $tflow_1 \sqsubseteq_S tflow_2$, iff $\underline{Sched}(tflow_1) \subseteq \underline{Sched}(tflow_2)$.

The execution proceeds by processing what is commonly known as a *job*. Each job has a name which uniquely identifies the job at any given time, a flow which identifies the workflow procedure which is operating upon the job. It also has a history of the event firings which have so-far taken place as part of the execution of the job. Actually, we keep track of the labels and times of the event firings for dynamic change analysis. In the sequel, we assume that jobs do not interfere with each other. This assumption is carried out using the so-called *copy rule*. A formal definition will be given in the next section.

Like in the case of (untimed) flow nets, the notion of soundedness carry over to timed flow nets and can be argued to be desirable for timed flow nets. Unfortunately, the nice decidability properties that flow nets enjoy break down for the timed flow nets. Indeed, these properties for the most part are linked to reachability analysis, and as we have previously mentioned timed flow nets resist in general any kind of reachability analysis.

Furthermore, the linkage to boundedness and liveness properties is broken (in fact, boundness and liveness are sufficient conditions but not necessary .) To see that, consider the timed flow net, $tflow_1$, depicted in Fig.2. Clearly, $tflow_1$ is not sound, there is a schedule whose underlying firing sequence is $t_1t_2t_3t_4t_5t_6$ which leads to the marking under which both the exit place s_{out} and the place p_8 are both marked. On the other hand, $tflow_1^*$ is live and 1-safe. To see the safeness, note that the only place which may not be 1-safe is p_8 (consider the untimed structure). However, note that the first iteration of $tflow_1^*$ will result in both s_{in} and p_8 marked, and that at the end of the nth iteration, one of the following things may occur:

1. if p_8 is not initially marked, then it will be marked with 1 token.
2. if p_8 is initially marked, then the token is either flushed (t_8t_9) or kept (t_8t_{10} or $t_1t_2t_3t_4t_7t_5t_6$).

The soundedness property do not carry over to timed flow nets from their underlying (untimed) flow nets. To see that consider the timed flow net $tflow_2$ depicted in Fig.2. Clearly, the underlying (untimed) flow net is sound, but the timed version is not. After firing t_1, (p_1, p_2) becomes a sort of home marking and (p_4, p_5) is not reachable.

2 Modeling Structural Change within Workflow Systems

We adapt the model of change from [9] to accommodate the temporal nature of the flow net and to analyze the change correctness on a job basis. Like in the previous work, we shall focus on a special type of workflow procedure change;

namely the *structural* change. Structural entails that the change is made to the structure of the procedure (as opposed to the data-value). A change is either *dynamic* or *static* with respect to a job; dynamic means that the change is applied while the job is in progress, otherwise if the change is applied before the job starts executing, then the change is static. Another classification could be made based on the scope of the change; if the change is applicable to a specific set of jobs (i.e. execution instances,) then the change is referred to as an *instance change*, otherwise it is said to be a *class change*. Examples of instance changes include exceptions. Re-engineering plans are in general considered as instance changes before the cut-off or roll-out date is reached and class changes onward. Critical changes such as fixing hard bugs or related to mission critical systems are considered as class changes.
In a nutshell, our model of structural change is driven by a well-defined discipline which makes its analysis more manageable. This discipline is articulated around the selection of the *change regions* and is based upon the principle of *change locality*.

The *old change region* , denoted $oldRegion$, contains all the activities of the old timed flow net, referred to herein as the *old net* and denoted $oldNet$, which are involved in the change (e.g. deleted, reorganized etc...). This means that when selecting the old region, places connected to these activities as well as the connecting edges are made part of the old region. The *new change region*, denoted $newRegion$, embodies the alterations that the old region undergoes as a result of the change. In order to make the analysis of the change more manageable, The scope of the change should be as much as possible limited to the change regions; this requirement is referred to as the the principle of change locality. In other words, the selection of the old change region minimizes its interaction with its context. This interaction is structurally maintained solely by the interface of the old region, and is reduced to tokens exchange; the context supplies tokens to the old region for consumption (through its input place) and consumes the tokens produced by the old change region in its output place. The old change region is said to be a *closed subnet* of the old net, written $\underline{closed}(oldRegion, oldNet)$.
After the change regions are selected properly, the replacement may take place, resulting in a new flow, referred to as the *new net* and denoted $newNet$. The new net is obtained from the old net by:

1. plugging the new change region into the old net by using the interface of the new change region as sockets.
2. removing all the elements of the old change region from the resulting flow net.

Note 13. We will write $newNet = oldNet\,[oldRegion \longrightarrow newRegion]$ to say that the timed flow net, $newNet$, is obtained from the timed flow net, $oldNet$, by applying the replacement mechanism as previously outlined. The pair $\delta = \langle oldRegion, newRegion \rangle$ will be referred to as a *replacement pair* and the tuple $repl = (oldNet, \delta, newNet)$ will be referred to as a *replacement step*.

Example 3. In the case or our order processing procedure, it has been decided to initiate *sh* and *bi* is sequence, instead of concurrently as it was previously done. Moreover, this change should not affect the completion times previously reached. The general consensus was to speed up *bi* by acquiring high end systems. The old and new change regions as well as the old and the new nets are depicted in Fig.1. The reader is asked to ignore the jumpers and their connectors for now; their meaning will be clear shortly.

The introduction of the replacement mechanism leads to the natural question as to whether timed flow nets properties are preserved. In the case of (untimed) flow nets, the question has been addressed in [20] for untimed workflow nets. We extend these results and we will enumerate some transformations which are "safe" with respect to soundedness. But first, we need to define the notion of k-embedding. Informally, *oldRegion* is k-embedded in *oldNet* ($1 \leq k$), written $\underline{embed}(k, oldRegion, oldNet)$, iff no more than k simultaneous "executions" of *oldRegion* are active at any given point. Note here that in this case the entry place of *oldRegion* is k-safe, but the converse does not generally hold.

Proposition 14. *Let* $repl = (oldNet, (oldRegion, newRegion), newNet)$ *be a replacement step such that* $\underline{embed}(1, oldRegion, oldNet)$

1. *if oldNet is sound, then oldRegion is sound.*
2. *if newRegion* $\sqsubseteq_T$ *oldRegion, then the following properties hold:*
   (a) $\underline{embed}(1, newRegion, newNet)$
   (b) $newNet \sqsubseteq_T oldNet$.
   (c) *if oldNet is sound and newRegion is sound, then newNet is sound.*

Proof. In what follows we will give a sketch of the proof.

Part1 $oldRegion^*$ is strongly connected and $(oldRegion; \underline{1})$ does not have dead transition, otherwise *oldNet* would not be sound. Note here that there exists a marking $m \in \underline{Reach}(oldNet, \underline{1})$ such that $oldRegion.s_{in}$ is marked under m (otherwise every transition of *oldRegion* would be dead in $(oldNet; \underline{1})$). Assume that *oldRegion* does not enjoy the clean the termination property, then there exists a marking $m' \in \underline{Reach}(oldRegion, \underline{1})$ such that $oldRegion.s_{out}$ and some $p \in oldRegion.pSet$ are marked. In order to get clean termination in *oldNet*, another token has to enter *oldRegion* to flush out the token in p, which means that 2 executions of *oldRegion* would be simultaneously in progress. Clearly, this violates the 1-embedding property.

Part2-a Without loss of generality, assume that the completion times of *oldRegion* and *newRegion* are time intervals. Let *newNet'* be the timed flow net obtained from *newNet* by replacing *newRegion* with an activity t_{new} such that $fdelay(t_{new}) = \underline{cpl\_time}(newRegion)$. Let *oldNet'* be the timed flow net obtained from *oldNet* by replacing *oldRegion* with an activity t_{old} such that $fdelay(t_{old}) = \underline{cpl\_time}(oldRegion)$. $\underline{embed}(1, newRegion, newNet)$ iff the distance between two consecutive firings of t_{new} is greater than $\underline{lf\_time}(t_{new})$ and $\underline{embed}(1, oldRegion, oldNet)$ iff the distance between two consecutive firings of

t_{old} is greater than $\underline{lf\_time}\,(t_{old})$. Since the latter holds and $\underline{lf\_time}\,(t_{new}) \leq \underline{lf\_time}\,(t_{old})$, the former holds too.

Part 2.b Using an induction on the length of the firing sequence in $newNet$ and a lengthy case analysis, show that the following property, which in essence states that the $newNet$ is externally marking weakly equivalent to $oldNet$, holds

$$\begin{array}{l}(\forall m \in \underline{Reach}(newNet, \underline{1})) \\ \quad (\exists m' \in \underline{Reach}(oldNet, \underline{1})) \\ \quad\quad [\forall tk\ (tk \in m\ \&\ loc(tk) \notin \underline{Interior}\,(oldRegion)] \Rightarrow \\ \quad\quad\quad [\exists tk'\ (tk' \in m'\ \&\ loc(tk) = loc(tk')\ \&\ av\_time(tk) = av\_time(tk'))]\end{array}$$

Let w be a firing sequence of $newNet$ which leads from $\underline{1}$ to a terminal marking of $newNet$. Without loss of generality, assume that that w contains at least 1 segment belonging to $newRegion$ (otherwise, w would be a valid sequence of $oldNet$ and the case is closed,) all these execution segments in $newRegion$ are ordered (consequence of 1-embedding). Each segment can be replaced by a segment in $oldRegion$ with exactly the same completion time, and the resulting sequence is valid in $oldNet$ due to the external marking weak equivalence.

Part2.c In this case, the soundedness of $newNet$ is equivalent to the soundedness of $newNet'$. Since the soundedness of $oldNet'$ (which is given by the soundedness of $oldNet$) implies the soundedness of $newNet'$, the result follows immediately.

Example 4. Note that for the case depicted in Fig.1, these results may be applied. Indeed, we have both $oldRegion \sqsubseteq_T newRegion$ and $newRegion \sqsubseteq_T oldRegion$; their completion time is $[4, 15]$. Thus, the soundedness property is preserved.

Next, we enumerate some transformations which preserve the soundedness property. These transformations, except $T0$, have been introduced and investigated by Van der Aalst in [20] for (untimed) workflow nets. Due to the space limitation, the figures will be omitted. The idea is to make sure that the time approximation property holds.

T0 *optimization* The firing delay window of an activity t is shrinked.

T1a *division*: An activity t is divided into two consecutive activities t_1 and t_2 such that:

$$fdelay(t_1) \bigcup fdealy(t_2) \subseteq fdealy(t).$$

T1b *aggregation*: The reverse of T1a with

$$fdelay(t) \subseteq fdelay(t_1) \bigcup fdealy(t_2).$$

T2a *specialization*: An activity t is replaced by two conditional activities t_1 and t_2 such that

$$fdelay(t_1) \subseteq fdealy(t)\ \&\ fdealy(t_2) \subseteq fdealy(t).$$

T2b *generalization*: The reverse of T2b with

$$fdelay(t) \subseteq fdelay(t_1)\ \&\ fdelay(t) \subseteq fdealy(t_2).$$

T3a : An activity t is replaced by two parallel activities t_1 t_2 such that

$$[max(\underline{ef\_time}(t_1), \underline{ef\_time}(t_2)), max(\underline{lf\_time}(t_1), \underline{lf\_time}(t_2))] \subseteq fdealy(t).$$

T3b: The reverse of T3b with

$$fdealy(t) \subseteq [max(\underline{ef\_time}(t_1), \underline{ef\_time}(t_2)), max(\underline{lf\_time}(t_1), \underline{lf\_time}(t_2))]$$

T4a : An activity t_1 is replaced by an iteration of $t2$ and such that either both t_1 and t_2 are immediate or

$$\underline{lf\_time}(t_1) = \underline{lf\_time}(t_2) = \infty \ \& \ \underline{ef\_time}(t_2) \leq \underline{ef\_time}(t_1).$$

T4b: The reverse of T3b with

$$fdealy(t_1) \subseteq fdealy(t_2).$$

Other rules such as sequentialization, parallelization and swapping can be derived from the previous ones.

Proposition 15. *The transformations above persevere soundedness.*

3 Modeling Dynamic Changes within Workflow Systems

The application of a change to an in-progress procedure raises the issue of the whereabouts of a job's work units after the change takes place. In [9] the authors have considered the following approaches to deal with this issue:

Flush-Change-Restart: cancel the job (Flush), make the change (Change) and resubmit the job to the new procedure for processing (Restart.) This kind of change, generally "safe" and *static* in nature, may be recommended to fix *hard bugs* in workflow systems or in mission critical systems. In some cases, it may not be cost effective (just to mention one of its down-sides); hours of work and almost finished products are lost.

Wait-Change: wait until the job reaches a safe state or is finished (Wait) and make the change (Change.) This solution may not be feasible; it may take some time before the system reaches the sought state. It is also inadequate to deal with punctual changes such as exceptions handling.

Change-Transfer: Work units associated with the job are transfered to the new procedure. The transfer may affect all or some of the wok units. The work units not affected by the transfer continue their progression in the old net as if the change never took place. The transfer can also be optimized to ensure "safeness". It is based upon the principle of change locality; the units of work evolving outside of the old change region remain unchanged in the new procedure. The work units bordering the old change region are moved to the interface of the new region. Some of the work units progressing inside of the old change region are moved to the new change region, others terminate their progression in the old

change region and are moved to the new region when they reach the exit place. Additionally, the connection to the old change region's entry place is severed so as not to allow in any new work unit. In [9], the authors have introduced the *Synthetic Cut-Over Change (SCOC)* to reflect the situation where no work unit can be safely transfered from the old change region. Heuristics have also been devised to determine cases where SCOC is a safe solution. The work has been expanded in an in-progress work [13] through the *Extended Synthetic Cut-Over Change (E-SCOC)* which reflects the case where the token transfer is partial.

Change-Jump: The Change-Transfer solution may unnecessarily delay a change. An improvement is to maintain the tokens of the old change region where they are and to set up jumpers which would allow these tokens to jump into the new change region.

A *jumper* is a *high level* box whose *in-lets* (i.e. input places) in the old change region and whose *out-lets* (i.e. output places) are in the new change region. It also has a *time expression* which associates with each tuple of input timed tokens a tuple of output timed tokens. The idea here is to be able to readjust the availability time-stamps of the tokens to achieve *time coherency* in the new change region. This is crucial since the firing semantics is driven by the availability time-stamps of the tokens.

The execution of the job resumes in a *hybrid timed flow net* where the old change region is linked, as long as it is active, to the new change region through jumpers, and to ensure connectivity, there is at least two jumpers linking the entry (exit) place of the old change region to the entry (exit) place of the new change region. The firing policy is modified to that these token jumps are triggered whenever possible.

Example 5. Fig.1. depicts a possible configuration with 5 jumpers, denoted J_i for $i = 0 \ldots 4$. their time expression, e_i, are defined as follows:

$$\begin{array}{l} e_0 := [p'_1 = p_1] \\ e_1 := [p'_2 = p_2] \\ e_2 := [p'_3 = p_2] \\ e_3 := [p'_4.av\_time = max(p_4.av\_time, p_5.av\_time)] \\ e_4 := [p'_5 = p_6] \end{array} \quad (1)$$

This means that when J_1 is used, the token in p_3 is destroyed and the token in p_2 is moved to p'_2. When J_2 is used, the token in p_5 is destroyed. When J_3 is used, the token with the minimal available time-stamp is destroyed and the other one is moved to p'_4. Finally, J_4 moves a token from p_6 to $p'5$.

When considering the reverse configuration where the jumpers are reversed; that is the roles of the in-lets and the out-lets are reversed. The new time expressions

become:

$$
\begin{aligned}
e_0 &:= [p_1 = p_1'] \\
e_1 &:= [p_2 = p_2' \;\&\; p_3 = p_2'] \\
e_2 &:= \begin{bmatrix} p_5 = p_3' \\ p_2.av\_time = p_3'.c\_time \end{bmatrix} \\
e_3 &:= \begin{bmatrix} p_5.av\_time = p_4'.c\_time \\ p_4.av\_time = p_4'.c\_time + 1 \end{bmatrix} \\
e_4 &:= [p_5' = p_6]
\end{aligned}
\tag{2}
$$

This means that J_1 splits the token in p_2' in two. j_2 moves the token to p_3' and creates a new one. The same holds for J_3 with respect to p_4'. J_4 is a token mover.

Example 6. Let Job_1 be a job which happens to be running on the old order processing procedure and assume that its state consists of two tokens $tk_8 = (p_2, 10, 13)$ and $tk_9 = (p_5, 13, 15)$. After the change is made, the jumper J_2 is used. tk_8 is destroyed and tk_9 is moved to p_3'. The new marking is $tk_{12} = (p_3', 13, 15)$.

Example 7. Assume that after an audit, the company finds out that the change decision was not such a "good idea" and decides to undo the change. Let Job_2 be a job running on the new order processing, with a state consisting of a tk_{12}. After the change is undone, the jumper J_2 is active and should be used. After the jump takes place, tk_9 and $tk_{13} = (p_2, 0, 13)$ are created (0 is the default creation time-stamp). Except for the creation time-stamp of tk_{13}, the marking in the old net is valid. This may be tolerable, because the creation time-stamp has no bearing on the firing semantics. However, this is problematic if another change takes place right before tk_{13} is used. A solution would be to mark the token as *forbidden from jumping.*

In the remainder of this section, we introduce formally jumpers and timed hybrid flow nets. Due to space limitations, we do not formalize their firing semantics.

Definition 16. Let $tflow_1, tflow_2$ be disjoint timed flow nets.
A $(tflow_1, tflow_2)$-**jumper** is is a system $jmp = (inLet, outLet, texpr)$ which consists of:

- finite and disjoint sets inLet of *input sockets* and outLet of *output sockets* of the jumper.

- $texpr$, called the *time expression* of the jumper, such that:

$$
\begin{aligned}
&inLet \subset tflow_1.pSet \\
&outLet \subset tflow_2.pSet \\
&texpr \subseteq (inLet \times \mathcal{T} \times \mathcal{T}) \times (outLet \times \mathcal{T} \times \mathcal{T})
\end{aligned}
$$

Moreover, $Jumps(tflow_1, tflow_2)$ denotes the class of all $(tflow_1, tflow_2)$-jumpers.

Note 17. We shall be interested in two particular jumpers, namely the *entry and exit jumpers*. These are jumpers which move tokens from the entry (exit) place of $tflow_1$ to the entry (exit) place of the $tflow_2$ without modifying the tokens time-stamps. They will be denoted respectively $\underline{entry\_jmp}(tflow_1, tflow_2)$ and $\underline{exit\_jmp}(tflow_1, tflow_2)$ and $\underline{interface\_jmps}(tflow_1, tflow_2)$ will denote the set containing both of these jumpers.

The timed hybrid flow nets are introduced to accommodate the dynamic change as outlined earlier. Each timed hybrid flow net has an (ordered) sequence of constituents, a root and set of jumpers. Each constituent is a timed flow net which represents an old change region in a previously carried out dynamic change. Since the changes are carried out in an orderly manner, the constituents are ordered. The root is also a timed flow net and represents the latest version (i.e. the new net of the latest change.) This means that no jumper has an input socket in the root.

Definition 18. A **timed hybrid flow net**, $thflow$, consists of:

- a nonempty sequence $tflows$ of pairwise disjoint timed flow nets, called *constituents*.
- a timed flow net,$root$, the *root*.
- $jmpers$, a set of pairwise disjoint jumpers such that:

$$\begin{array}{l} \forall jmp \in jumpers, \exists i < j \ [jmp \in Jumps(tflows[i], tflows[j])] \\ \forall jmp \in jumpers, [jmp.inLet \bigcap \underline{Elem}(root) = \emptyset] \end{array} \quad (3)$$

Moreover, $\underline{THFNets}$ denotes the class of all timed hybrid flow nets.

Note 19. The notion of marking carries over to timed hybrid nets; each marking is the sum of the root and the constituents markings. Events firing occur within the boundary of a constituent or the root as formalized in the previous section, or across them using the jumpers as described earlier in this section. Furthermore, we assume that jumping occurs whenever possible.

Definition 20. A **job** is a system job where:

- $name \in \mathcal{JN}$, the name of job.
- $thflow \in \underline{THFNets}$, the flow of job.
- $state \in \underline{Mark}(thflow)$, the state of job.
- $hist \in \mathcal{E}^*$, the history of job.

Moreover, $\underline{Jobs}$ denotes the class of all jobs.

Definition 21. A **dynamic change** is a system, $change = (oldJob, newJob, repl)$, which consists of:

- $oldJob \in \underline{Jobs}$, the *old job*.

– $newJob \in \underline{Jobs}$, the *new job*.

– $repl = (oldR, newR)$, the *replacement pair*

which are such that:

$$newJob.name = oldJob.name \tag{4}$$
$$newJob.state = oldJob.state \tag{5}$$
$$newJob.hist = oldJob.hist \tag{6}$$
$$newRoot = oldRoot\,[oldR \longrightarrow newR] \tag{7}$$
$$newConst = oldConst \bullet oldR \tag{8}$$
$$newJmps \supseteq oldJmps \tag{9}$$
$$newJmps \supseteq \underline{interface\_jmps}(oldR, newR) \tag{10}$$
$$newJmps - oldJmps \subseteq Jumps(oldR, newR) \tag{11}$$

where

$n = mflows_1.length$
$newRoot = newJob.thflow.root$
$newConst = newJob.thflow.constituents$
$newJumps = newJob.thflow.jumpers$
$oldRoot = oldJob.thflow.root$
$oldConst = oldJob.thflow.constituentsoldJumps = oldJob.thflow.jumpers$

In the last definition, conditions 8-10 state that the name, state and history information carry over to the new job. Condition 11 ensures that the root of the new job is the last version of the procedure. Condition 12 appends the old region to the old sequence of the constituents. Conditions 13-15 reflects the possibility of setting up jumpers from the old to the new change region; at least the entry and the exit jumpers are added to the previous list of jumpers. In particular, if only these jumpers are added then, the change is referred to as a **synthetic cut-over change**.

To conclude this section, we would like to report that we are able to extend the results from [9] to the new timed model of workflow procedures and dynamic changes. In particular, if the new change region is a schedule approximation of the old change region, then the synthetic cut-over change is correct when a history-based correctness is adopted. Other results concerning change composition and iteration are under investigation.

4 Related Work

Recently, the problem of workflow structural change has been the focus of numerous work efforts, but none of these efforts consider the time issues. Thus, our comparison will be done with respect to the untimed hybrid flow nets.
In [4], the authors introduce a class of high level Petri nets, called reconfigurable

nets, which dynamically modify their own structure. As far as dynamic change is concerned, the reconfigurable nets can be used to emulate synthetic cut over changes but fails in general to emulate jumpers. The reason is that the model does not allow the creation nor the disappearing of tokens, but only token movements. On the other hand, reconfigurable nets are better suitable than hybrid flow nets to support multiple *modes of operation.*

In [1], the authors are independently adopting a methodology similar to flow jumpers. Their dynamic correctness revolves around the notion of *safe state* w.r.t. a change. According to their model, a dynamic change occurs only if the state reached by a job (in the old procedure) is safe w.r.t. the change. To comply with this requirement, they propose *linear jumpers* as means to "force" a job into a safe state (in the old procedure.) There seems to be at least one fundamental difference in our respective approaches. Their model accommodates *retroactive changes*, in the sense that in some cases, getting to a safe state may require undoing some of the activities which have taken place. This gives rise to the not so trivial issue of the *undo semantics.*

In [25], the issue of workflow flexibility is addressed. The authors introduce *ad-hoc workflows* based on *process templates.* These process templates are considered as *reference models.* They also give a set of static structural transformations which may be used to build safe and successfully terminating workflow nets starting from a library of basic process templates which enjoy these properties.

In [24], the authors define process equivalence based on *delay bisimilarity.* Similarity between cases (i.e. jobs) is *conserved* by considering them as *extensions* or *reductions* of the same ancestor. In the event that a change results in a process extension, the change can be applied dynamically without delay to a running job. However, no mention is made if the change results in a process reduction.

5 Conclusions and Summary

Dynamic structural change to office procedures is a pervasive unsolved problem within workflow environments. This paper has introduced the timed flow nets as a way of accommodating time issues into the design of workflow systems and the analysis of their static changes. It has also expanded on the issue of "safe" static transformations which preserve the soundedness properties.

This work has also briefly presented a new Petri-net based model, namely the timed hybrid flow nets, that is especially suitable to address workflow dynamic changes . In a companion technical report to this paper, we formally define timed hybrid flow nets, their semantics and their application to the problems of dynamic change. We also expand upon the results from [9], state and establish results concerning the dynamic change composition and iteration.

The issue of dynamic change correctness is currently under investigation in a broader context than in [9, 20]. This effort is concerned with the design and implementation of SL-DEWS, a specification language for the dynamic evolution of workflow systems. We hope that by the time of the 1998 Petri net conference in Portugal, we will have interesting results to report on SL-DEWS.

Acknowledgements We wish to thank the anonymous referees for their helpful comments which helped to improve the quality of this work and for pointing out some related references.

References

1. A. Agostini, F. De Michelis. "Simple workflow models" In Proceedings of WFM98: Workflow Management: Net-Based Concepts, Models, Techniques and Tools, PN98, Lisbon, Portugal.
2. M. Ajmone Marsna, G. Balbo, A. Bobbio, C. Chiola, G. Conte, A. Cumani. "On Petri Nets with Stochastic Timing" In Proc. of the International Workshop on Timed Petri Nets, Torino, 1985, IEEE Computer Society Press.
3. M. Ajmone Marsna, G. Balbo, G. Conte. "A Class of Generalized Stochastic Petri Petri Nets for the Performance Evaluation of Multiprocessor Systems" ACM Transactions on Computer Systems, 2 (1984).
4. E. Badouel, J. Oliver. "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynmaic Changes" In Proceedings of WFM98: Workflow Management: Net-Based Concepts, Models, Techniques and Tools, PN98, Lisbon, Portugal.
5. G. Berthelot, H. Boucheneb. " Occurence Graphs For Interval Timed Coloured Nets" Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science, volume 815, Springer-Verlag, 1994.
6. B. Berthomieu, M. Diaz. " Modeling and verification of time depenedent systems using time Petri nets" IEEE Transactions on Software Engineering, vol. 17, No 3, March 1991.
7. G. De Michelis, and Ellis, C.A. Computer Supported Cooperative Work and Petri Nets. Third Advanced Course on Petri Nets, Dagstuhl Castle, Germany (1996). Springer Verlag Lecture Notes in Computer Science.
8. A. Ellis and Nutt, G.J. "Modeling and Enactment of Workflow Systems". In M. Ajmone Marsan, editor, Application and Theory of Petri Nets 1993, volume 691 of Lecture Notes in Computer Science, pages 1-16. Springer-Verlag, Berlin, 1993.
9. C.A. Ellis, Keddara, K and Rozenberg, G. "Dynamic Change within Workflow Systems". Proceedings of the Conference on organizational Computing systems, ACM Press, New York (1995) 10-21.
10. C. Guezzi, D. Mandrioli, S. Morasca, P. Mauro. "A general way to put time into Petri nets" In Proc. of the Fifth International Workshop on Software Specification, Vol. 14-3 of ACM SIGSOFT Engineering Notes, Pittsburg, Pennsylvania, USA, 1989.
11. C. Guezzi, S. Morasca, M. Pezze. "Validating Timing Requirements for TB Net Specifications" The Journal of Systems and Software, vo. 27, No 7, November 1994.
12. K. Jensen. "Coloured Petri Nets: Basic concepts, Analysis Methods and Practical use. volume 1: Basic Concepts". EATCS Monographs on Theoretical Computer Science, Springer-Verlag 1992.
13. K. Keddara. "On the Dynamic Evolution of Workflow Systems" Ph.D. Thesis in preparation.
14. P. Merlin, D.J. Farber. "Recoverability of communication protocols". IEEE Transactions on Communications, 24, 1976.
15. T. Murata. "Petri nets: properties, analysis, and applications. Proceedings of the IEEE 77(4), 1989.

16. C. Ramchandani "Analysis of Asynchroneous Concurrent Systems by Timed Petri Nets" Project MAC, TR 120, MIT, 1974
17. W. Reisig. "Petri Nets". Springger 1985.
18. J. Sifakis. "Use of Petri Nets for Performance Evaluation". Measuring, Modeling and Evaluating Computer Systems, H. Beilnerand E. Gelenbe editors, North Holland, 1977
19. Saastamoinen "On the Handling of Exceptions in Information Systems" University of Jyvaskyla PhD Dissertation, Nov. 1995.
20. W.M.P. van der Aalst. "Verification of Workflow Nets". In P. Azema and G. Balbo, editors, Application and Theory of Petri Nets 1997,volume 1248 of Lecture Notes in Computer Science, pages 407-426. Springer-Verlag, Berlin, 1997.
21. W.M.P. van der Aalst. "Finding Erros in the Design of a Workflow Process". In Proceedings of WFM98: Workflow Management: Net-Based Concepts, Models, Techniques and Tools, PN98, Lisbon, Portugal.
22. W.M.P. van der Aalst." Interval Timed Colored Petri Nets and their Analysis". Application and Theory of Petri Nets 1993, 14th International Conference, Chicago, Illinois, USA, LNCS 691, Springer-Verlag.
23. R. Vlak. "Self-Modifying Nets, a Natural Extension of Petri Nets". Proceedings of Icalp'78, Lecture Notes in Computer Science vol.62 (1978) 464-476.
24. M. Voorhoeve, W.M.P. Van der Aalst. "Conservative Adaption of Workflow" In M. Wolf and U. Reimer, editors, Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow, Basel, Switzerland, 1996
25. M. Voorhoeve, W.M.P. Van der Aalst. "Ad-hoc Workflow: Problems and Solutions" In R. Wagner, editor, Proceedings of the 8th DEXA Conference on Database and Expert Systems Applications, Toulouse, France, 1997.
26. WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011) Technical Report, Workflow Management Coalition, Brussels, 1996.
27. W. Zuberek. "Timed Petri nets and preliminary performance evaluation". In Proc. 7th Annual Symposium on Computer Architecture, La Baule, France.

This article was processed using the LaTeX macro package with LLNCS style

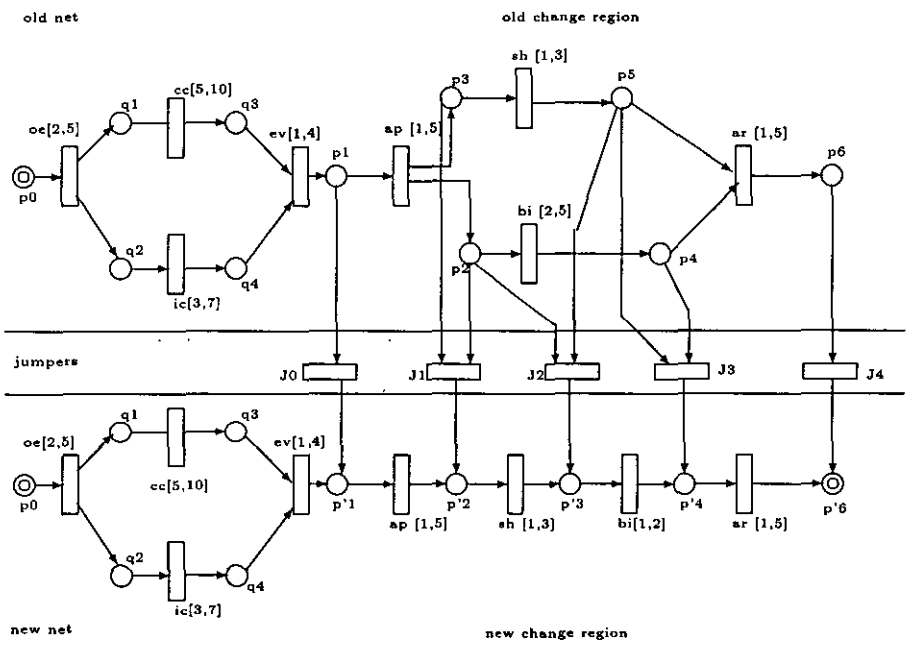

old net
old change region
sh [1,3]
p3
p5
cc[5,10]
q1
q3
oe[2,5]
ev[1,4]
ap [1,5]
ar [1,5]
p6
p1
p0
bi [2,5]
q2
p2
p4
q4
ic[3,7]
jumpers
J0
J1
J2
J3
J4
q1
q3
oe[2,5]
ev[1,4]
p0
cc[5,10]
p'1
p'2
p'3
p'4
p'6
q2
ap [1,5]
sh [1,3]
bi[1,2]
ar [1,5]
q4
ic[3,7]
new net
new change region

Fig. 1.

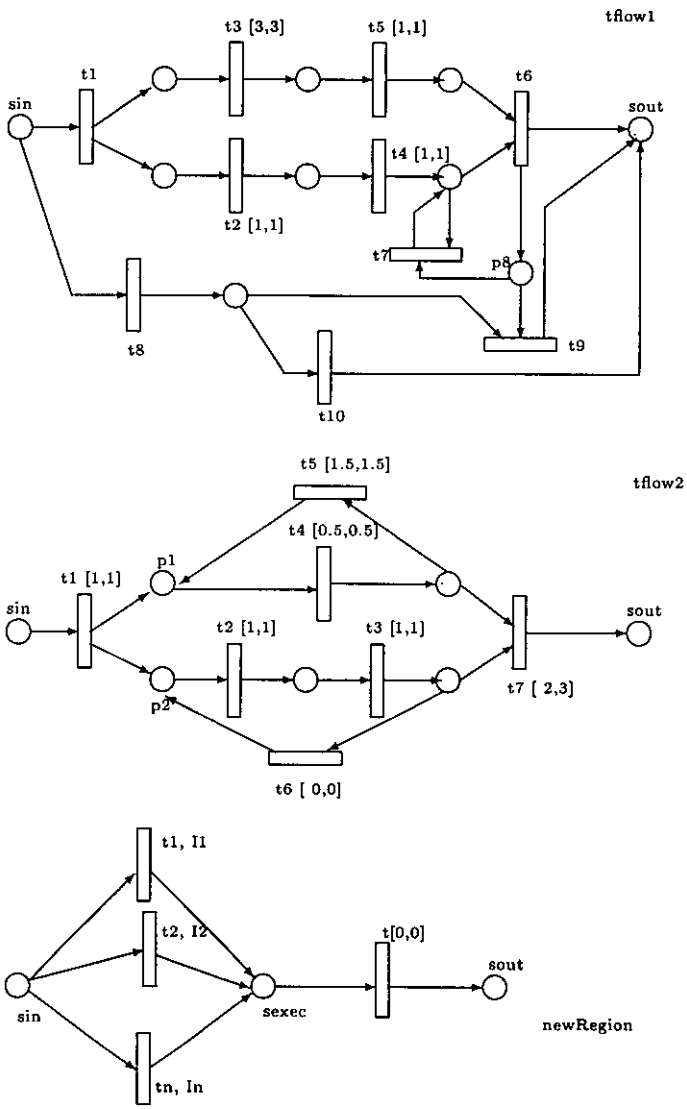

tflow1
t3 [3,3]
t5 [1,1]
t1
t6
sin
sout
t4 [1,1]
t2 [1,1]
t7
p8
t8
t9
t10
t5 [1.5,1.5]
tflow2
t4 [0.5,0.5]
p1
t1 [1,1]
sin
t2 [1,1]
t3 [1,1]
sout
t7 [2,3]
p2
t6 [0,0]
t1, I1
t2, I2
t[0,0]
sout
sin
sexec
newRegion
tn, In

Fig. 2.

Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes*

Eric Badouel[§] and Javier Oliver[¶]

[§] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France.
E-mail: Eric.Badouel@irisa.fr.

[¶] DSIC, UPV, Camino de Vera s/n, 46071 Valencia, Spain.
E-mail: fjoliver@dsic.upv.es.

Abstract. We introduce a class of high level Petri nets, called reconfigurable nets, that can dynamically modify their own structure by rewriting some of their components. Boundedness of a reconfigurable net can be decided by constructing its coverability tree. Moreover such a net can be simulated by a self-modifying Petri net. The class of reconfigurable nets thus provide a subclass of self-modifying Petri nets for which boundedness can be decided. Delayed dynamic changes within workflow systems in the sense of [8] can then be handled in an extension of van der Aalst's workflow nets [2]. For this class (the *reconfigurable workflow nets*), a notion of soundness has been defined that can also be verified using the coverability tree construction.

Keywords: Reconfigurable Nets, Workflow Systems, Boundedness, Self-Modifying Petri Nets.

1 Introduction

Since their introduction in the early sixties [17], Petri nets have come to play a pre-eminent role in the formal study of the behaviour of concurrent and distributed systems. Let us mention some of the attractive features that have made this model successful. First of all, like vector addition systems or commutative semi Thue systems, Petri nets are a very simple and natural extension of automata. Therefore the study of their mathematical properties becomes a manageable task; in particular much effort have been devoted to decidability and complexity issues for Petri nets. Second, a lot of techniques and automated tools support the verification of properties of systems modelled by Petri nets. For instance one can decide by constructing its coverability tree whether a Petri net is bounded, i.e. whether it is a finite state system. Reduction techniques and linear algebra techniques have also received wide attention. Third, Petri net is a graphical tool that can easily be used for the description and the design of concurrent systems.

* This work was partially supported by the H.C.M. Network *Express* and by CICYT, TIC 95-0433-C03-03.

Recently, Petri nets have been used for modelling *Computer Supported Co-operative Work* (CSCW) [9, 5]. These applications, also called *groupware* applications, involve distributed systems of agents (computer systems or humans) which cooperate to solve a global task and whose structure can dynamically change. More precisely, we consider in this paper *workflow systems*. These systems aim to support the realization of work procedures by a group of collaborating agents by coordinating the flow of tasks within the distributed system. As in [2] we define a workflow net as a Petri net with a specific input place and a specific output place. A token in the input place corresponds to a new *case* entering the system, the structure of the Petri net describe the set of *tasks* required to process this case and the order in which these tasks can be executed (taking the distributed nature of the system into account). Finally a token in the output place witnesses the termination of the case. An important feature of these workflow systems is their ability to manage dynamic change: the structure of the Petri net should be allowed to vary as a case proceeds within the system. Petri nets however do not offer a direct way to express processes whose structure evolves along computations. For that reason they have been used in conjunction with sets of rewriting rules in order to cope with workflow systems: such a system is locally described by a Petri net while rewriting rules allow for the modification of the structure of the Petri net. The purpose of this paper is to introduce *reconfigurable nets* which is a class of high level Petri nets that can dynamically modify their own structures by rewriting some of their components thus supporting dynamic changes within workflow systems. Reconfigurable nets is a very natural extension of Petri nets, we therefore have confidence that many of the theoretical results and automated tools that exist for Petri nets could be used or adapted for them. For instance, we show in this paper that we can decide the boundedness property for reconfigurable nets using a variant of coverability trees.

The Dynamic nets of Asperti and Busi [1] is also a model that allows for dynamic changes. In Dynamic nets tokens are names for places, an input token of a transition can be used in its postset to specify a destination, and moreover the creation of new nets during the firing of a transition is also possible. The work of Asperti and Busi recasts in the context of net theory ideas and concepts that originated in the π-calculus [15] and the related join-calculus [10, 11]. It is our opinion that the intricacy of this model leaves little hope to obtain significant mathematical results and/or automated verification tools in a close future. Moreover this model is probably too sophisticated to be easily used for the modelling and design of groupware applications.

Self-modifying nets introduced by Valk [18, 19] and their subclass of *stratified Petri nets* [3] is another extension of Petri nets. Dynamicity is introduced there via self-modification. More precisely the flow relations between places and transitions in self-modifying nets are linear functions of the marking. Techniques of linear algebra used in the study of the structural properties of Petri nets can be adapted to this extended framework; in particular each transition may be associated with a matrix and the modification of the marking due to a sequence

of firable transitions can be coded by the corresponding product of matrices. Even though self-modifying nets constitute a small variation in the Petri net paradigm, some important properties are lost, e.g. we cannot decide the boundedness of self-modifying nets. Moreover even if this class of nets has a clean and concise definition it is hopeless to describe and design realistic systems directly in this formalism.

It is our opinion that self-modifying nets is a very reasonable attempt to add mobility in Petri nets but that they should be used rather as a back-end model. Reconfigurable nets on the other hand constitute a more direct formalisation of the manner in which groupware applications are described using a combination of Petri nets and rewriting rules as in [8]; moreover we describe in this paper a translation of reconfigurable nets into equivalent self-modifying nets. Therefore reconfigurable nets can be viewed as a subclass of self-modifying nets for which boundedness can be decided.

The rest of the paper is organized as follows. Reconfigurable nets are introduced in Section (2) and we indicate how they can be used for modelling workflow nets with a dynamic structure, the so-called reconfigurable workflow nets. We show in Section (3) that we can decide whether a reconfigurable net is bounded by constructing its coverability tree, and that the soundness of a reconfigurable workflow net implies its boundedness and can also be verified using the coverability tree. We also prove an analogue of a result of [2] showing that the soundness of a reconfigurable workflow net reduces to the boundedness and liveness of the reconfigurable net obtained by adding an extra transition connecting the output place of the reconfigurable workflow net to its input place. In Section (4) we show that any reconfigurable net can be simulated by a self-modifying net. Finally we conclude in Section (5).

2 Reconfigurable Nets

Reconfigurable nets are high level Petri nets supporting dynamic changes within workflow systems. For instance one can admit local changes in the scheduling of the tasks required to process a case which is currently flowing in the system. If the case is an order request from a customer the involved tasks may be Order Check, Inventory Check, Credit Check, Shipping, Billing, and Archiving [8]. A dynamic change may then enable the parallel execution of two tasks (e.g. Shipping and Billing) that were previously performed sequentially in some order; it may also refine some task into more elementary tasks with a prescribed ordering. We assume however that the set of involved tasks as well as the set of local changes are known in advance and can be listed. This assumption implies that the set of *tasks instances* is finite. This set constitutes the set of transitions of the reconfigurable net. The switching from one configuration to another one due to a local change is taken care of by the introduction of a new kind of place content which denotes whether a place does exist or not in the current state of the system.

Definition 1 *A reconfigurable net is a structure* $N = (P, T, F, R)$ *where* $P = \{p_1, \ldots, p_m\}$ *is a non empty and finite set of places,* $T = \{t_1, \ldots, t_n\}$ *is a non empty and finite set of transitions disjoint from* P $(P \cap T = \emptyset)$, $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is a weighted flow relation, and* $R = \{r_1, \ldots, r_k\}$ *is a finite set of structure modifying rules. A structure modifying rule is a map* $r : P_1 \to P_2$ *whose domain and codomain are disjoint subsets of places* $(P_1, P_2 \subseteq P$ *and* $P_1 \cap P_2 = \emptyset)$. *A marking of net* N *is a map* $M : P \to \mathbb{N} \cup \{\alpha\}$ *where* $\alpha \notin \mathbb{N}$, *when* $M(p) = \alpha$ *place* p *is said not to exist in marking* M *whereas* $M(p) = n \in \mathbb{N}$ *expresses that* p *exists in marking* M *and has value* n. *We let* $\mathcal{M}$ *denote the set of markings of net* N. *We let* $E = T \cup R$ *denote the set of events of the reconfigurable net. We let* $M[e>M'$ *denote the fact that event* e *is enabled in marking* M *and that the net reaches marking* M' *when firing this event. This transition relation is defined as follows. A transition* $t \in T$ *is enabled in marking* M *if:*

$$\forall p \in P \qquad M(p) \neq \alpha \Rightarrow M(p) \geq F(p,t)$$

When transition t *is fired in marking* M, *the resulting marking* $M[t>M'$ *is such that* $\forall p \in P$

$$M(p) = \alpha \Rightarrow M'(p) = \alpha$$
$$M(p) \neq \alpha \Rightarrow M'(p) = M(p) - F(p,t) + F(t,p)$$

A structure modifying rule $r \in R$ *is enabled in marking* M *if:*

$$\forall p \in P_1 \quad M(p) \neq \alpha$$
$$\forall p \in P_2 \quad M(p) = \alpha$$

The firing of this enabled rule r *produces the new marking* M' *defined as:*

$$\begin{array}{ll} \forall p \in P_1 & M'(p) = \alpha \\ \forall p \in P_2 & M'(p) = \sum\{M(q) \mid q \in P_1 \ \wedge \ r(q) = p\} \\ \forall p \in P \backslash (P_1 \cup P_2) & M'(p) = M(p) \end{array}$$

A marked reconfigurable net is a reconfigurable net together with an initial marking.

The firing policy of transitions is like in the Petri net obtained by discarding the non existing places. This Petri net is called a *configuration* of the reconfigurable net. As long as no structure modifying rule take place, the reconfigurable net behaves exactly like this Petri net. Structure modifying rules produce a structure change in the net by removing existing places and creating new ones, thus moving the system from one configuration to another one. When a place is removed, the tokens of this place do not disappear, but they are moved to other places of the net. Hence, the number of tokens remains constant through the application of structure modifying rules. The rule defines how tokens should be moved in the net. Places of set P_2 which are not in the range of r are places created by the structure modifying rule and containing initially no token. Roughly speaking a reconfigurable net can be seen as a bunch of Petri nets (its configurations) which correspond to the various *modes* of operation of the system. The structure

modifying rules allow to switch from one mode of operation to another one while it modifies the current marking accordingly: work cases are not processed during the firing of a structure modifying rule, therefore there is no creation nor disappearing of tokens, these tokens are simply displaced from vanishing places to created ones. Thus a system modelled by a reconfigurable net has the ability of dynamically change its own structure when certain conditions are met. For instance if the content of some place becomes too large (there is a large amount of work cases waiting for being processed) one can duplicate the ouput transitions of this place, technically we replace this place by a new one having twice as many output transitions playing the same role as the output transitions of the original place. Using reconfigurable nets one can also easily implement the *delayed dynamic changes* of [8]. These dynamic structural changes also called *synthetic cut-over changes* are defined as follows, quoted from [8].

> *[in a synthetic cut-over change] both the old and the new change regions are maintained in the new procedure. This ensures that tokens already in the old change region will continue their progression as if the change did not take place immediately (which justifies the attribute delayed). However tokens evolving in the context of the old change region will never enter the old change region (but possibly new change region); that is to say that in view of these tokens the change is immediate.*

We can illustrate synthetic cut-over change with the example of Fig. 1. This

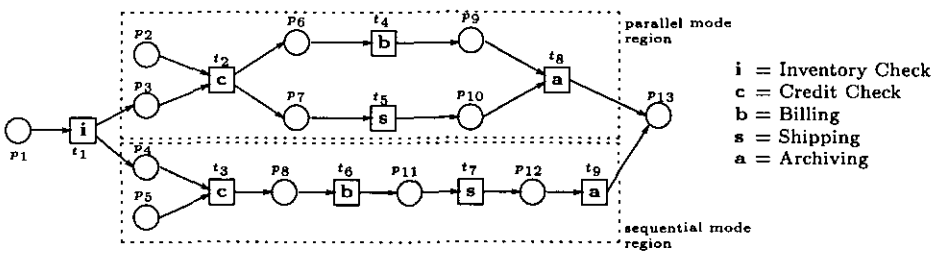


Fig. 1. synthetic cut-over change

reconfigurable net describes how to proceed an order request from a customer, there are two modes of operation corresponding to distinct regions in the graphical representation of the net, one in which the Billing and Shipping operations are processed sequentially and the other in which they are processed in parallel. The structure modifying rule $r : \{p_4; p_2\} \rightarrow \{p_3; p_5\}$ given by $r(p_4) = p_3$ and $r(p_2) = p_5$ permits to switch from the sequential mode of operation to the parallel mode of operation. Conversely the structure modifying rule r^{-1} realizes the switching in the converse direction. Figure 2 represents a fragment of the marking graph of this reconfigurable net, a place is graphically represented in a

given state if and only if that place exists in the current marking (i.e. its value is different from α). Observe that when we switch from the sequential mode to the parallel mode the tokens which are in the old region (sequential mode region) *continue their progression as if the change did not take place* but the tokens in the context (in place p_1) will now enter the new region (the parallel mode region).

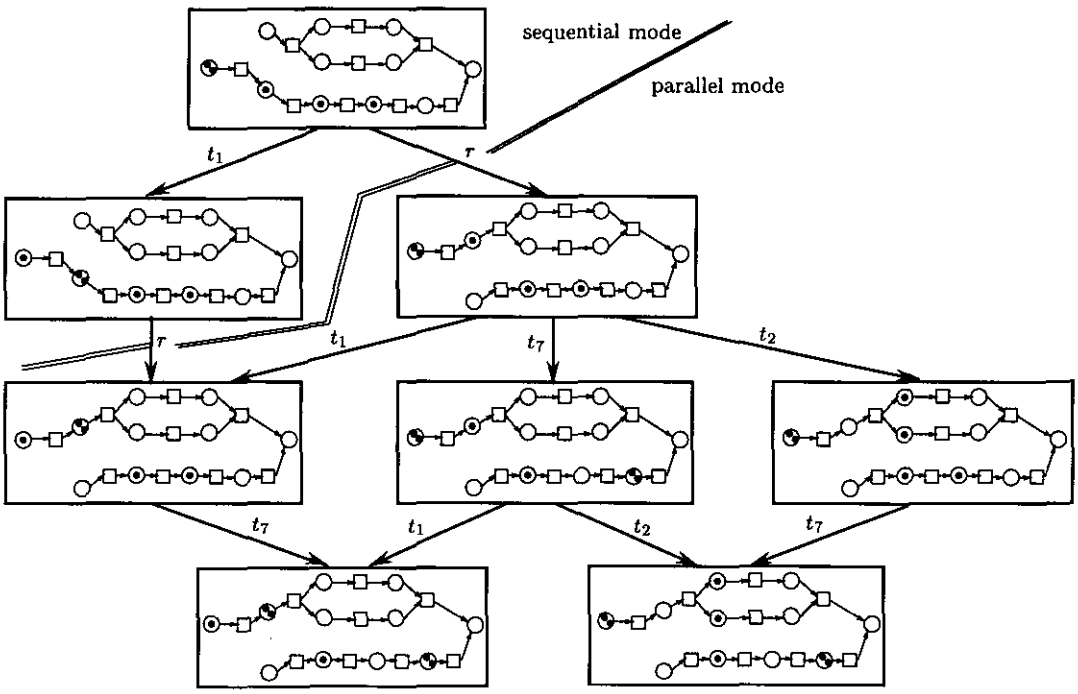


Fig. 2. (part of) the marking graph of the reconfigurable net of Fig. 1

The set of places that exists in marking M, let $D(M) = \{p \in P |\ M(p) \neq \alpha\}$, is termed the *domain* of M. Two markings are said to be equivalent when they have the same domain: $M_1 \equiv M_2 \Leftrightarrow D(M_1) = D(M_2)$. A *mode of operation* is an equivalence class for $\equiv$, it can be identified with a subset $D \subseteq P$ of places. Usually, a reconfigurable net is implicitly attached with a fixed subset of modes of operation. In the above example one has two modes of operation, the sequential mode and the parallel mode, whose respective domains are $P \setminus \{p_3; p_5\}$ and $P \setminus \{p_2; p_4\}$; any marking whose domain is different from these two sets intuitively should not correspond to any state of the system. Moreover as in [2] we consider that nets that model workflow systems have two distinctive places, an *input place* i which is a source place i.e. a place with no pre-transitions: $\forall t \in T \quad F(t, i) = 0$; and an *ouput place* o which is a sink place i.e. a place with no post-transitions

$\forall t \in T \quad F(o,t) = 0$. These places correspond respectively to the beginning and the termination of the processing of a case. In the example of Fig. 1 the input place is place p_1 and the output place is place p_{13}. We end up with the notion of a reconfigurable workflow net which is an adaptation of van der Aalst's notion of workflow net [2].

Definition 2 *A reconfigurable workflow net $\mathcal{N} = (N, \mathcal{O}, i, o)$ is a reconfigurable net $N = (P, T, F, R)$ with an explicit set of modes of operation $\mathcal{O} \subseteq 2^P$ and two distinguished places $i, o \in P$ where i is a source place and o is a sink place. Every mode of operation $\Omega \in \mathcal{O}$ contains places i and o; moreover the set $\mathcal{O}$ is strongly connected in the sense that every $\Omega \in \mathcal{O}$ derives from every other $\Omega' \in \mathcal{O}$ by a (finite) sequence $r_1, \ldots, r_n$ of structure modifying rules, where Ω derives from Ω' by a rule $r : P_1 \to P_2$ when $\Omega = (\Omega' \setminus P_1) \cup P_2$. Finally if Ω and Ω' are subsets of places such that $\Omega = (\Omega' \setminus P_1) \cup P_2$ for some structure modifying rule $r : P_1 \to P_2$, then $\Omega \in \mathcal{O} \Leftrightarrow \Omega' \in \mathcal{O}$.*

The set of modes of operation of a reconfigurable workflow net is therefore a connected component of the directed graph whose vertices are the subsets of places and whose arcs are pairs (Ω, Ω') such that $\Omega = (\Omega' \setminus P_1) \cup P_2$ for some structure modifying rule $r : P_1 \to P_2$; and moreover this component is strongly connected. If $\Omega \in \mathcal{O}$ is a mode of operation, we let i_Ω stand for the marking of domain Ω with one token in place i and no token elsewhere, and similarly o_Ω is the marking of domain Ω with one token in place o and no token elsewhere. If $U \subseteq E$ is a subset of events (usually T, R or E itself), we let $M[U>M'$ if marking M' can be reached from marking M by firing a sequence of events in U. Because of the strong connectedness we can restore any particular mode of operation before starting (or after finishing) the processing of a case: i.e. $i_\Omega[R>i_{\Omega'}$ and $o_\Omega[R>o_{\Omega'}$ for any pair $\Omega, \Omega' \in \mathcal{O}$ of modes of operation.

A token in the input place corresponds to a case entering the system. This case then *flows* through the system until a token appearing in the output place indicates the termination of this case. In the meantime the role played by the marking is twofold. On the one hand, it accounts for the current (distributed) state of progress of the case; on the other hand, it encodes the current state of the system which processes the case. Without loss of generality we can assume that the state of the system be empty when starting a new case. When the case terminates the system should have recovered its initial state in order to be ready to process a new case. The mode of operation may however have changed, but this is not significant because a case should be unaware of the current mode of operation of the system. Finally, since such a system is intended to process cases endlessly it should not have a degraded behaviour: if a transition becomes dead in some state (i.e. it cannot be fired from this state on), then this transition might have been discarded in the first place! These requirements are captured in the following definition which is an adaptation of the similar definition for workflow nets [2]. We let $M \sqsubseteq M'$ when M and M' are markings with the *same domain* D such that $\forall p \in D \quad M(p) \leq M'(p)$ and we let $\mathcal{M}$ denote the set of markings.

Definition 3 *A reconfigurable workflow net is sound if the following conditions are met.*

(i) *Every processing of a case can terminate:*

$$\forall M \in \mathcal{M} \quad \forall \Omega \in \mathcal{O} \quad i_\Omega[E>M \Rightarrow \exists M' \in \mathcal{M}\ \exists \Omega' \in \mathcal{O}\ (M[E>M' \wedge o_{\Omega'} \sqsubseteq M')$$

(ii) *Every termination of a case restores the initial state of the system (but possibly the mode of operation):*

$$\forall M \in \mathcal{M} \quad \forall \Omega, \Omega' \in \mathcal{O} \quad (i_\Omega[E>M \wedge o_{\Omega'} \sqsubseteq M) \Rightarrow M = o_{\Omega'}$$

(iii) *There are no dead transitions:*

$$\forall t \in T \quad \forall \Omega \in \mathcal{O} \quad \exists M, M' \in \mathcal{M} \quad i_\Omega[E>M \wedge M[t>M'$$

Observe that the processing of a case may not terminate and that a change of mode of operation may be required in order to reach termination. The above definition states some properties of the expected behaviour of a workflow system processing an individual case but it says nothing about that system when several cases are being processed concurrently. In real applications every case has an *identity*, an agent performing a task within a worflow system knows which case he is currently processing. We may consider therefore that every new case entering the system is given a *colour* and that this colour is distinct from the colours of the other cases currently flowing through the system. The resulting colored Petri net behaves as follows: in order to fire, a transition is only allowed to pick from its input places tokens of the same colour, it then produces tokens in its ouput places of that same colour. This means that the concurrent processing of multiple cases is represented as the *non interfering* superimposition of the processings of the individual cases. Therefore a reconfigurable workflow net is considered only with respect to an individual case and this justifies the above definition. A weak form of interference between cases exists however due to the fact that the current state of the system may have an effect on the decision as to whether an allowed structure modifying rule should be invoked. For example if the number of tokens in a certain place exceeds a given threshold one may regulate the flow by invoking some structure modifying rule, the converse modification may be invoked latter when the content of that place goes under another threshold; in that sense the manner in which a case is processed may be influenced by the other cases. Notice that such thresholds do not appear in our definition of structure modifying rules which reflect the fact that the decisions concerning the invocation of these rules are *external* to the system. Moreover this interference concerns only the modes of operation which are used when processing a case and our formalism allows the designer of the system to make sure that the processing of a case is insensitive to the dynamic changes occurring within the system. We can if necessary enrich the description of the net by adding extra places and transitions in order to ensure that the processing of cases are independent of the dynamic changes. For instance, one can identify a list of properties that characterize the fact that the case has been correctly processed. In general such a property corresponds to the completion of a task. These properties are represented by extra places $p_{n+1}, \ldots, p_{n+k}$. Another extra place is introduced as the new output place of the

enriched net. This place is filled by an extra transition whose preconditions are the places $p_{n+1}, \ldots, p_{n+k}$ together with the old output place. In that manner a token in the new output place indicates that the case has been correctly processed regardless of the dynamic changes that may have occurred.

3 Boundedness of a Reconfigurable Net

The reachability tree of a marked reconfigurable net is the tree whose root is labelled with the initial marking and such that if V is an arbitrary vertex of that tree labelled with marking M, the arcs originating in V are in bijective correspondence with the firings $M[e>M'$ and the arc associated with $M[e>M'$ is labelled with event e and has its extremity labelled with M'. The reachability tree is thus the "unfolding" of the marking graph of the marked net. If the net is unbounded this tree is infinite. Similar to what is done for ordinary Petri nets, a finite approximation of the reachability tree called the *coverability tree* can be constructed. Two properties are at the basis of the algorithm of Karp and Miller [14]. They correspond to the two following propositions.

Proposition 4 *The order relation between markings is a well-ordering.*

Proof: We recall (see e.g. [6]) that an order relation $(X, \leq)$ is a well-ordering if for every infinite sequence $(x_i, i \in I\!N)$ indices $i < j$ can be found such that $x_i \leq x_j$, equivalently if every infinite sequence in X has an infinite increasing subsequence. The usual ordering on $I\!N$ is a well-ordering. Moreover, by extracting subsequences iteratively (n times), we notice that if $(X, \leq)$ is a well-ordering, then X^n with the pointwise ordering is also a well-ordering. Finally, $I\!N \cup \{\alpha\}$ with the order $x \leq y \Leftrightarrow (x = y = \alpha \vee [x, y \in I\!N \wedge x \leq y])$ is a well-ordering. Indeed, if $(x_i, i \in I\!N)$ is a sequence in $I\!N \cup \{\alpha\}$, then we can extract a subsequence which is constantly equal to α or an increasing sequence of integers according whether we respectively have an infinite number of indices $i \in I\!N$ such that $x_i = \alpha$, or $x_i \in I\!N$. Therefore the order between markings is a well-ordering. ∎

Proposition 5 *The firing rule is monotone:* $\forall e \in E(M_1[e>M_2 \wedge M_1 \sqsubseteq M_1') \Rightarrow \exists M_2' \ (M_1'[e>M_2' \wedge M_2 \sqsubseteq M_2')$; *moreover* $|M_1'| - |M_1| = |M_2'| - |M_2|$. *Thus if* $M[u>M'$ *with* $M \sqsubseteq M'$ *then the sequence* $u \in E^*$ *of firings can be reproduced, i.e.* $\exists M[u^n>M^{(n)}$ *for every* $n \in I\!N$, *and then* $|M^{(n)}| = |M| + kn$ *where* $k = |M'| - |M|$.

Proof: If $e = t \in T$ we have $(M_1[t>M_2 \wedge M_1 \sqsubseteq M_1') \Rightarrow \exists M_2' \ (M_1'[t>M_2' \wedge M_2 \sqsubseteq M_2')$ and $M_1' - M_1 = M_2' - M_2$ (the property of constant effect) as these properties hold for Petri nets. If $e = r \in R$, the first property holds trivially while the property of constant effect is weaken: as token are moved from some places to other places we only have conservation of the total number of tokens, i.e. $|M_1'| - |M_1| = |M_2'| - |M_2|$. ∎

The key observation for the algorithm of Karp and Miller for Petri nets is that,

because of the property of constant effect, when markings M and M' and a sequence of firings $u \in T^*$ can be found such that $M[u>M'$ and $M \sqsubseteq M'$, one can deduce $M[u^n>M^{(n)}$ for every $n \in \mathbb{N}$. Moreover, for every place $p \in P$ such that $M(p) < M'(p)$ one has $M^{(n)}(p) = M(p) + nk$ where $k = M'(p) - M(p) > 0$ and therefore this place is not bounded. Because of the weak form of the property of constant effect, this observation no longer holds for reconfigurable nets. However if one is not concerned with the boundedness of any particular place of the net but with the boundedness of the net itself (i.e. whether there exists some place in the net that is unbounded), then the above propositions are sufficient and boundedness can be verified using the following simplified version of coverability tree.

Definition 6 *The coverability tree of a marked reconfigurable net (N, M_0) is constructed by the following algorithm:*

- *Initially the tree is reduced to its root labelled M_0 and tagged as a "new" vertex.*
- *While "new" vertices exist, do the following:*
  - *Select a new vertex V, let M be its label.*
  - *For every firing $M[e>M'$ do the following:*
    - *Create a new vertex V' labelled M' and an arc from V to V' labelled e.*
    - *If there exists some node V'' on the path from the root to vertex V whose label M'' is such that $M'' \sqsubseteq M'$ then*
      - *If $M'' = M'$ then tag vertex V' "old" else tag it "unbounded".*

      else tag V' "new".
  - *Withdraw V from the set of "new" vertices.*

Proposition 7 *The coverability tree of a marked reconfigurable net is finite.*

Proof: Since the order relation on the set of markings is a well-ordering, the coverability tree of a marked reconfigurable net contains no infinite branch. Since moreover, each vertex has at most $|E|$ successors we deduce by König lemma that this tree is finite. ∎

Proposition 8 *A marked reconfigurable net is bounded if and only if no vertex of its coverability tree is tagged "unbounded".*

Proof: If the coverability tree contains no vertex tagged "unbounded" then the set of labels of its vertices coincides with the set of markings of the reconfigurable net reachable from the initial state (label of the root), therefore the marked reconfigurable net is bounded. If on the contrary the coverability tree contains some vertex V' tagged "unbounded". Thus there exists some vertex V on the path from the root such that $M_0[u>M[v>M'$ where u labels the path from the root to vertex V, v labels the path from vertex V to vertex V', $M \sqsubseteq M'$ and $M \neq M'$. Then by Prop. 5, $M[v^n>M^{(n)}$ with $|M^{(n)}| = |M| + kn$ where $k = |M'| - |M| > 0$. Since there are finitely many places the net is unbounded. ∎

Corollary 9 *The boundedness of reconfigurable net is decidable.*

A reconfigurable workflow net N is said to be bounded if the marked reconfigurable net (N, i_Ω) for $\Omega \in \mathcal{O}$ some mode of operation is bounded. This definition does not depend on the choice of $\Omega \in \mathcal{O}$ because $i_\Omega[R > i_{\Omega'}$ for every pair $\Omega, \Omega' \in \mathcal{O}$.

Proposition 10 *A sound reconfigurable workflow net is bounded and we can decide whether a reconfigurable workflow net is sound.*

Proof: If (N, i_Ω) is not bounded then as seen in the proof of Prop. (8) there exists markings M and M' such that $i_\Omega[E > M$, $M[E > M'$, $M \sqsubseteq M'$ and $M(p) < M(p')$ for some place p. Since N is sound one has $M[u > o_{\Omega'}$ for some sequence $u \in E^*$. By monotony $M'[u > M''$ with $o_{\Omega'} \sqsubseteq M''$ and $M'' \neq o_{\Omega'}$ which contradicts the fact that N is sound. Once boundedness has been checked, the properties i to *(iii)* of Def. 3 may be checked directly on the coverability tree which then coincide with the reachability tree. ∎

However as noted by Hack in [12] "*The size of Karp and Miller's construction in their decision procedure for boundedness and coverability can grow as fast as Ackermann's function of the size of the Petri net*" which shows the intractability of the verification of soundness property via the construction of the coverability tree. Van der Aalst showed in [2] that soundness of a workflow net reduces to boundedness and liveness of a Petri net obtained by adding an extra transition connecting its output place to its input place. Since it is possible to decide boundedness and liveness of free-choice Petri nets in polynomial time [4], he deduced therefrom that soundness of free-choice workflow nets can be decided in polynomial time. We show that van der Aalst's construction can be carried to reconfigurable nets with no significant changes, unfortunately one cannot directly deduce therefrom a polynomial time algorithm for the decision of soundness of reconfigurable workflow nets all of whose configurations are free-choice Petri nets.

Definition 11 *If $\mathcal{N} = (N, \mathcal{O}, i, o)$ is a reconfigurable workflow net where $N = (P, T, F, R)$ and $\Omega \in \mathcal{O}$ is a mode of operation, we let $\overline{\mathcal{N}}_\Omega = (\overline{N}, i_\Omega)$ be the marked reconfigurable net consisting of the reconfigurable net $\overline{N} = (P, \overline{T}, \overline{F}, R)$ and initial marking i_Ω where $\overline{T} = T \cup \{t^*\}$ with $t^* \notin T$ a new transition and the extended flow relation $\overline{F} : (P \times \overline{T}) \cup (\overline{T} \times P) \to \mathbb{N}$ is given by*

$$\overline{F}(t,p) = \begin{cases} F(t,p) & \text{if } t \in T \text{ and } p \in P \\ -1 & \text{if } t = t^* \text{ and } p = o \\ 1 & \text{if } t = t^* \text{ and } p = i \\ 0 & \text{otherwise} \end{cases}$$

$\overline{N}$ is obtained from N by adding a new transition t^* which is enabled when a case has reached termination (there is one token in the output place) and then removes that case to the system and introduces a new one (by adding one token in the input place).

Proposition 12 *The reconfigurable workflow net $\mathcal{N} = (N, \mathcal{O}, i, o)$ is sound if and only if the reconfigurable net $\overline{\mathcal{N}}_\Omega$ is live and bounded.*

Proof: We first notice that since each initial state i_Ω is reachable from any other initial state by structure modifying rules ($i_\Omega[R>i_{\Omega'}$), the reconfigurable net $\overline{\mathcal{N}}_\Omega$ is live and bounded if and only if $\overline{\mathcal{N}}_{\Omega'}$ is live and bounded for any $\Omega' \in \mathcal{O}$. We first show that if $\overline{\mathcal{N}}_\Omega$ is live and bounded then $\mathcal{N}$ is a sound reconfigurable workflow net. Since $\overline{\mathcal{N}}_\Omega$ is live, transition t^* is potentially firable in every reachable marking, i.e. condition *(i)* in Def. 3 is satisfied. If $\Omega, \Omega' \in \mathcal{O}$, we let $<\Omega, \Omega'>$ denote the set of integers $n \in \mathbb{N}$ for which there exists some marking M such that $i_\Omega[E \cup \{t^*\}>M$, $i_{\Omega'} \sqsubseteq M$, and $|M| = n + 1$. That is to say, by Prop. 5, that $<\Omega, \Omega'>$ records all possible increases of the size of markings along computations in $\overline{\mathcal{N}}_\Omega$ from some marking greater than i_Ω to some marking greater than $i_{\Omega'}$. Therefore $(n \in <\Omega, \Omega'> \wedge m \in <\Omega', \Omega''>) \Rightarrow n+m \in <\Omega, \Omega''>$. Now $<\Omega, \Omega'> \neq \emptyset$. Actually since condition *(i)* in Def. 3 is satisfied, $i_\Omega[E>M$ for some M such that $o_{\Omega''} \sqsubseteq M$ for some $\Omega'' \in \mathcal{O}$; since $o_{\Omega''}[R>o_{\Omega'}$ and by Prop. 5 we deduce $M[R>M'$ with $o_{\Omega'} \sqsubseteq M'$ and then $M'[t^*>M''$ with $i_{\Omega'} \sqsubseteq M''$ as required. Therefore, since $\overline{\mathcal{N}}_\Omega$ is bounded, we deduce that $<\Omega, \Omega'> = \{0\}$ for all $\Omega, \Omega' \in \mathcal{O}$, and thus $(i_\Omega[E>M \wedge o_{\Omega'} \sqsubseteq M) \Rightarrow M = o_{\Omega'}$, i.e. condition *(ii)* in Def 3 is satisfied. Condition *(iii)* in Def 3 follows from the fact that $\overline{\mathcal{N}}_\Omega$ is live for every $\Omega \in \mathcal{O}$.

Conversely, let us assume that $\mathcal{N}$ is sound.

First we show that $\overline{\mathcal{N}}_\Omega$ is bounded. Since $\mathcal{N}$ is sound, the extended net $\overline{\mathcal{N}}$ returns to some initial state i_Ω when t^* fires, it is then enough to check that the marked reconfigurable net (N, i_Ω) is bounded for every $\Omega \in \mathcal{O}$. If this is not the case, then by construction of the coverability tree, we deduce there exist markings M_1 and M_2 such that $i_\Omega[E>M_1$, $M_1[E>M_2$, $M_1 \sqsubseteq M_2$, and $M_1 \neq M_2$. By soundness of $\mathcal{N}$, we deduce $M_1[u>o_{\Omega'}$ for some $u \in E^*$, and then by Prop. 5 $M_2[u>M_2'$ with $o_{\Omega'} \sqsubseteq M_2'$ and $M_2' \neq o_{\Omega'}$ (because $|M_2'| - 1 = |M_2| - |M_1| > 0$) which contradicts soundness of $\mathcal{N}$.

Second we show that $\overline{\mathcal{N}}_\Omega$ is live. Since $\mathcal{N}$ is sound, transition t^* is potentially firable in every reachable marking and its firing always leads to some initial state i_Ω, since moreover $i_\Omega[R>i_{\Omega'}$ for arbitrary pair of mode of operations, we deduce that net $\overline{\mathcal{N}}_\Omega$ is cyclic (every initial state i_Ω and thus any reachable marking is reachable from any reachable marking). Since moreover there is no dead transitions in $\overline{\mathcal{N}}_\Omega$ (by condition *(iii)* in Def. 3 and the fact that t^* is not dead) we deduce that this marked net is live. ∎

4 Reconfigurable Nets as Self-Modifying Nets

The purpose of this section is to show that reconfigurable nets are self-modifying nets. Self-modifying nets [18, 19] are generalizations of place/transition nets where the flow relation between a place and a transition depends on the marking.

Definition 13 *A self-modifying net is a structure $N = (P, T, F)$ where $P = \{p_1, \ldots, p_m\}$ is a non empty and finite set of places, $T = \{t_1, \ldots, t_n\}$ is a non empty and finite set of transitions disjoint from P, and $F : (P \times T) \cup (T \times P) \to \mathbb{N}^{P_*}$ is the flow relation where $P_* = P \cup \{*\}$ and $* \notin P$. A vector $\varphi \in \mathbb{N}^{P_*}$ can be represented by a formal sum $\varphi = \lambda_0 + \sum_{i=1}^{m} \lambda_i \cdot p_i$ where the constant coefficient is the entry corresponding to the fictituous place: $\lambda_0 = \varphi(*)$ and $\lambda_i = \varphi(p_i)$. A marking of net N is a map $M : P \to \mathbb{N}$. If $M \in \mathbb{N}^P$ is a marking and $\varphi \in \mathbb{N}^{P_*}$, we let $\varphi(M) = \lambda_0 + \sum_{i=1}^{m} \lambda_i \cdot M(p_i)$ denote the evaluation of the affine function φ in marking M. We let $M[t>M'$ when transition t is enabled in marking M and leads to marking M'. This transition relation is given by:*

$$M[t>M' \Leftrightarrow \forall p \in P \;\; M(p) \geq F(p,t)(M) \;\; \wedge \;\; M' = M - F(p,t)(M) + F(t,p)(M)$$

A marked self-modifying net is a self-modifying net together with an initial marking.

Proposition 14 *Any marked reconfigurable net can be associated with a marked self-modifying net with isomorphic marking graph and whose set of transitions is the set of events of the reconfigurable net.*

Proof: The translation of a reconfigurable net into an equivalent self-modifying net is staightforward: we represent each place p of the reconfigurable net by three places $\exists_p$, $\neg\exists_p$ and p. The first two places are complementary 1-bounded places whose contents indicate whether place p exists in the current marking and the third place, also denoted p, has the same content than the original place p when this place exists. Figure 3 gives a sketch of the translation whose precise definition follows. Any reconfigurable net $N = (P, T, F, R)$ is associated

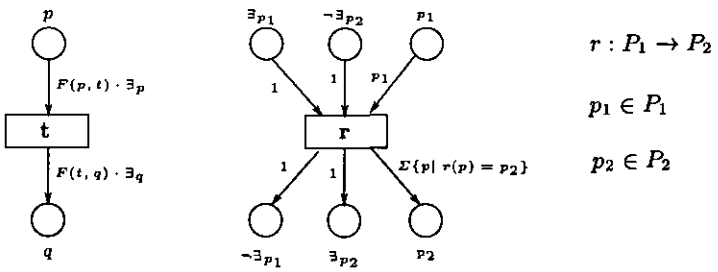


Fig. 3. translating a reconfigurable net into an equivalent self-modifying net

with a self-modifying net $\tilde{N} = (\tilde{P}, \tilde{T}, \tilde{F})$ defined as follows. The set of places $\tilde{P} = \exists P \cup \neg\exists P \cup P$ is the disjoint union of three copies of set P whose respective typical elements are noted $\exists_p$, $\neg\exists_p$ and p for p ranging in P. The set of transitions $\tilde{T} = T \cup R$ consists of the transitions of the original net together with its set of structure modifying rules, i.e. its set of events. Finally the flow relation $\tilde{F}$ is

given by the following identities where $\tilde{p}$, t and $r : P_1 \to P_2$ range respectively in $\tilde{P}$, T and R.

$$\tilde{F}(\tilde{p},t) = \begin{cases} F(p,t) \cdot \exists_p \text{ if } \tilde{p} = p \in P \\ 0 \qquad\qquad \text{otherwise} \end{cases} \quad \tilde{F}(t,\tilde{p}) = \begin{cases} F(t,p) \cdot \exists_p \text{ if } \tilde{p} = p \in P \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$

$$\tilde{F}(\tilde{p},r) = \begin{cases} 1 \text{ if } \tilde{p} = \exists_{p_1} \wedge p_1 \in P_1 \\ \quad \text{or } \tilde{p} = \neg\exists_{p_2} \wedge p_2 \in P_2 \\ p_1 \text{ if } \tilde{p} = p_1 \in P_1 \\ 0 \text{ otherwise} \end{cases} \quad \tilde{F}(r,\tilde{p}) = \begin{cases} 1 \text{ if } \tilde{p} = \neg\exists_{p_1} \wedge p_1 \in P_1 \\ \quad \text{or } \tilde{p} = \exists_{p_2} \wedge p_2 \in P_2 \\ \sum\{p | r(p) = p_2\} \text{ if } \tilde{p} = p_2 \in P_2 \\ 0 \text{ otherwise} \end{cases}$$

A marking M of the reconfigurable net N is associated with the marking $\tilde{M}$ of the self-modifying net $\tilde{N}$ given by

$$\begin{aligned} \tilde{M}(\exists_p) &= \text{if } M(p) \neq \alpha \text{ then } 1 \quad \text{else } 0 \\ \tilde{M}(\neg\exists_p) &= \text{if } M(p) \neq \alpha \text{ then } 0 \quad \text{else } 1 \\ \tilde{M}(p) &= \text{if } M(p) \neq \alpha \text{ then } M(p) \text{ else } 0 \end{aligned}$$

The above relations induce a bijective correspondance between the markings of N and those markings $\tilde{M}$ of $\tilde{N}$ such that $\forall p \in P \quad \tilde{M}(\exists_p), \tilde{M}(\neg\exists_p) \in \{0;1\}$ and $\tilde{M}(\exists_p) = 1 \Leftrightarrow \tilde{M}(\neg\exists_p) = 0$ and $\tilde{M}(\exists_p) = 0 \Rightarrow \tilde{M}(p) = 0$. A direct comparison of Def. 1 and Def. 13 shows that an event $e \in T \cup R$ of N is enabled in a marking M of the reconfigurable net N if and only if as a transition of the self-modifying net $\tilde{N}$ it is enabled in the associated marking $\tilde{M}$; moreover $M[e>M'$ in N if and only if $\tilde{M}[e>\tilde{M}'$ in $\tilde{N}$. Therefore the mapping $\tilde{(\cdot)}$ is an isomorphism between the marking graph of the marked reconfigurable net (N, M) and the marking graph of the marked self-modifying net $(\tilde{N}, \tilde{M})$ for any marking M of N. ∎

Boundedness is not decidable for self-modifying nets whereas it is decidable for reconfigurable nets. In order to better delimit the borderline between those self-modifying nets for which boundedness can be decided from those for which it cannot, let us precise some terminology. A pair $(p,t) \in P \times T$ is termed an *input arc* (with respect to transition t) if $F(p,t) \neq 0$. Similarly a pair $(t,p) \in T \times P$ such that $F(t,p) \neq 0$ is termed an *output arc*. An input arc (p,t) is an ordinary arc if $F(p,t) \in \mathbb{N}$, and it is a reset arc if $F(p,t) = p$. An output arc (t,p) is an ordinary arc if $F(t,p) \in \mathbb{N}$. A self-modifying net is a *post-self-modifying net* (respectively a *pre-self-modifying net*) [18] if every input arc (resp. ouput arc) is an ordinary arc; and it is a *Reset/Set nets with infinite capacity* [13] if every input arc is either an ordinary arc or a reset arc. Valk proved in [18] that boundedness of post-self-modifying nets can be decided. The same result has been erroneously stated for the class of Reset/Set nets with infinite capacity and for the class of pre-self-modifying nets. Indeed Dufourd [7] has proved recently that boundedness is indecidable for the class of Petri nets with reset arcs, i.e. for the class of self-modifying nets such that every input arc is either an ordinary arc or a reset arc and every output arc is an ordinary arc.

5 Conclusion

In this paper we have introduced a class of high level Petri nets, called *reconfigurable nets*, that can dynamically modify their own structures by rewriting some of their components. Boundedness of a reconfigurable net can be decided by constructing its coverability tree. Moreover such a net can be simulated by a self-modifying Petri net. The class of reconfigurable nets thus provide a subclass of self-modifying Petri nets for which boundedness can be decided. Delayed dynamic changes within workflow systems in the sense of [8] can then be handled in an extension of van der Aalst's workflow nets [2]. For this class (the *reconfigurable workflow nets*), a notion of soundness has been defined that can be verified using the coverability tree construction.

A reconfigurable net can be seen as a bunch of Petri nets: its configurations. A configuration of a reconfigurable net gives a description of the system for some mode of operation. It could be interesting to investigate the properties of a reconfigurable net in relationship with specific assumptions on its configurations, e.g. according whether they are acyclic, 1-safe or free-choice. The workflow net models of [5] for instance are acyclic, extended free-choice elementary net systems, whereas the worflow nets of [2] are usually assumed to be free-choice or almost free-choice. An open question in that direction is whether there exists a polynomial time algorithm for deciding the soundness property of free-choice reconfigurable workflow nets. Additional assumptions concerning the set of structure modifying rules may also be considered. For instance in a forthcomming paper we shall restrict our attention to the class of reconfigurable nets whose structure modifying rule $r : P_1 \to P_2$ are bijections. Under some extra assumption such a net can be simulated by a stratified Petri net [3], that is to say by a self-modifying Petri net for which a stratification of the set of places into layers exists so that the flow relations attached to a place involve only the content of places of lower layers. The self-modifying Petri nets that we have used to simulate reconfigurable nets were not stratified, and this was essential since reconfigurable nets unlike stratified Petri nets are in general not *reversible* in the sense that we cannot for each firing $M[e>M'$ deduce marking M from the data of event e and marking M'; reconfigurable nets are reversible however if all structure modifying rules are assumed to be bijective.

We have assumed, as it is implicitly done in [2] for workflow nets, that the system can identify, e.g. by using coloured tokens, each of the cases that are processed. Since a marking accounts at the same time for the current states of progress of the cases and the state of the system which processes these cases, this implies that the state of the system itself be multi-coloured, i.e. it is a vector of states associated with each of the cases currently flowing within the system. This assumption may be considered indesirable and we may seek for a non interference condition to be added to the definition of soundness that will ensure that the behaviour is not changed when colours are forgotten.

Finally, it might be interesting to investigate the notion of a *controlled* reconfigurable net which stands for a reconfigurable net together with a control part that regulate the flow in the system. This control would have the marking

of the evolving net as input and for a set of allowed structure modifying rules as output.

References

[1] ASPERTI, A., and BUSI, N., *Mobile Petri Nets.* Technical Report UBLCS-96-10, University of Bologna, Italy (1996).

[2] VAN DER AALST, W.M.P., *Verification of Workflow Nets.* Proceedings of ICATPN'97, volume 1248 of Lecture Notes in Computer Science, Springer Verlag (1997) 407–426.

[3] BADOUEL, E., and DARONDEAU, PH., *Stratified Petri Nets.* Proceedings of FCT'97, volume 1279 of Lecture Notes in Computer Science, Springer Verlag (1997) 117–128.

[4] DESEL, J., and ESPARZA, J., *Free Choice Petri Nets.* Volume 40 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (1995).

[5] DE MICHELIS, G., and ELLIS, C.A., *Computer Supported Cooperative Work and Petri Nets.* Third Advanced Course on Petri Nets, Dagstuhl Castle, Germany (1996). To appear in Springer Verlag Lecture Notes in Computer Science.

[6] DIESTEL, R., *Graph Theory.* volume 173 of *Graduate Texts in Mathematics.* Springer Verlag (1996).

[7] DUFOURD, C., FINKEL, A., and SCHNOEBELEN, PH., *Reset nets between decidability and undecidability.* To appear in Proceedings of ICALP'98.

[8] ELLIS, C., KEDDARA, K., and ROZENBERG, G., *Dynamic Change within Workflow Systems.* Proceedings of the Conference on Organizational Computing Systems, ACM Press, New York (1995) 10–21.

[9] ELLIS, C.A, and NUTT, G.J., *Modeling Enactment of Workflow Systems.* Proceedings of ICATPN'93, volume 691 of Lecture Notes in Computer Science, Springer Verlag (1993) 1–16.

[10] FOURNET, C., and GONTHIER, G., *The reflexive chemical abstract machine and the join-calculus.* Proceedings of the 23^{rd} ACM Symposium on Principle of Programming Languages, (1996).

[11] FOURNET, C., GONTHIER, G., LÉVY, J.-J., and RÉMY, D., *A Calculus of Mobile Agents.* Proceedings of CONCUR'96, volume 1119 of Lecture Notes in Computer Science, Springer Verlag (1996) 406–421.

[12] HACK, M.H.T., *The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems.* Computation Structures Group Memo 107. Cambridge, Massachussetts: MIT, Department of Electrical Engineering (1974).

[13] HEINEMANN, B., *Subclasses of Self-Modifying Nets.* In C. Girault and W. Reisig (Eds.) First European Workshop on Application and Theory of Petri Nets. Informatik Fachberichte, Springer (1982) 187–192.

[14] KARP, R.M., and MILLER, R.E., *Parallel program schemata.* Journal of Computer and System Sciences vol. 3 (1969) 147–195.

[15] MILNER, R., PARROW, J., and WALKER, D., *A calculus of mobile Processes, I-II.* Information and Computation, vol. 100, no 1, (1992) 1–40 and 41–77.

[16] MURATA, T., *Petri Nets: Properties, Analysis and Applications.* Proceeding of the IEEE, 77(4) (1989) 541–580.

[17] PETRI, C. A., *Kommunikation mit Automaten*, Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn (1962). *English translation*: Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, vol. 1, suppl. 1 (1966).

[18] VALK, R., *Self-Modifying Nets, a Natural Extension of Petri Nets.* Proceedings of Icalp'78, Lecture Notes in Computer Science vol. 62 (1978) 464–476.
[19] VALK, R., *Generalizations of Petri Nets.* Proceedings of MFCS'81, Lecture Notes in Computer Science vol. 118 (1981) 140–155.

This article was processed using the LaTeX macro package with LLNCS style

Simple Workflow Models

Alessandra Agostini, Giorgio De Michelis

Cooperation Technologies Laboratory
Information Sciences Dept. University of Milano
(email: {agostini, gdemich}@dsi.unimi.it)

Abstract. Workflow management systems are considered a hot technology but they do not have up to now the diffusion of productivity tools, e-mail systems and/or groupware platforms. We think that this is due to the fact that existing workflows management systems, in general, do not offer all the services needed by the potential users. In particular, they do not have modeling capabilities adequately supporting exceptions, multiple views as well as static and dynamic changes. In this paper we introduce the modeling environment of the workflow management module of the Milano system —a prototype of a CSCW platform we are developing at the Cooperation Technology Laboratory of the University of Milano. The underlying idea of the Milano workflow management module is that workflow models must be simple and based on a formal theory, so that the various views, properties, changes can be computed when needed and not explicitly modeled. The modeling environment of Milano is based on a subclass of the Elementary Net Systems and on its properties. An example, derived from a real bank procedure, is discussed throughout the paper.

1. Introduction

Since several years, workflow management systems are announced as the next best-selling computer application (Koulopoulos, 1995) but up to now they do not have reached the success of other packages as productivity tools, e-mail systems, web-browsers and even groupware platforms.

Why do workflow management systems remain in limbo while almost every observer argues for their utility and so few users really apply them within real organizations? The question has not a unique simple answer (Abbott, Sarin, 1994), but it deserves the attention of anyone interested in the development of workflow technology. Let us recall one issue emerging when we try to understand what workflow management systems should be to become usable in real work situations; we do not claim that it is the unique point with respect to the above question, but we think that its relevance should not be underestimated.

The relevance of workflow technology has grown together with the emergence of process oriented organizations and the related change management techniques (e.g.,

business process reengineering and continuous process improvement; White, Fischer, 1994). Any workflow management system, therefore, should be oriented to support changes in the organization and to make it flexible. Moreover, workflow changes should be as fast as possible in order to react timely to process changes, and they should be implemented on the fly into already running workflow instances. With respect to these features, most existing workflow management systems appear to be inadequate: it is difficult to interrupt them, to exit their normal flow and to reenter into it, while breakdowns are very frequent (they are the norm in many cases); they are based on complex and sometimes multiple process models (integrating data models, normal and exceptional flows, role descriptions), whose changes need careful and time consuming analysis; they need to be designed by expert programmers, introducing a time delay between process and workflow changes; they do not support multiple viewpoints on the process, corresponding to the various actors with the different objectives and roles they support (the manager, the initiator, the task executor and, why not, the customer).

Many observers have also argued that most workflow management systems make business processes too rigid, not allowing their users to react freely to the breakdowns occurring during their evolution (Bowers et al., 1995). Some of them seem to charge the responsibility of this rigidity to their using formal workflow models (formal models can not fully capture the knowledge people use while acting within a business process), other to the strict coupling between modeling and executing they introduce (models should be cognitive artifacts; Norman, 1991; not constraining the behaviour of the actors; Suchman, 1987; Dourish et al., 1996). We agree with the above points, but we are convinced that the rigidity of existing workflow management systems should not be attributed neither to their using formal models nor to their coupling of modeling and execution, but to the above mentioned weaknesses affecting them.

We argue in this paper that, contrary to what appears common sense, formal, theory-based models can contribute to the solution of the above problems, if they are conceived from a different perspective. Good algebra, in fact, offers effective tools for creating a process modeling environment exhibiting the following properties:

- it allows to simulate the process before its execution;
- it allows formal verification of some workflow properties;
- it supports an unambiguous graphical representation of the workflow;
- it allows to use a minimal input for redundant outputs, through the algorithmic completion of the model;
- it supports multiple views of the process, through synthesis algorithms and model conversions;
- it allows the automatic derivation of exceptional paths from the acyclic normal flow of the process, when needed;
- it enacts automatically model changes on the running instances of a workflow, protecting them from undesired outcomes.

What is needed in order to get all these services from algebraic theory is to keep workflow models as simple as possible, i.e., to use a *divide et impera* approach to the

workflow, treating in a distinct way: the execution of the tasks embedded in the workflow steps; the data flow, the control flow, the latter being the only issue to be handled directly by the workflow management system.

In this paper, we present the prototype of the workflow management system we are developing within the Milano system, a groupware platform supporting its users while performing concurrently various cooperative processes, discussing how, the theory it embodies is providing the above services to its users. An example derived from a real credit procedure used within an Italian bank is used throughout the paper.

2. The Workflow Management System of Milano

In 1994 at the Cooperation Technology Laboratory the authors —together with Maria Antonietta Grasso, and several students— started the development of the prototype of a new CSCW system, called Milano (De Michelis, Grasso, 1994; Agostini et al., 1997). Milano is a CSCW platform supporting its users while performing within cooperative processes (De Michelis, 1995, 1997). Milano is based on a situated language-action perspective (Suchman, 1987; Winograd, Flores, 1986; Winograd, 1987) supporting them to keep themselves aware of the history they share with the actors with whom they cooperate. It offers them a set of tools strictly integrated with each other to live with them that history; in particular, a multimedia conversation handler and a workflow management system. Without adding more details about the other components of Milano (the interested reader can find a more complete account on it in (Agostini et al., 1997)), let us spend some more words on its workflow management system and, in particular on its specification module.

The Milano workflow management system is a new generation workflow management system (Abbott, Sarin, 1994): its aim is to support not only its users while performing in accordance with the procedure described in its model, but also when they either need to follow an exceptional path or when they need to change the workflow model. The workflow model, therefore, within Milano is not only an executable code, but also a cognitive artifact. It is, in fact, an important part of the knowledge its different users (the initiator of a workflow instance, the performer of an activity within it, the supervisor of the process where it is enacted and, finally, the designer of the workflow model) share while performing within a cooperative process.

The model, therefore, must not only support the execution of several workflow instances, but it must also support the enactment of any model change on all the ongoing instances (dynamic changes). On the other hand, its cognitive nature requires that a workflow model supports all its users to understand their situation, to make decisions, to perform effectively. The workflow model is not merely a program to be executed and/or simulated by the execution module with a graphical interface to make it readable by its users. Rather, it is a formal model whose properties allow the

user to get different representations of the workflow, to compute exceptional paths from the standard behaviour, to verify if a change in the model is correct with respect to a given criterion and to enact safely a change on the ongoing instances.

For this reason, the specification module of the Milano workflow management system is based on the theory of Elementary Net Systems (ENS) (Rozenberg, 1987; Thiagarajan, 1987). In fact ENS has some nice mathematical properties that appear suitable to provide the above services. For instance, using ENS, it is possible to compute and classify forward- and backward-rolls linking their states; there is a synthesis algorithm from Elementary Transition Systems (ETS) to ENS (Nielsen et al., 1992); the morphisms in ENS (ETS) preserve some important behavioural properties. Moreover, since Milano is based on the idea that workflows must be as simple as possible, its workflow models constitute a small subcategory of ENS, namely Free-Choice Acyclic Elementary Net Systems, whose main properties are computable in polynomial time, allowing an efficient realization of the specification module.

3. A Workflow Example: the Credit Procedure

The selected example is the process through which a customer request for a new credit is managed by a bank. This example is extracted from a real case study; for a more complete description see (Schael, Zeller, 1993; Agostini et al., 1994).

In the credit procedure the client interacts with the agency director who is responsible for the whole procedure. After a preliminary informal investigation of which the director is in charge, where the client motivation is exploited, the documents the clients provides to support her request are collected and two parallel processes start: in the first, the information about the client accounts are collected and the practice is perfected, while in the second, several external data bases are checked to control if the client has other credits, financial insolvency or any other critical financial situation. The two processes meet when a report on the credit request is written. At this point the decision process starts, that follows different paths depending on the value of the requested credit.

If the credit request is under 50,000,000 Lit. the agency director can write a credit proposal and submit it to the district coordinator for her approval. Otherwise other bodies of the bank have to write the credit proposal and/or decide its approval. If the credit proposal gets the corresponding approval, the client has to sign a copy of the contract.

The organizational structure of the bank is sketched in Figure 1, where only people or offices taking part in the credit procedure are shown. In the following a brief description of the organizational roles belonging to the cooperation network is provided.

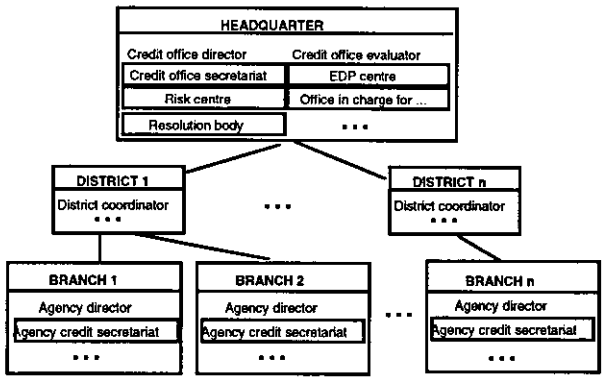


Figure 1: Organizational Structure of the Bank

The **Agency Director** —AD in short in the next Figures— is responsible for the agency's overall performance and for commercial development. According to the organizational model of the bank, she is the initiator of the credit procedure. In line with current rules of the bank, her role is characterized by autonomy and full responsibility in initiating a credit request procedure; her deliberation competence is limited to loans ≤ 50,000,000 Lit.

The **District Coordinator** (DC) has responsibility over the bank budget for credit operations in her area of competence. She is informed on all credit proposals in order to explore new business opportunities. Her deliberation competence is limited to loans ≤ 80,000,000 Lit.

The **Credit Office director** (CO) is an experienced manager. She has the responsibility to assure a check on credit activities in general. Her deliberation competence is limited to loans ≤ 200,000,000 Lit.

The **Resolution Body** (RB) —that is, the Top Management Council of the Bank— has the deliberation competence on all the loans > 200,000,000 Lit.

In order to take her decision, the agency director asks agency employees (generally credit and/or EDP experts) for information about the client. This information can be obtained from several sources, which can be internal or external to the bank.

4. Modeling Workflows in Milano

Let us introduce, in the following, the main definitions and facts about modeling workflows in Milano and let us illustrate them through the credit procedure example. To avoid repetitions, we refer, for the main definitions on Elementary Net Systems and Elementary Transition Systems to (Rozenberg, Thiagarajan, 1986; Nielsen et al., 1992; Bernardinello, 1993)

As anticipated above, the specification module offers two different representations of a workflow model: the first one, called Workflow Net-Model, is based on

Elementary Net Systems, while the second one, called Workflow Sequential-Model, is based on Elementary Transition Systems.

Definition 1 - Workflow Net Model
A Workflow Net-Model is an Elementary Net System, $\Sigma=(B,E,F,c_{in})$, such that the following hold:
a) Σ is structurally acyclic (there are not cycles in the graph);
b) Σ is extended Free-Choice (all conflicts are free).

The class of Workflow Net Models is called WNM.

Example 2
In Figure 2 it is presented the Workflow Net Model representing the credit procedure described in paragraph 3.

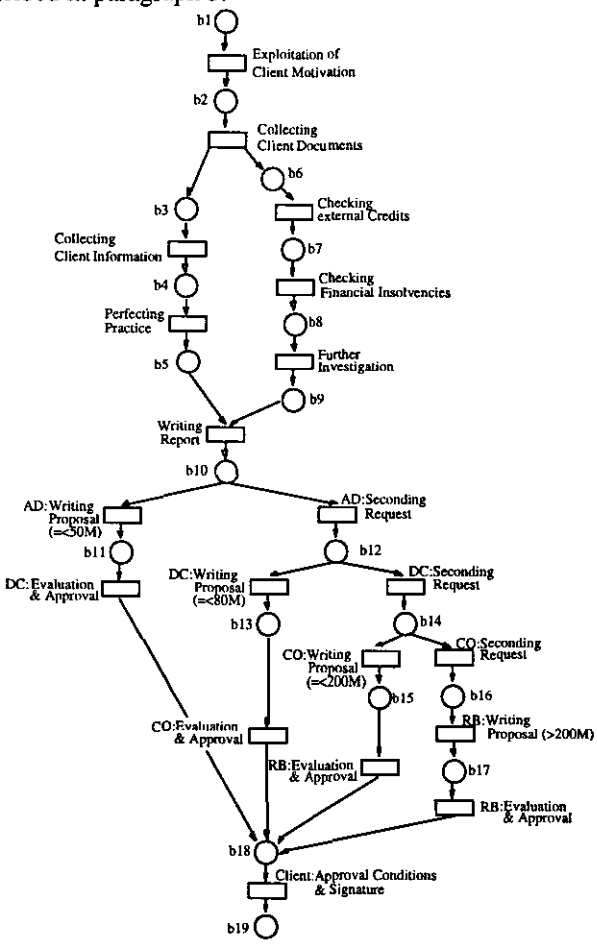


Figure 2

We would like to anticipate that within Figure 2 —thanks to the capacity of handling exceptions by jumps see Example 9— we can avoid to specify all possible cases of withdraw or rejection of the credit procedure. In this particular procedure it is of paramount importance since rejections and withdraws can occur in almost every state of the process.

Definition 3 - Workflow Sequential Model
A Workflow Sequential Model is an Elementary Transition System $A=(S,E,T,s_{in})$, such that the following hold:
a) A is acyclic (there are not cycles in the graph);
b) A is well structured (all diamonds have no holes and the transitions with the same name are parallel lines in a diamond).
The class of Workflow Sequential Models is called WSM.

Example 4
Figure 3 presents the Workflow Sequential Model of the credit procedure introduced in paragraph 3, whose Workflow Net Model is given in Figure 2.

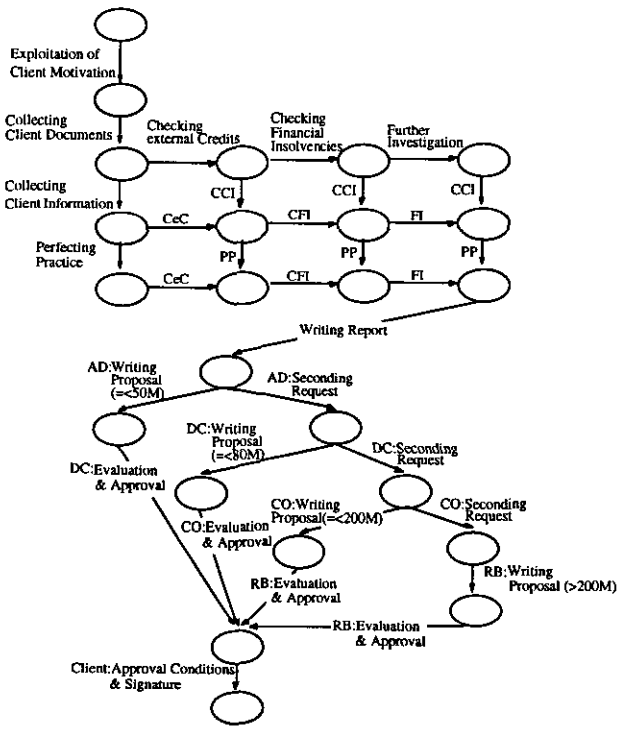


Figure 3

While the Workflow Net Model (Figure 2) is a local state representation making explicit, for example, the independence between the actions of 'Perfecting Practice' and 'Checking Financial Insolvencies', the Workflow Sequential Model (Figure 3) is a global state representation, where the path followed during the execution of an instance is made immediately visible.

It is well known that the sequential behaviour of an ENS can be represented as an ETS and, conversely, given an ETS it is possible to synthesize an ENS whose sequential behaviour is equivalent to the source ETS (Nielsen et al., 1992). It is easy to show that the above relation between ENS and ETS restricts itself to a relation between WNM and WSM.

The algorithm to build the ENS corresponding to ETS is based on the computation of Regions (subsets of S uniformly traversed by action names). While the algorithm presented in (Nielsen et al., 1992) generates a saturated ENS, having a place for each region of the source ETS, Luca Bernardinello (1993) has introduced a synthesis algorithm generating an ENS having a place for each Minimal Region of the source ETS, that is not a minimal representation of an ENS having the behaviour described in the source ETS but has some nice properties (e.g., it is contact-free and state-machine decomposable) making it very readable and well structured. We have therefore decided to normalize each WNM to its Minimal Regional representation and to associate to each WSM its minimal regional representation.

Fact 5
The sequential behaviour of a WNM can be represented as a WSM and conversely, given a WSM there is a WNM whose sequential behaviour is equivalent to it.

Proof outline
The proof is based on the fact that the sequential behaviour of an acyclic extended free-choice Elementary Net System is acyclic and well structured and, conversely, the (Minimal) Regions of an acyclic well structured Elementary Transition System are such that the corresponding Elementary Net System is both acyclic and extended free-choice.

The synthesis algorithm for ENS has been proved to be NP-complete (Badouel et al., 1997), making impossible to use it in real applications. The strong constraints imposed to WNM allow a rather efficient computation of Minimal Regions, so that it is usable in the specification module of the Milano Workflow Management System. Let us sketch the algorithm for the computation of the Minimal Regions of a WNM.

Algorithm 6

Let A=(S,E,T, s_{in}) be a Workflow Sequential Model. The following algorithm computes the minimal regions of A.

```
begin
C := {(S - { s_in},{ s_in})};
R:= ∅;
while C _ ∅ do
    C := C - (S',r) with S' maximal;
    E_r := {e| ∃s ∈ S', e exits s};
    E'_r := {e| e ∈ E_r and ∃s ∈ S' – r; e exits s};
    if E_r = ∅
        then
            R := R ∪ {r};
        else
            if E'_r = ∅
                then
                    R := R ∪ {r};
                    C := C ∪ {(S",r')| ∃e ∈ E_r, r' = {s| e enters s}
                    and S" ={s| s ∈ S' - (r ∪ r') and s reachable from a state of r'};
                else
                    C:= C ∪ {(S",r')| ∃e ∈ E'_r, r' = r ∪ {s| e exits s}and S" = S' - r'};
            fi
        fi
    od
end.
```

Example 7

Figure 4 labels each state of the WSM of Figure 3 with the Minimal Regions containing it.

It is not difficult to see that the WNM of Figure 2 has a place for each of its Regions (it is therefore the result of the synthesis algorithm applied to the WSM of Figure 4) and that the WSM of Figure 3 is isomorphic to it.

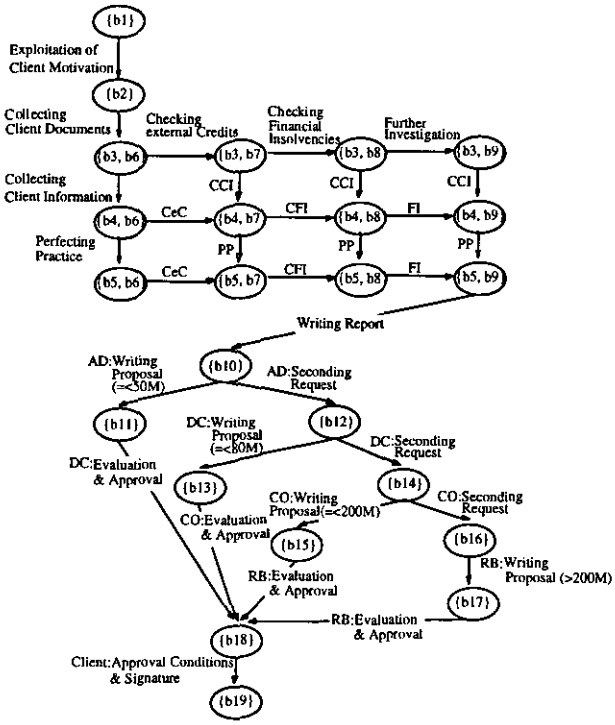


Figure 4

Fact 8
The algorithm given above is polynomial in the size of A (of its set of States, S).

Proof outline
The number of elements we can put in C lies between |S| and $2.\sqrt{|S|}$. Moreover each step of the algorithm requires at most one observation of each element of S.

The efficiency of Algorithm 6 grants that the switch between the two representations of a workflow model (namely WNM and WSM) can be computed whenever necessary, so that there are no constraints imposing a particular representation to the user. The problems related to the graphical visualization of the two representations (e.g., multi-dimensional diamonds will appear as intricate and difficult to read graphs) are not considered in this context. They are taken into account within the framework of a system for the visualization of graph-based models (Bertolazzi et al., 1995).

The reader may object that the constraints imposed to WNM (WSM) are too strong so that the actors are forced to follow very rigid prescriptions. This is not true, since the actors, whenever they can not act in accordance with the model, can jump (either forward or backward) to another state from which execution can progress again. The

freedom in the choice of the states that may be reached through jumps is not constrained by the model but can be constrained in accordance with the rules of the organization where the workflow is modeled. The actors are supported in the choice of an authorized jump by the possibility of computing and classifying composed paths in the graph.

Without entering into irrelevant technical details, let us present a simple example where it is assumed that the organization allows two different classes of jumps: *strongly linear jumps* (moving in the WNM only one token) not requiring any type of authorization and *weakly linear jumps* (canceling two or more tokens and writing one token in the WNM) requiring the authorization of the process initiator, i.e. of the person responsible for the execution of the procedure.

Example 9

Let an instance of the credit procedure presented in Figures 2, 4 be in the state $\{b_4, b_9\}$ (Figure 5, a). Then the allowed strongly linear jumps —dashed lines in Figure 5, b)— can either move the process back to the states $\{b_4, b_8\}$, or $\{b_4, b_7\}$, or $\{b_4, b_6\}$, or $\{b_3, b_9\}$, or move the process forward to the state $\{b_5, b_9\}$. In practice —in the state $\{b_4, b_9\}$— the backward strongly linear jumps allow the bank employees to refine the investigation on the client. In other words, when an employee needs additional information, which might have been produced previously in the process, she can directly jump backward and ask to her collegue responsible for one of the previous activities.

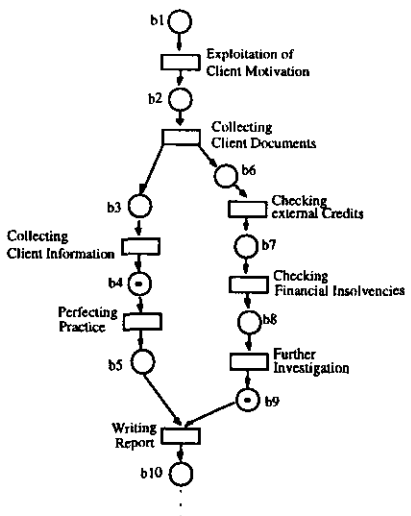


Figure 5 a)

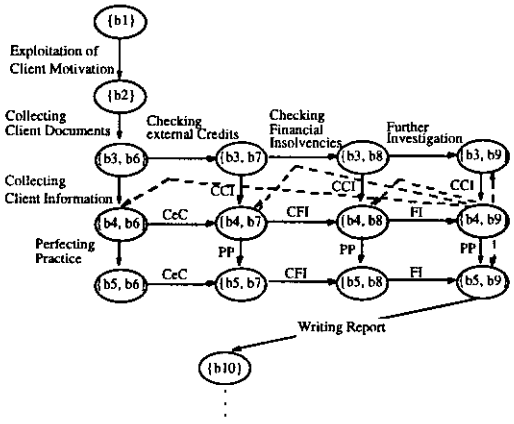


Figure 5 b)

From the same state $\{b_4, b_9\}$ (Figure 5, a), weakly linear jumps —dashed lines in Figure 6— may either move the process back to the states $\{b_1\}$ or $\{b_2\}$, or move forward to all possible states $\{b_{10}\}$ ... $\{b_{19}\}$.

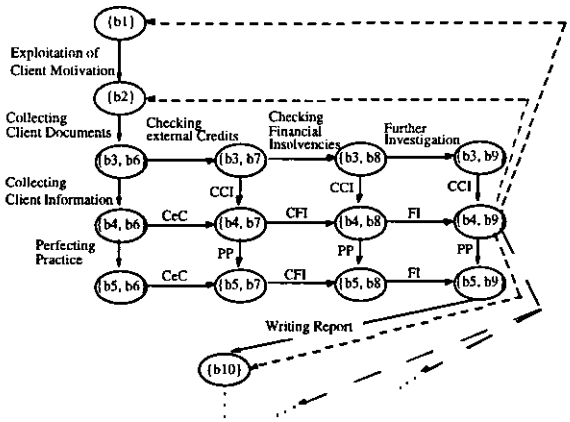


Figure 6

As previously anticipated, within the credit procedure, these jumps allow to handle all cases of rejection or withdraw of the credit, while leaving its definition as simple as possible. For instance, every time the experienced managers —e.g., the agency director, the district coordinator, etc.— intend to reject the request of the credit, it is sufficient to jump forward at the end of the procedure.

To make an additional simple example of the utility of these jumps, every time a well known and very important client of the bank asks for a new credit, the agency director would like to jump directly to the negotiation part of the credit (e.g., the state

$\{b_{10}\}$), or even directly to the signature of the credit contract (i.e., the state $\{b_{18}\}$). Of course, while strongly linear jumps can be applied directly by the employees, weakly linear jumps (as the ones described above) involve the approval of a responsible manager like the agency director.

The modeling framework constituted by the couple (WNM, WSM) is therefore offering various services to its various categories of users. Actors, initiators, administrators and designers can choose between WNM and WSM to have the most effective visualization of the workflow model with respect to their current interest; actors and initiators can analyze the context in which a breakdown occurs choosing how to solve it.

Administrators and/or designers receive from the above modeling framework also some relevant services with respect to their responsibility on the model and on its changes. If we assume that they are free to design the most efficient and/or effective workflow for executing within some constraints characterizing what, anyhow, the procedure must do, then they need to check any change with respect to those constraints. Our modeling framework provides them with some services supporting both change design and its verification with respect to the constraints imposed to the procedure. They can, in fact, define a Minimal Critical Specification (see Definition 10, below) that must be satisfied by the adopted workflow model and by all its changes, using it as a reference to guide changes. The theory embedded in the framework (i.e., the properties of the morphisms between WNMs and/or WSMs) allows it to support them with the automatic verification of the correctness of changes. Moreover, they can enact the change on all the already ongoing instances of the workflow, moving to the new model all the instances that are in a safe state while postponing the enactment of the change in those instances that are in an unsafe state until they reach a safe one (for the definition of safe and unsafe states see Definition 11, below).

These services are based on the following:

- the class constituted by a minimal critical specification together with all the workflows that are correct with respect to it is closed under the morphisms induced by the action-labels;
- the composition of morphisms and inverse morphisms (morphisms always admit inverse, since they are injective and total).allows to distinguish between safe and unsafe states with respect to a given change.

Let us explain the above claim with some simple examples, assuming that any workflow model must have the same set of action labels as its minimal critical specification and that only changes not modifying the set of action labels are allowed.

Definition 10 - Minimal Critical Specification
A WSM, A = (S, E, T, s_{in}), is correct with respect to a minimal critical specification MCS = (S', E, T', s_{in}') if and only if the morphism induced by E, g:S —> S', is injective and total.

As its name evokes and its definition grants, a minimal critical specification is less constraining than any workflow model correct with respect to it, i.e. it admits a larger class of behaviours. Whenever no minimal critical specification is given, it can be assumed that the n-dimensional diamond representing the sequential behaviours of the workflow where all the n actions labels are concurrent is the implicit minimal critical specification to be taken into account.

Definition 11 - Unsafe states with respect to a change

Let A=(S, E, T, s_{in}) be a WSM and A'=(S', E, T', s_{in}') be the a WSM being the effect of a change on it. Let both, A and A', be correct with respect to the minimal critical specification, MCS = (S", E, T", s_{in}"). Let, finally, g: S—>S" and g':S'—>S" be, respectively, their morphisms on MCS induced by E: then S - $g^{-1}(g'(S'))$ is the set of *unsafe* states of A with respect to the given change. If a state is not unsafe with respect to a change, then it is *safe* with respect to it.

S - $g^{-1}(g'(S'))$ contains all the states of A not having an image in S' (the new changed model); therefore it is impossible to move an instance being in one of them to the changed model since we can not find univocally the state in which it will be after the change. Moreover, any choice we do for it, does not allow a correct completion of the process.

Example 12

Let the WSM of Figure 7 (in the cube only the most external edges are labeled, since every parallel arc has the same label) be the effect of a change to the WSM of Figure 4.

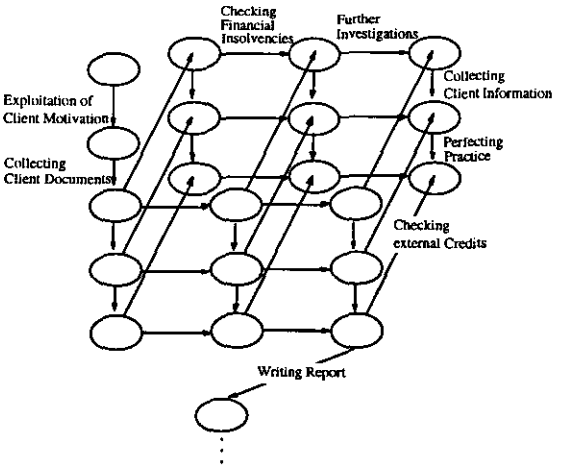


Figure 7

This change makes the procedure more efficient allowing to perform concurrently the activity 'Checking external Credits' and the sequence of activities 'Checking Financial Insolvencies', 'Further Investigation'. In this case all states of the original

procedure are safe states with respect to this change; that is, all running instances can be safely moved in the new model.

Example 13

Let the WSM of Figure 8, b, be the effect of a change of the WSM of Figure 8, a. In this case, the bank decided that possible 'Financial Insolvencies' of the Client should be checked as soon as possible in order not to proceed further and wasting time in 'Checking external Credits' in case of client's insolvency.

Then the three shaded states of the first WSM (Figure 8, a) are its only unsafe states with respect to this change.

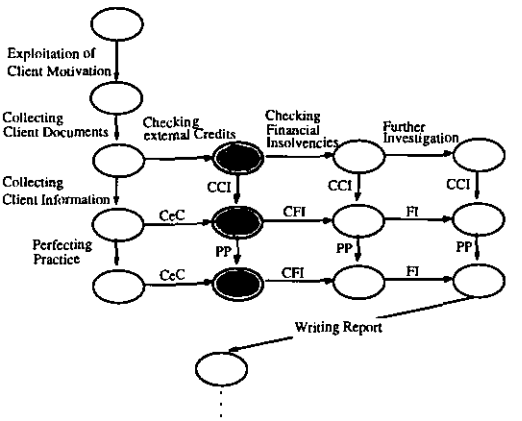


Figure 8 a)

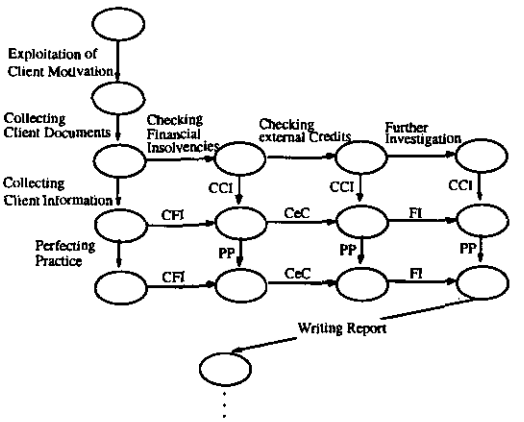
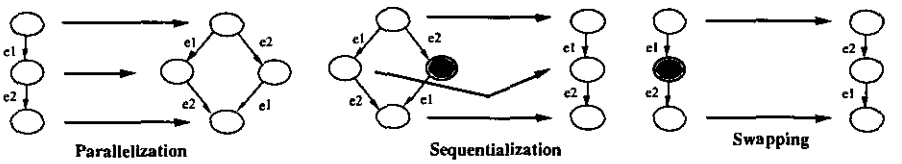


Figure 8 b)

Example 14

Figure 9 summarizes the three patterns of change allowed by our theoretical framework: *parallelization,* making two sequential action labels concurrent (Figure 9, a); *sequentialization*, creating a sequence with two concurrent action labels (Figure 9, b); *swapping*, inverting the order of two sequential action labels (Figure 9, c). The shaded states represent the unsafe states.



a) b) c)

Figure 9

The class of changes introduced above is quite small. An extension of the allowed changes may be obtained weakening the condition that the minimal critical specification contains all the action labels of any workflow model correct with respect to it, to the one imposing only that its action labels are contained in the set of action labels of any workflow model correct with respect to it.

Finally, a precise definition of action-label refinement within the above theoretical framework will further extend the class of changes supported by the specification module of the Milano workflow management system.

Conclusion

The approach we have followed in the development of the modeling capabilities of the workflow management system of Milano is, for what we know, the first attempt to use the synthesis of Elementary Net Systems proposed by Nielsen, Rozenberg and Thiagarajan (Nielsen et al., 1992) and further developed by Bernardinello (1993) to the application domain.

The treatment of dynamic changes we propose is strictly related to the one proposed by Ellis, Rozenberg and Keddara (Ellis et al., 1995). The main difference between them is that while Ellis and co-workers move any workflow instance to a new model, where unsafe states and paths are preserved in order to avoid inconsistencies, in our approach the move of the instances is delayed until they reach a safe state.

Solutions for the static change of workflow models have been developed by several scholars (Abbott, Sarin, 1994; Swenson et al., 1994; Simone et al., 1995; Dourish et al., 1996; Voorhoeve, van der Aalst, 1997). While some of them have a different objective, since they allow any change without any request for consistency

between the old and the new model (Swenson et al., 1994, Dourish et al., 1996), other are based, as is our proposal, on transformation rules granting the desired consistency relation between the two models (Simone et al., 1995; Voorhoeve, van der Aalst, 1997). Generally they propose larger classes of transformation rules than us, since they do not take into account the dynamicity of changes. It is our intention, in any case, to extend our transformation rules to allow also refinements and/or abstractions.

A part from the full integration of the workflow management system within the Milano platform and from the new software modules it needs, some further developments are planned in the Cooperation Technologies Laboratory at the University of Milano.

As already mentioned above, we plan to extend the transformation rules for changing a workflow model to allow also refinements and abstractions. This requires the development of both its theoretical basis and its modeling capabilities.

We want to enrich our modeling framework with a recursive capability, allowing to reduce a part of a workflow model to a single node, if and when it has the required interfaces with the rest of it.

The graphical interface we need for our workflow management system is rather complex, since managing changes and exceptions with respect to multiple different representations requires the automatic generation of a graphical representation on the basis of a formal model. We are experimenting with a graphical editor developed at the University of Rome (Bertolazzi et al., 1995) to evaluate if it fits within our system.

Acknowledgments

The authors thank the anonymous referees of this paper, who helped to improve its quality. Moreover special thanks are due to our students, Roberto Tisi, Paolo Bertona, Pietro Nardella and Mario Manzoli, who greatly contributed to the development of the workflow management system of Milano.

References

[Abbott, Sarin, 1994] Abbott, K. R., Sarin, S. K. Experiences with Workflow Management: Issues for The Next Generation. In *Proceedings of the Conference on Computer Supported Cooperative Work*, ACM, New York, 1994, pp. 113-120.

[Agostini et al., 1994] Agostini, A., De Michelis, G., Grasso, M. A., Patriarca, S. *Reengineering a business process with an innovative Workflow Management System: a Case Study*. Collaborative Computing, 1.3, 1994, pp.163-190.

[Agostini et al., 1997] Agostini, A., De Michelis, G., Grasso, M. A. Rethinking CSCW systems: the architecture of Milano. In *Proceedings of the Fifth European Conference on Computer Supported Cooperative Work*, Kluwer Academic Publisher, Dordrecht, 1997, pp. 33-48.

[Badouel et al., 1997] Badouel, E., Bernardinello, L., Darondeau, P. *The synthesis problem for elementary net systems is NP-complete.* Theoretical Computer Science, 186, pp. 107-134.

[Bernardinello, 1993] Bernardinello, L. Synthesis of Net Systems. In *Application and Theory of Petri Nets*, LNCS 691, Springer Veralg, Berlin, 1993, pp. 89-105.

[Bertolazzi et al., 1995] Bertolazzi, P., Di Battista, G., Liotta, G. Parametric Graph Drawing. In IEEE Transactions on Software Engineering, 21.8, 1995.

[Bowers et al., 1995] Bowers, J., Button, G., Sharrock, W. Workflows from within and from without: technology and cooperative work on the print industry shopfloor. In *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work,* Kluwer Academic Publisher, Dordrecht, 1995, pp. 51-66.

[Brauer et al., 1987] Brauer, W., Reisig, W., Rozenberg, G. (Eds.) *Petri Nets: Central Models and Their Properties*. LNCS 254, Springer Verlag, Berlin, 1987.

[De Michelis, 1995] De Michelis, G. Computer Support for Cooperative Work: Computers between Users and Social Complexity. In C. Zucchermaglio, S. Bagnara and S. Stucky (eds.) *Organizational Learning and Technological Change* (eds.), Springer Verlag, Berlin, pp. 307-330.

[De Michelis, 1997] De Michelis, G. *Work Processes, Organizational Structures and Cooperation Supports: Managing Complexity.* Annual Reviews in Control, 21, 1997, pp. 149-157.

[De Michelis, Grasso 1994] De Michelis, G., Grasso, M. A. Situating conversations within the language/action perspective: the Milan conversation Model. In *Proceedings of the Conference on Computer Supported Cooperative Work,* ACM, New York, 1994, pp. 89-100.

[Ellis et al., 1995] Ellis, C, Keddara, K., Rozenberg, G. Dynamic Change within Workflow Systems. In *Proceedings of the Conference on Organizational Computing Systems.* ACM Press, New York, 1995, pp. 10-21.

[Koulopoulos, 1995] Koulopoulos, T. M. *The Workflow Imperative.* Van Nostrand Reinhold, New York, 1995.

[Nielsen et al., 1992] Nielsen, M., Rozenberg, G., Thiagarajan, P.S. *Elementary Transition Systems*. Theoretical Computer Science, 96, 1992.

[Norman, 1991] Norman, D. A. Cognitive Artifacts. In Carroll J. M. (ed.) *Designing Interaction. Psychology at the Himan computer Interface.* Cambridge University Press, Cambridge, 1993, pp. 17-38.

[Rozenberg, 1987] Rozenberg, G. Behaviour of Elementary Net Systems. In: [Brauer et al. 1987], pp. 60-94.

[Schael, Zeller, 1993] Schael, T., Zeller, B. Workflow Management Systems for Financial Services. In *Proceedings of the Conference on Organizational Computing Systems.* ACM, New York, NY, pp.142-153.

[Simone et al., 1995] Simone, C., Divitini, M., Schmidt, K. A notation for malleable and interoperable coordination mechanisms for CSCW systems. In *Proceedings of the Conference on Organizational Computing Systems.* ACM Press, New York, 1995, pp. 44-54.

[Suchman, 1987] Suchman, L. A. *Plans and Situated Actions. The Problem of Human-Machine Communication.* Cambridge University Press, Cambridge, 1987.

[Swenson et al., 1994] Swenson, K. D., Maxwell, R. J., Matsumoto, T., Saghari, B., Irwin, K. *A Business Process Environment Supporting Collaborative Planning.* Collaborative Computing, 1.1, pp. 15-34.

[Thiagarajan, 1987] Thiagarajan, P. S. Elementary Net Systems. In: [Brauer et al. 1987], pp. 26-59.

[Voorhoeve, van der Aalst, 1997] Voorhoeve, M., van der Aalst, W. Ad-hoc Workflow: Problems and Solutions. In *Proceedings of the 8th International Workshop on Database and Expert Systems Applications*. IEEE Computer Society, California, 1997, pp. 36-41.

[White, Fischer, 1994] White, T. E., Fischer, L. (Eds.) *The Workflow Paradigm,* Future Strategies, Alameda, 1994.

[Winograd, Flores, 1986] Winograd, T., Flores, F. *Understanding Computers and Cognition.* Ablex, Norwood, 1986.

[Winograd, 1988] Winograd, T. *A Language/Action Perspective on the Design of Cooperative Work.* Human Computer Interaction, 3.1, 1988, pp. 3-30.

[WMC, 1994] Workflow Management Coalition, *Coalition Overview.* TR-WMC, Brussels, 1994.

The Formal Representation of Call Processing in Call Centers Using a Petri Net Approach

Nikolay Anisimov[1,2], Konstantin Kishinski[1], Alec Miloslavski[1],

[1]Genesys Telecommunication Laboratories, Inc.
1155 Market Street, San Francisco, CA, USA 94103
E-Mail: {anisimov,kotc,alec}@genesyslab.com

[2]Institute for Automation & Control Processes,
Russian Academy of Sciences
5 Radio St., Vladivostok, 690041, Russia

Abstract. In this paper we apply a formal approach, based on Petri nets, to design a logical structure for call centers based on sophisticated computer telephony integration (CTI) applications. A typical call center consists of a set of operators, called agents, who process inbound calls from clients. This call processing may involve the use of computer systems and other devices, such as faxes, as well as communication with other agents. The treatment of each call being processed is heavily regulated by a script, which is specially designed for specific kinds of calls by the experts in telemarketing. However, the design of such scripts can be problematic. In this paper, we stress the need for tools supporting a scripting process. We propose a formal model intended to serve as a basis for such tools. Specifically, we introduce formal models called script nets for formal representation of scripts and of the call center as a whole. We have also introduced various ways to structure script nets, using a transition hierarchy and macroplaces.

1. Introduction

Over the last few years phone call-center systems have continued to grow at a remarkable pace. Several manufacturers and service providers are developing and introducing systems with enhanced functionality, principally through what is known as computer-telephony integration (CTI) [15]. The general purpose of a call center is to connect operators called *agents* with members of the public called *clients*, i.e., people interested in using the services of the call center. Typically a call center is based on at least one telephony switch to which agent stations are connected by extension lines and directory numbers, and to which incoming and outgoing trunk lines may carry telephone calls between the switch and the parties who call in. In addition, most modern high-capacity

call centers have agent stations that include computer platforms, often PCs, equipped with video display units (VDUs). The PC/VDU platforms are typically interconnected, usually by a local area network (LAN). There may also be servers of various sorts (e.g. data base or fax server) for various purposes on the LAN, and the LAN may also be connected to a CTI server, in turn connected to the central switch through a CTI link.

Within a call center, agents process telephone calls from clients and carry out call-related business.

The typical processing of calls includes using data from computer systems, including databases; incorporating other devices such as fax and e-mail; and communication with other agents. The communication of the agent during call processing is heavily directed by a specific scenario, specially developed for such calls by telemarketing experts. These scenarios are referred to as *scripts*. The same agent can work with varying call types, controlled by different scripts. Thus, a call center is a distributed system, usually built on top of a local area network that connects agent stations, server computers and telephone equipment.

It is interesting that concept of workflow management [1,6] can be very useful in designing call centers. In fact, a call center can be understood as a specific case of a workflow system, which substitutes telephone calls for documents circulating in the system. We should also mention that because office activity very often involves working with inbound and outbound calls, the processing of such calls should be naturally incorporated into workflow management systems.

The present paper is devoted to call center management, and is specifically directed toward scripting for call centers. Usually, scripts are written in a relatively high-level programming language. The complexity of a call center presents a challenge for any programming tool. Moreover, as with any other sort of programming, when a bug appears or a change is made in the purpose or operation either of a call center or a segment of call center operations, it is often necessary to rewrite a large number of scripts. This endeavor is no small task, and may take a considerable time. Moreover, such reprogramming introduces numerous opportunities for errors, both in programming and in the layout of the script.

Given the nature of call center management, and scripting in particular, it is highly desirable to reduce the complexity and amount of effort required to direct these activities. It is especially important to simplify the activities of agents, such as engagement with clients, and to provide enough flexibility so that changes and adaptations can be easily and quickly made without fear of error. To handle this issue we need special tools, such as a visual language, graphical editor, and others. Essentially, the requirement is to build a platform for a generation of CTI applications of varied types. Such tools for the generation of CTI applications already exist, but for the most part they do not take into account the distributed nature of call centers and therefore do not allow the production of scripts with complex communications between agents, hardware and other resources. In this paper we present the progress we have made in an ongoing project aimed at designing such a platform.

If we examine scripts of a typical call center, we see that their key features include the flexible use of resources during call processing; extensive manipulation of calls, including attached data; allowance for exceptions; complexity of real-world scripts;

parallel call processing; and strict requirements for real-time call processing. It is clear that scripting tools should be designed according to a formal approach. This paper is devoted to developing such an approach, using the theory of Petri nets [11,12]. More specifically, in this paper we build a Petri net-based formal model for representing call center scripts.

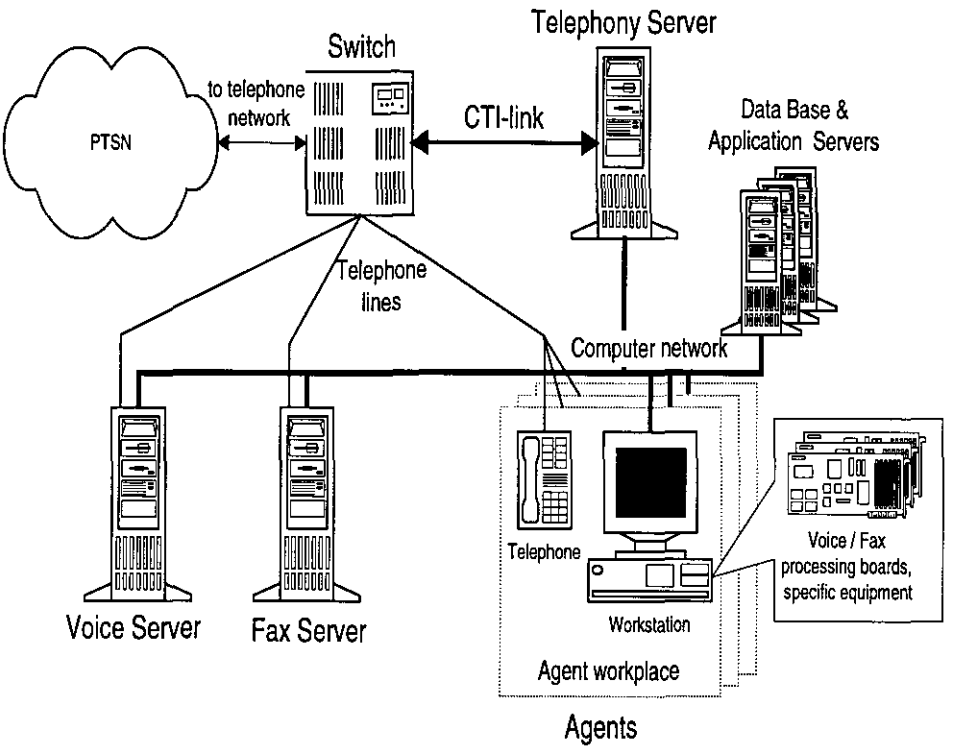


Figure 1: A call center environment

Structure of a call center

In this section we present an abstract model of a typical call center that will serve as a subject for the formalization process. From now on, the call center will be referred to as a "system ".

Typically, a system operates with a set of resources. These are: equipment (e.g. phones, fax machines, switches, a local area network, etc.), software components (database, text editor, etc.) and personnel involved in system operation (agents, administration). All communications of the system is accessed through these resources. A typical Call-Center environment is shown on Figure 1.

From the point of view of applications, the system can be perceived as a collection of communicating objects. We will divide all objects of the application level into two types: resource objects and call objects. The former represents objects corresponding physical resource of the system while the latter represents objects intended for call processing.

The behavior of each object is regulated by a scenario specification, called a 'script'. There may be several objects working in accordance with one and the same script. For example, for a script describing the behavior of a telephone, there may be several objects corresponding to actual telephones in the system; a script specifying the call processing, may have several objects processing different calls of the same type.

We will assume that each script is identified by a unique name within the system. Moreover, we associate with each script a domain of object names, to identify each object within the script. This addressing scheme allows us to uniquely identify objects within the whole system.

3. Formal Model

For formal specification of scripts of CTI-applications, we developed a Petri net-based model called *script-net* [3] combining some featured from other Petri net models [11,12]. The model consists of the following four (quite orthogonal) constituents:

1. High level and object oriented Petri net model called *cooperative nets* [14] that allows to represent complex system as a set of subsystems communicating via the client server protocol [13];
2. For structured specification of complex scripts, we suggest the concept of hierarchical transition [8]. Under this concept, a net can be represented as a set of disjointed subnets with links between hierarchical transitions and subnets forming a hierarchical structure. Firing any hierarchical transition results in execution of its internal net. This construction makes script representation modular and allows for the simple modification and reuse of specifications.
3. For processing exceptions in scripts, such as receipt of unsolicited events, we suggest a macroplace construction [2].
4. To represent real-time constraints that are very critical for scripts, we incorporate Merlin's time constructs into our model [10].

In the following we consider some of these constituents in detail and show how they can be used.

3.1. Basis of script-nets

As a basis of script-net we take an object-oriented model of high level Petri nets known as *cooperative nets* [14] that allows us to represent a system as a set of communicating subnets. In particular, each transition can be associated with the process of communication (sending or receiving of message) with other script-nets:

- Sending a command: $s(script(v).com(v_1,\ldots,v_n))$, where $script(v)$ specifies a target object, and $com(v_1,\ldots,v_n)$ is a command with parameters.

- Receiving a command: $r(script(v).com(v_1,\ldots,v_m))$, that gives a command with parameters from the object $script(v)$.

This enables us to specify a communication between a CTI application and a server using a client-server protocol [13]. Moreover, it is possible to associate a transition with a creation of new objects for some script-net.

- Creating a new object: $c(script(v).v_1,\ldots,v_k)$, where $script(v)$ identifies a creating object and $v_1,\ldots,v_n$ its initial parameters.

Thus each script of a CTI application can be described as a corresponding script-net. In this case the process of call processing can be understood as creating an object (injecting a token in head place of script net) and moving it through the net. Some examples of script-nets are presented in [3,4].

We allow multilabeling of net [5], i.e., labeling where each transition may be labeled by a set of expressions, such as sending and receiving a message, or creating a new object. This extension can simplify specifications and make them more compact.

By collecting communicating script-nets, we can produce a script system that represents a call center's logical structure. In this structure we can distinguish application scripts and system scripts representing system services such as resource management and call routing.

3.2 Macronets: exception handling

At this point, we should note that scripts describing real scenarios are usually extremely complicated to work with and therefore require some means of modularization. We will consider the problem of structural representation of script nets. In this respect, we can point out two techniques for modularization in Petri net-based models we would like to employ – hierarchical transitions and macroplaces. The first technique is well elaborated within the framework of high-level Petri nets, e.g. see [8]. Generally, it consists of representing a hierarchical net as a set of disjoint subnets with links between transitions and subnets forming a hierarchical structure. Firing of such a hierarchical transition causes an execution of its internal net that consists of the firing of a transition (or step) sequence from initial marking to the terminal one. So using this technique we can represent script nets as a set of hierarchical organized script subnets.

At the same time, in call processing, we may face situations, which are asynchronous to normal processing, and a reaction to such events should also be specified. For example, there may be situations when, during the dialogue between agent and client, the telephone line is disconnected (e.g. suddenly client puts down a receiver); as well as more sophisticated situations when the processing of current calls is interrupted and the agent is forwarded to process new calls with higher priority. Moreover, processing of such broken calls could be recommenced upon availability of agents. To specify such situations in script nets, special constructs are needed. To accomplish this, we suggest using the concept of macronets reported in [2] and generalized on high level Petri nets [4].

Petri nets with macroplaces. Notions of macronets and macroplaces have been introduced in [2] for specification of such situations where starting the execution of one procedure may interrupt execution of another procedure. Syntactically, a macronet is

defined similar to nets with hierarchical transitions, however we use macroplaces instead of transitions. In other words, a macronet could be perceived as a set of Petri nets equipped with hierarchical links of the type "*place* → *net*".

Graphically, a macronet can be represented as a set of included nets, each internal net being drawn within a circle of corresponding macroplaces. The head place of an internal net is marked by an incoming extra arc.

The firing rules of macronets are as follows:

- A macroplace is considered to have a token if its internal net also has a token;
- adding a token to a macroplace results in adding a token to the head place of the internal net;
- removing a token from macroplace results in removing a token from the internal net no matter what position it is in.

It is clear that the concept of a macroplace is helpful for representing various situations where an interruption is involved.

High level macronets. Using standard possibilities of high-level nets we generalize the notion of macroplaces, allowing us to specify more general constructions. First, we will be able to build constructions where execution of an internal net can be interrupted only in specified regions. Second, we can specify a head place of internal nets dynamically, helping us to inject a token into any desired place.

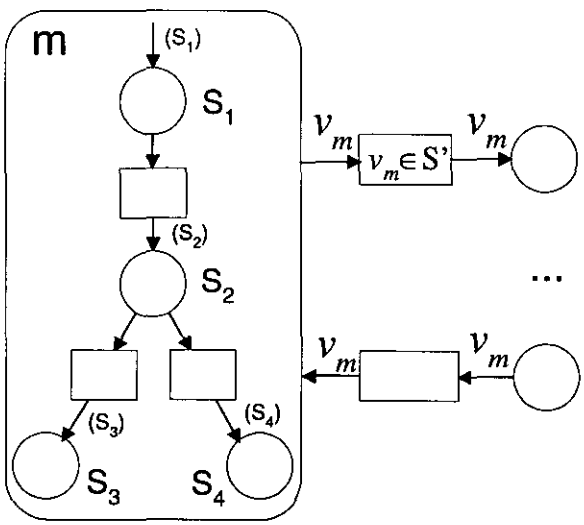


Figure 2: Macroplace and internal net

Let m be a macroplace and $N_m=(S_m, T_m, F_m)$ be its internal subnet. For an internal net we introduce a data type $type_m$ with a domain equal to a set of internal places: $Dom(type_m)=S_m=\{s_1,...,s_n\}$. Let us add to the token internal net an item of type $type_m$, the value of the item is exactly equal to the place where the token is situated. This can be easily implemented by assigning to tuples of incoming arcs with corresponding values

from S_m, see Figure 2. Then we add to this a precondition of outgoing transition $t \in m^\bullet$ an expression of the type $v_{m} \in S'$, where $S' \subseteq S_m$. Firing of the transition t results in removing a token from a place of S'. Note that we can 'remember' the actual place where the token was before it had been removed. For an arc coming in to the macroplace m, the corresponding item of the tuple is stated in a suitable manner. For instance, if it is equal to a place $s_i \in S_m$ the adding of a token to macroplace m will result in injecting a token into s_i of the internal net. Apart from a constant, we can write a variable of the type $type_m$ that allows us to determine the incoming place dynamically. Specifically, we can replace taken to its point of origination, see Figure 2. More strict definition of high-level macronets can be found in [3].

With the aid of a macroplace, one can easily specify the next situation in an agent's scenario:

- Interruption of a script execution (naturally with an apology to a client) at any stage with subsequent return to an initial state. In this case the processing of the interrupted call is cancelled
- Interruption of a script execution only if it is in special regions of the script.
- Interruption of a script execution while noting the place of interruption and possible current parameters of the call processing. This information can be used for future recommencing of the processing of the call.

3.3 Time constraints

To represent real-time constraints that are very critical for scripts, we incorporate Merlin's time constructs [9] into our model. In particular, each transition in a script-net can be associated with a pair $[t_{min}, t_{max}]$ that provides a time interval enabling the transition. This enables us to specify timeouts in script execution.

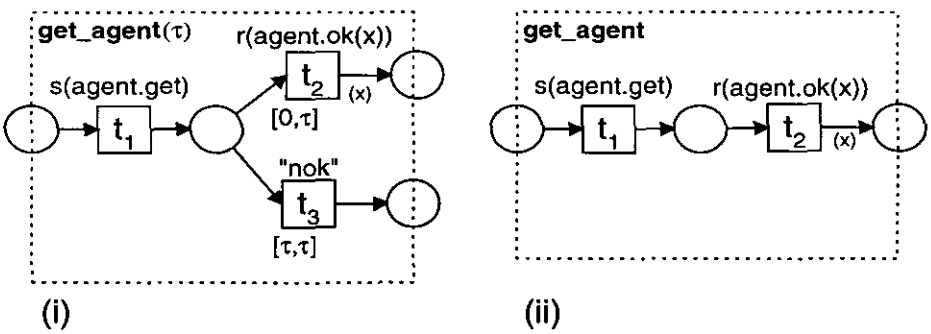


Figure 3: Resource capturing

Using this client server scheme of communication and time constraints, we can build a mechanism of resource capturing/release. On Figure 3 (i) the scheme of agent capturing within time interval τ is depicted. The capturing is started by sending c command ***get*** (firing of the transition t_1) to the script ***agent*** that defines the behavior of the agents. If within the time interval $[0,\tau]$ a positive reply is received ***r(ok(x))*** (firing t_2) then the agent with that identifier x is considered to be captured. If within time interval nothing happens

then transition t_3 fires hence there are no agents available. This construction is called ***get_agent(τ)***. If no time interval is specified then the transition t_3 will never fire and thus can be removed, see Figure 3 (ii).

4. Examples

In the section we discuss a methodology of representing scripts according to the model we are introducing, with the aid of some realistic script examples.

Example 1. On Fig.4 the script corresponding to resource of operators (agents) is depicted. The agents can be in three states: READY, BUSY, NOT-READY (NR for short). The transition from READY to BUSY is caused by receiving a command ***get*** from an object ***X*** and sending it a reply ***ok***. Note that this transition is labeled by two labels that correspond to receiving and sending commands. In a BUSY state the agent can return to a READY state by receiving a command ***r(X.free)*** from the application. Moreover, in a BUSY state the agent can move to a NOT-READY state (e.g. switching to more urgent work) informing the application by sending a command ***not_ready***. The place *S* says that the operator *a* can work with script *A*, the operator *b* can work with script *B* and *c* can work with both scripts *A* and *B*.

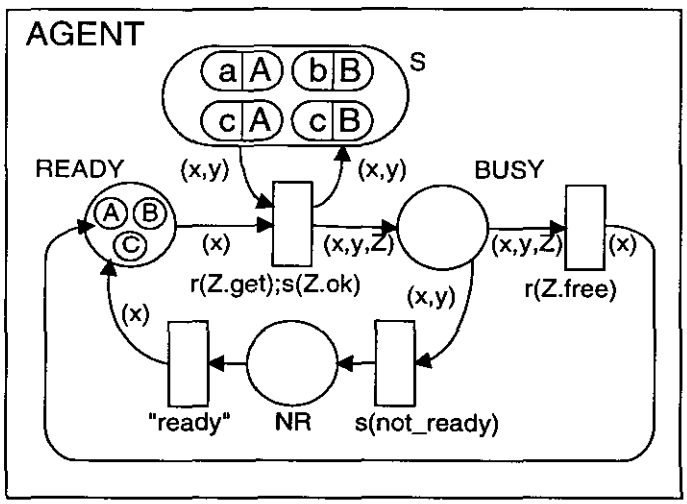


Figure 4: Script-net of agent.

Example 2. In this example we build a script-net corresponding to a script for the processing of a retail catalogue sales call center [9]. In this context, many customers call to inquire about the availability of items in a catalogue, status of their order, delivery options, and similar routine questions. Such simple calls can be processed automatically by a voice processing system. Other calls, especially those involving new customers who need special assistance, must be processed by an operator. In Figure 5, we present a script

net corresponding to processing this type of call. When the system enters a call into a script (a token appears in a head place s_0), it plays a greeting and gives a choice of pressing "1", "2", or "3" (the transition t_1). These choices correspond to calls concerning availability of items, the status of a current order, or other types of calls, respectively.

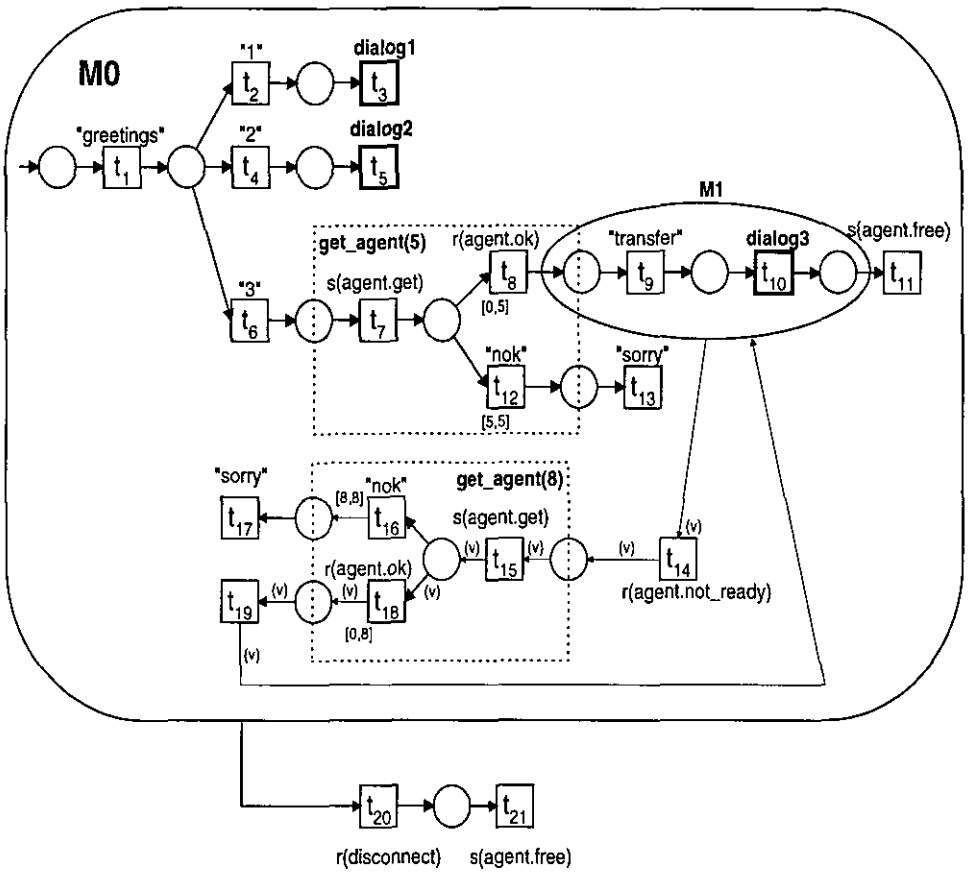


Figure 5: Example of a Script-Net, in Catalogue Sales Context

In the first two cases the call is processed automatically (hierarchical transitions t_2 and t_4). The third case needs the intervention of an operator, who is captured by the expression ***s(agent.get)***. Here, the agent is the name of the script-net for resources corresponding to operators, get the name of capturing command. If there is a free operator in the system, he is captured. At this point the call is transferred and the operator works with the customer (hierarchical transition t_{10}). At the end of the conversation, the operator is released (t_{11}). If there are no free operators (t_{12}) the system plays a recorded sound file with appropriate explanations.

In this example, two possibilities for agent capturing within the time interval τ are shown within dashed boxes. The capturing is initiated by sending a ***get*** command to the "agent"

script that defines the behavior of the agents. If a positive reply ***r(ok)*** is received within the time interval represented by $[0,\tau]$ then the agent with that identifier is considered to be captured. If within the time interval nothing happens, then transition ***nok*** fires, indicating that no agents are available. This construction is called ***get_agent(τ)***.

Imagine that an agent involved in call processing presses a "not ready" button on his telephone and becomes unavailable. This event corresponds to firing the transition t_{14}. At this point the script tries to find another agent (t_{15}). If another agent is indeed available, control of the script is returned to the same place where it was interrupted, and call processing continues. The state where the call processing was interrupted is saved in the variable v.

Alternatively, imagine that a client suddenly hangs up during a call. This event corresponds to firing the transition t_{20}. Firing of this transition disrupts the execution of the script including the construction defined above, releases its active agent (if any), and then terminates the call processing.

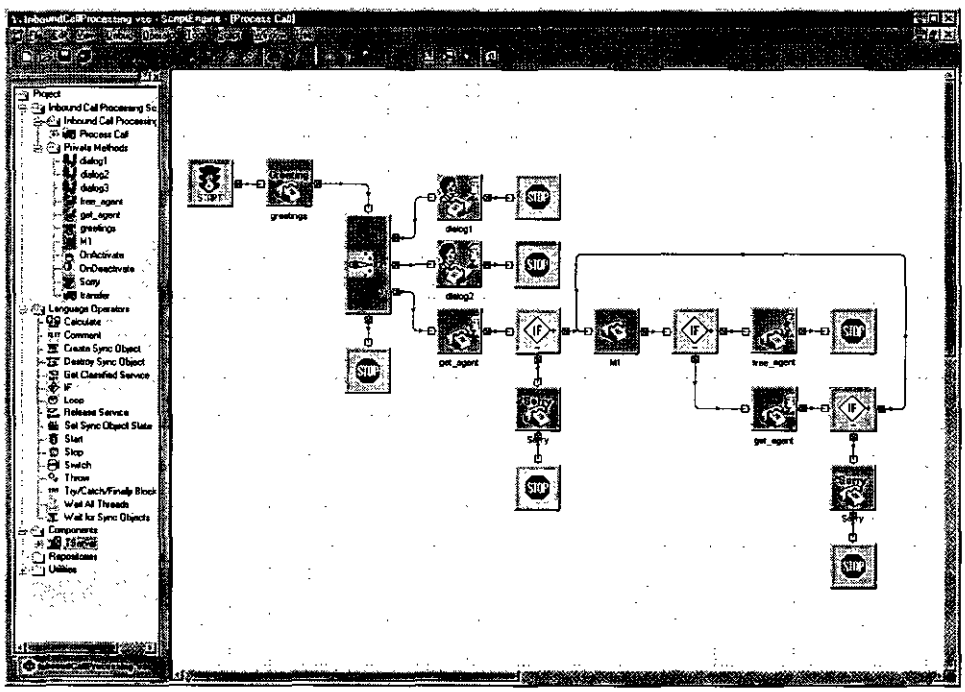


Figure 6: Graphical script editor

4. *Implementation Issues*

In this section we briefly sketch some tools for generation call center applications based on developed formal model. In particular, we developed a programming system for

building scripts for processing both inbound and outbound calls with extensive intervention of living agents. The system functions in an Intranet environment, is based on thin-client technology, and uses a graphic language to describe agent work scripts. More specifically, the system comprises the following components:

- A front-end graphical language for specification scripts with semantics based on Petri net-based model;
- A graphic script editor that supports the scripting language and allows to build scripts in a simple and convenient way;
- Form manager for creating a set of forms to be interchanged between application and agent station during a call processing;
- A script engine that executes scripts upon emerging inbound and outbound calls in the run-time stage.

The graphical scripting language allows one to represent a script as a graph where each node depicted by icon corresponds to elementary communication with other objects (e.g. devise object, agent). Arrows between icons define a causal relation between communication actions. Among other features of the language, we can mention features which inherited from the formal model:

- Parallel constructs allowing one to represent multithreading and synchronization between threads;
- Hierarchical constructs which allow one to build scripts in a modular fashion;
- Exception handling constructs which enable one to specify the reaction of the scripts on receiving asynchronous and unsolicited events.

On Figure 6, we present the snapshot of the script editor with the fragment of the script discussed in the example 2.

5. Concluding Remarks

In this paper we have proposed the Petri net-based model for design different distributed CTI applications. This approach enables a designer to represent a logical structure of complex applications for call centers.

In the nearest future we plan to pay more attention to architectural aspects in the process of formalization taking into consideration different related architectural approaches [7].

Considering almost all CTI-applications are real time systems, we must take into consideration time and stochastic aspects of the model under discussion, i.e., it is desirable to calculate availability of a call center (for specific sort of calls), agent's loading, optimal configuration of call-center, etc. By extending our model towards stochastic Petri nets, these issues can be addressed.

Acknowledgements. The authors are grateful to Evgeny Petrovikh and Pavel Postupalski for they helpful discussions. The first author has also benefit from a grant from the Russian Fund for Basic Research No. 96-01-00177.

REFERENCES

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. To appear in the Journal of Circuits, Systems and Computers (1998).

2. N.A.Anisimov. An Algebra of Regular Macronets for Formal Specification of Communication Protocols. Computers and Artificial Intelligence, Vol.10, No.1 (1991), pp. 541–560.

3. N.Anisimov, K.Kishinski, A.Miloslavski. Petri Net Based Model for Design of CTI-applications, in Proc. of the Int. Conference "Computational Engineering on Systems Application: CESA'96", July 9-12, 1996, Lille, France.

4. N.A.Anisimov, K.P.Kishinski, A.Miloslavski, P.A. Posupalski, Macroplases in High Level Petri Nets: Application for Design Inbound Call Centre, In: Proc. Int. Conference on Information Systems Analysis and Synthesis (ISAS'96), Orlando, Florida, USA (July 22-26, 1996), pp.153-160.

5. N. Anisimov, M. Koutny. On Compositionality and Petri Nets in Protocol Engineering. In: Protocol Specification, Testing and Verification, XV. Chapman & Hall, pp.71-86, 1996.

6. C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In: M.Ajmone Marsan (ed.), 14th International Conference on Application and Theory of Petri Nets 1993, Lecture Notes in Computer Science, Vol. 691, pp.1-16. Springer-Verlag, Berlin, 1993.

7. Enterprise Computer Telephony Forum, S.100 Revision, Media Services "C" Language, Application Programming Interface, 1996.

8. K.Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1: Basic Concepts, EATCS Monograph on Theoretical Computer Science, Springer Verlag, 1992.

9. E. Margulies. Voice Processing Applications Flatiron Publishing, 1995, ISBN 0-936648-70-8

10. P.M.Merlin, D.J.Farber. Recoverability of Communication Protocols — Implication of a Theoretical Study. IEEE Trans. Commun. COM–24 (1976) 1036–1043.

11. J.L.Peterson. Petri Net Theory and the Modelling of Systems, (Prentice–Hall Inc., 1981)

12. W.Reisig. *Petri Nets: An Introduction.* EATCS Monograph on Theoretical Computer Science (Springer–Verlag, 1985).

13. C. Sibertin-Blanc, A Client-Server Protocol for the Composition of Petri Nets. . In: M.Ajmone Marsan (ed.), 14th International Conference on Application and Theory of Petri Nets 1993, Lecture Notes in Computer Science, Vol. 691 (1993) pp. 377-396

14. C. Sibertin-Blanc, Cooperative Nets, In: R. Valette (ed.) 15th International Conference on Application and Theory of Petri Nets 1994. Lecture Notes in Computer Science, Vol.815, Springer Verlag (1994), pp.471-490

15. R.Walters. Computer Telephone Integration, Artech House (1993)

Appendix A: Basic notions

A net is a tuple $N = \langle S, T, F \rangle$ where $S = \{s_1, s_2, \ldots, s_n\}$ is a set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is a set of transitions such that $S \cap T = \varnothing$, $F \subseteq S \times T \cup T \times S$ is a flow relation. For each $t \in T$ define its a pre-set of places as ${}^\bullet t = \{s | (s,t) \in F\}$ and a post-set $t^\bullet = \{s | (t,s) \in F\}$. Analogously, ${}^\bullet s = \{t | (t,s) \in F\}$ and $s^\bullet = \{t | (s,t) \in F\}$. A marking of a net N is a function $M : S \to \{0,1,2,\ldots\}$. A Petri net is a tuple $\Sigma = \langle N, M_0 \rangle$ where N is a net and M_0 is an initial marking.

Parameterized Petri nets for modelling and simulating human organisations in a workflow context

Emmanuel Adam[(1,2)*], René Mandiau [(1)], Emmanuel Vergison[(2)]

[(1)]LAMIH – URA CNRS 1775, Université de Valenciennes et du Hainaut Cambrésis,
Le Mont Houy - B.P.311 - 59304 Valenciennes Cedex - FRANCE
rene.mandiau@univ-valenciennes.fr

[(2)]SOLVAY Research and Technology, SOLVAY S.A.
Rue de Ransbeek, 310 - 1120 Bruxelles - BELGIUM
emmanuel.adam@solvay.com, emmanuel.vergison@solvay.com

Abstract. In the past few years, the demand from large organisations for computer supported co-operative work systems has increased noticeably. Moreover, most organisations are so complex that it is hard to offer them off-the-shelf solutions. In this paper, we suggest a perspective for tackling human organisation issues based on a requirement analysis coupled to a computerised modelling process which is defined by Parameterized Petri nets. Our approach has been thoroughly tested in the Patent Department of a large company.

Keywords. Parameterised Petri nets, Workflow, CSCW, modelling, simulation.

* : *PhD student whose thesis is jointly sponsored by the Nord-Pas de Calais Region (France) and SOLVAY S.A.*

1. Introduction

Today, most companies and organisations are seeking for methodologies and tools capable of helping them to make appropriate management decisions. In these organisations, the workforce may vary from a few up to tens of thousands of people and, the range of skill and knowledge levels is usually quite large. More than ever, co-operative work turns out to be a compelling way to reach project goals and company objectives effectively and in due time.

Methods and tools allowing for better co-operative work practices have emerged recently as for example the CSCW approach ((Scrivener, 1994), (Bowers & Benford, 1991), (Connoly & Edmonds, 1994)), and efficient operational software appeared on the market (Lotus Notes, Microsoft Exchange, Novell's and Digital's solutions).

Although they seem to be well suited to solve problems where just a few people are just interacting in a more or less isolated process and workflow issues, they are not appropriate for large complex systems at least without preliminary, sound and detailed analysis and modelling steps (Schael, 1997).

The purpose of this paper is to present a way to tackle human organisation problems based on both an analysis and a modelling exercise. It is based on a real case study, which will be exposed in detail. We will conclude this paper by discussing some research perspectives.

2. Requirement analysis in a complex human organisation: the context

In general, a complex organisation is hierarchical and composed of semi -autonomous subsystems ((Koestler, 1969), (Adam et al., 1997)). Our study is focused on the complex human organisation within a co-operative workplace, where management of people is a crucial issue.

Moreover, many organisations today strongly expressed a need for optimising their management processes and activities by using appropriate computerised means.

By questioning and interviewing people, we already can already obtain a local but rather patchy solution. To identify real needs in complex organisations, it is necessary to analyse them in detail and to define a rigorous method. Human activities and also human factors characterising co-operative work are to be made explicit. The representations issued from this modelling exercise will serve as a base for discussion with the human actors within the different processes; their goals are to define specifications for new management solutions.

3. Analysis and modelling approach

Our case studies were conducted in an Industrial Patent Department of a large Company. The workforce in this department is comprised mainly of experts in patents and trademark issues. Most of these people have no background in computer science.

We initially a quite some time analysing organisational practices. Then, we looked at a method that helped us to identify its intrinsic structure. For this purpose, we have analysed representative methods such as MERISE (Tardieu et al., 1991), OMT (Rumbaugh et al., 1991), OSSAD (Dumas & Charbonnel, 1990), SADT (IGL, 1989), MKSM (Ermine, 1995). Most of these methods have a data model, an activity model and a processing model. For our case study, we needed a detailed data modelling (such as OMT or UML one). We also needed a representation of the actor level in the organisation (such as in OSSAD), and it was necessary for us to follow data flows between system actors.

After having shown that no method completely fit with our objectives (Adam et al., 98), further to this set of methods, we propose one for the modelling and the simulation of human organisation.

3.1. The analysis phase

The analysis of the organisation and the understanding of its practices are key points in our approach.

To properly identify its needs, at general levels, it was necessary to perform a detailed analysis of the human activities within the department. On the other hand, in order to assess the needs at specific levels, an analysis of the individual tasks was also necessary; the techniques we used are quite conventional: observations, questionnaires, interviews, protocol analysis, document analysis... ((Diaper, 1989), (Wilson & Corlett, 1990), (Kolski, 1997)).

It is also worth noting that nothing would have been possible without the support and the agreement of the different actors.

3.2. The modelling phase

The modelling phase started by considering and defining the basic «raw material» of the Patent Department (the document) and by setting up a data model. We therefore chose an object model, which allows a clear representation for the data (especially a hierarchical link between types of documents). An object model not only allows the representation of static data (for example paper documents integrated in the organisation data flow), but also dynamic ones (for example electronic documents). We opted for the OMT object model (Rumbaugh et al., 1991) because it satisfies our «readability» criteria. (Fig 1(a)).

After the data modelling exercise, we started modelling the data flows on the basis of the activity analysis which has been described: a global view of working mechanisms in the organisation has been exploited by using the Actigram model of SADT (IGL, 1989) (Abed & Angue, 1994). Moreover hierarchical and/or responsibility concepts had also to be introduced; we drew inspiration from OSSAD (Dumas & Charbonnel, 1990) and we represented responsibility levels in rows. (Fig 1(b)).

Based on the data and data flow models, we then tackled data processing which required more detail than activity models. There are few processing models oriented towards human organisations. OSSAD is one of them, it proposes a model able to represent both co-operative and hierarchical concepts. (Fig 1(c))

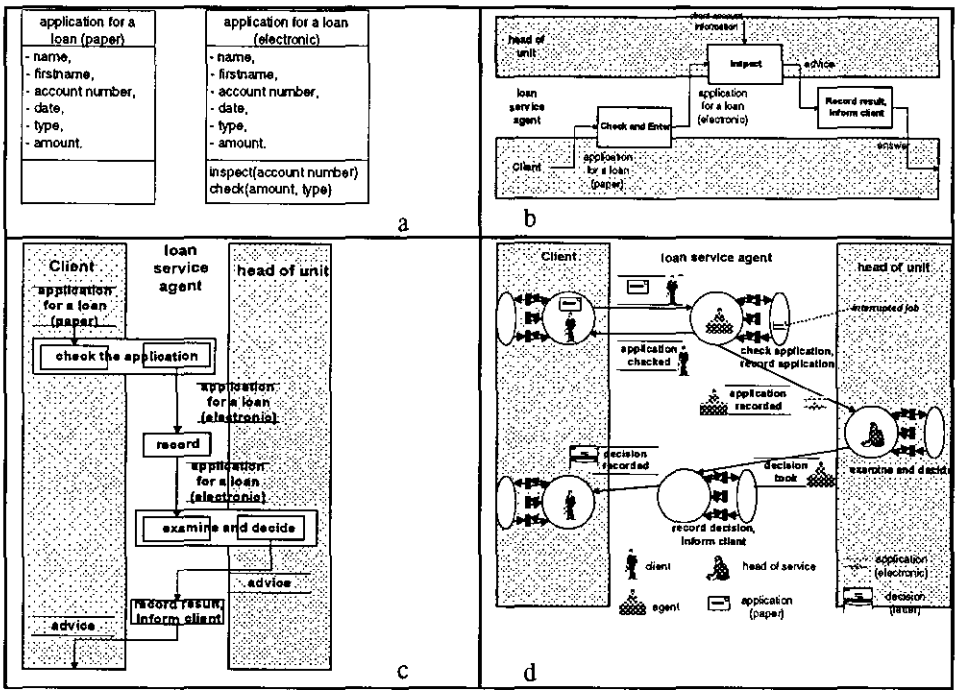


Fig. 1. Four steps for modelling an organisation (example inspired from (Dumas, 1990)

(a) Object model of a paper and electronic application
(b) Activity model of an application for a loan
(c) Process model of an application for a loan
(d)Dynamic model of an application for a loan: the parameterised places allow us to easy represent interrupted job and state change (in the second place, the agent memorises his job when client interrupts him)

3.2.1. Modelling the Dynamics: choice of the Petri Net

Three models have been proposed so that most people (including those NOT conversant with computer science) can easily use them, the first one for modelling data, the second for modelling data flow and one for modelling data processing.

These models, however, are still not sufficient to model the dynamics of human organisations, which have to take into account interruptions, parallel work and loops (Reason, 1990), (Jambon, 1996).

Several authors (such as (Abed & Angue, 1994), (Jambon, 1996), (Palanque 92)) suggested the Petri net formalism.

We have chosen a parameterised Petri net (Agimont, 1996) (Gracanion et al, 1994), which allows for interrupt, parallel and/or synchronised data management. (Fig 1(d) for a better readability, the net is drawn in time)

The four models introduced to date have to be discussed with the staff concerned (during meeting and brainstorming) in order to identify the critical points within the processes and to set up new solutions in a collaborative way.

3.2.2. Choice of Parameterised Petri Net

The Petri network has to allow the follow-up of system data flows, as well as actors' flows; especially those who move from one office to another. This network must also allow identification of document state and actors' activities. In particular, it has to be able to represent interrupts, which are inherent in human organisations. Classical Petri net does not give sufficiently clear representation of these requirements (Fig.2 & 3).

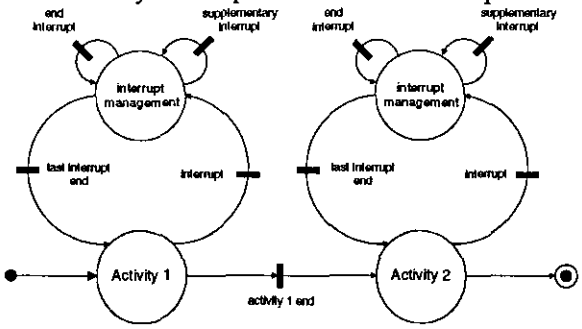


Fig. 2. PN where place represents activities, e.g. the actor going from place to place

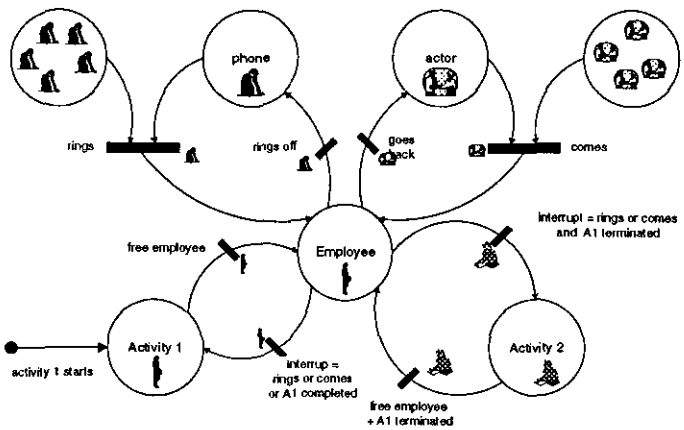


Fig. 3. PN where actor is represented by a place *(his state is define by token nature in the place).*

In these examples, representation of interrupts is possible. But the application to an entire procedure would make understanding (by non-experts) somewhat difficult. We must bear in mind the fact that modelling will be presented to actors for validation

To simplify the global structure, the parameterised Petri net has been proposed by several authors. (Agimont, 96).

3.2.2.1. Definition

A parameterised Petri net is define as follow:

0) $RPP = \{C, D, P_P, T, I, O\}$

1) $C = \{CV_1, ..., CV_{|C|}\}$

2) $D = CV_1 \times CV_2 \times ... \times CV_{|C|}$

3) PP_i is a parameterised place, by definition a D subset

4) $D = CV_1 \times CV_2 \times ... \times CV_{|C|}$ is a parameterisation descriptor

with $\bigcup_{i=1}^{Pp} PP_i = D$ and $PP_i \cap PP_j = \varnothing$

5) pt_i is a vector transition, $pt_i : I(pt_i) \rightarrow O(pt_i)$, $I(pt_i)$ is the set of consummate places, and $O(pt_i)$, the set of product places.

6) $T = \{pt_1, ..., pt_{|T|}\}$ is the set of all actions that can be executed by the system.

7) $t_i = \{pt_{i1}, ..., pt_{i|t|}\} \subseteq T$ is a parameterised transition $\Leftrightarrow$

$\forall i, j, k : pt_i, pt_j \in t, PP_k \in P_P,$

$|I(pt_i) \cap PP_k| = |I(pt_j) \cap PP_k|$ and $|O(pt_i) \cap PP_k| = |O(pt_j) \cap PP_k|$

8) The set of all parameterised transition PT is defined by parameterisation descriptor P_p and by T, the set of vector transition.

9) A parameterisation is a correspondence between a PPN and a set of PPN which represent all the same system, so that only the P_p descriptor changes.

The next paragraph is an example of a PPN application.

3.2.2.2. Example

Instead of the traditional philosophers' diner example (which have been modelled with PPN in (Agimont, 96), let us take the following example:

Six computers have to scan and print (i.e. to photocopy) documents and classify scanned documents. But, there are only three scanners and three printers.

This problem can be classically modelled, but parameterised Petri net allows simplification of the net.

Let us come back to the definition:

$$RPP = \{C, D, P_P, T, I, O\}$$

$C = \{CV_1, CV_2\}$ with

$CV_1 = \{ comp_0, comp_1, comp_2, comp_3, comp_4, comp_5, scan_0, scan_1, scan_2, imp_0, imp_1, imp_2 \}$,

$CV_2 = \{ manage, copy, free, busy \}$

We have $D = CV_1 \times CV_2$

Let us define vector transitions (we shall see that there are several definitions of parameterised places).

There are two rules, for two actions, vector transitions are:

launch photocopy:

$I(pt_i) = \{ (comp_j, manage), (scan_{(j-1) \bmod 2}, \text{free}), (imp_{j \bmod 2}, free) \}$

$O(pt_i) = \{ (comp_j, copy), (scan_{(j-1) \bmod 2}, busy), (imp_{j \bmod 2}, busy) \}$

launch management:

$I(pt_i) = \{ (comp_j, copy), (scan_{(j-1) \bmod 2}, busy), (imp_{j \bmod 2}, busy) \}$

$O(pt_i) = \{ (comp_j, manage), (scan_{(j-1) \bmod 2}, free), (imp_{j \bmod 2}, free) \}$

for $i \in [0,11]$ (the 12 objects) et $j \in [0,6]$ (6 computers)

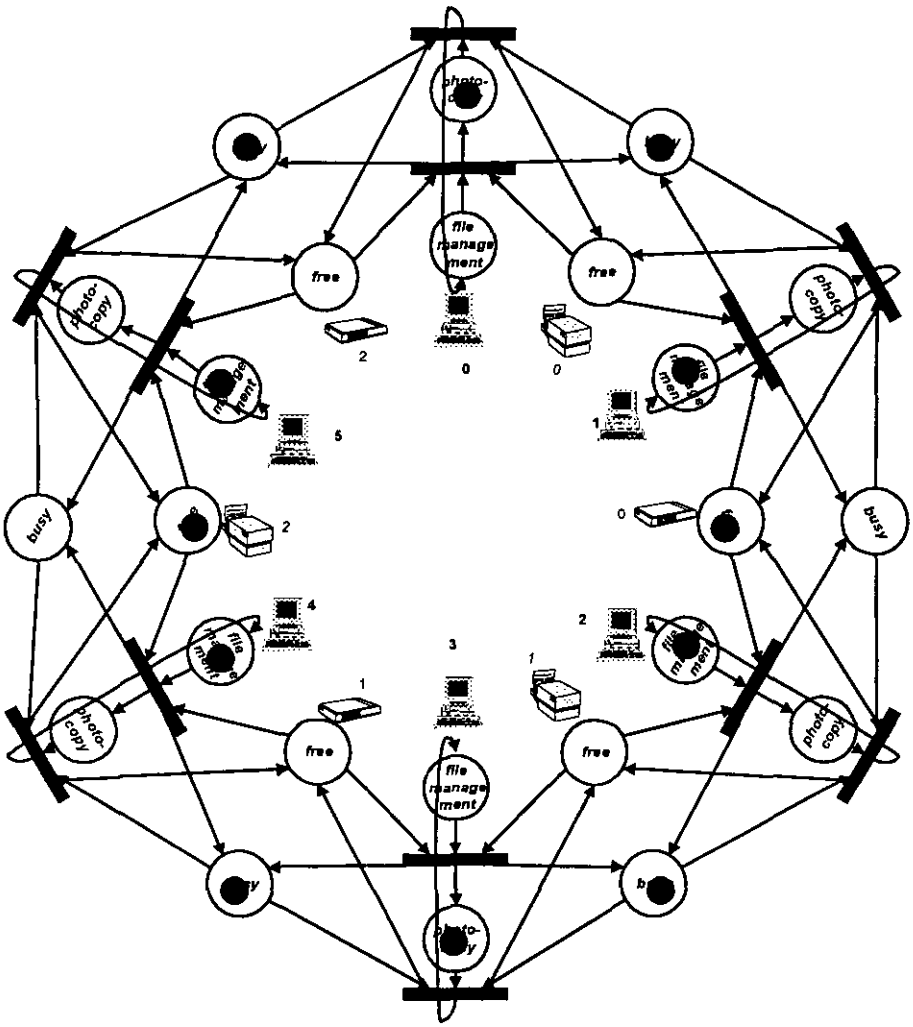
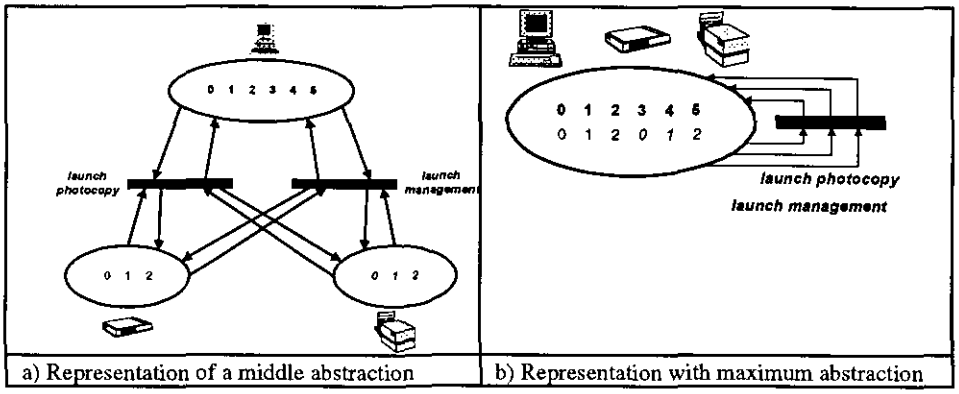


Fig. 4. Maximum detailed representation

We have $T = \{pt_1, ..., pt_{12}\}$

Parameterised places (Ppi), which are included in parameterisation descriptors (Pp), are not unique. It is possible to obtain the Petrinet (the fig. 3, or the fig. 4), with a higher abstraction level. The Petrinet illustrated by the Fig. 5 represents the highest abstraction level, all the system is defined by one place.



a) Representation of a middle abstraction | b) Representation with maximum abstraction

Fig. 5. Two different representations by PPN

So, parameterisation allows to simplification of a net, but the problem is limiting the abstraction, to have a net still readable. The Petri net on the fig. 5a is a good compromise between abstraction and detail. It allows, with rules, to follow system dynamic.

In our case, parameterisation to interrupt management has been applied.

3.2.3. Use of Parameterised Petri Net

Our net is based on the fig.2. This simplification leads to the building of two parameterized places building, one for activity representation, and the other for interrupt representation (fig.6). In actual fact, this two places system describes a workspace (an office).

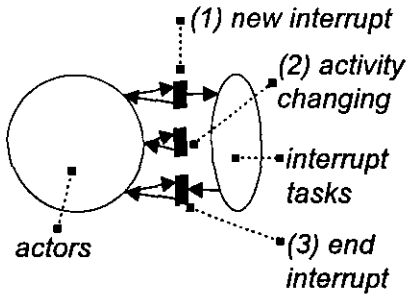


Fig. 6. Interrupt management by PPN

Activities are represented by states of actors which are present in the main place (this place may also contain used documents too).

Transition 1 is fired by an interruption. Actor state (the current activity, his progress in this activity) is stored in the second place.

Transition 3 is fired by the end of interrupt management. Actor returns to his task, or on the most urgent task.

Transition 2 is fired at the end of an activity. Actor gets a new state corresponding to a new activity (related to the documents in the main place).

Let us come back to the definition for this system:

$$RPP = \{C, D, P_P, T, I, O\}$$

$C = \{CV_1, CV_2\}$ with

$$CV_1 = \{w_1, w_2, ..., w_{|nb\_w|}, doc_1, doc_2, ..., doc_{|nb\_doc|}, int_1, int_2, ..., int_{|nb\_int|}\}$$

$$CV_2 = \begin{Bmatrix} activity_1, activity_2, ..., activity_{|nb\_activity|}, \\ waiting, processing, interrupted, completed \end{Bmatrix}$$

We have $D = CV_1 \times CV_2$

In this set of two places, there are three rules: to store state, to restore state, to change state. The vector transitions are defined by:

Store state

$$\begin{cases} I(pt_i) = \{(w_w, activity_a); (doc_d, processing); (doc_{d'}, waiting)\} \\ O(pt_i) = \{(w_w, activity_{a'}); (doc_d, interrupted); (doc_{d'}, processing); (int_{|int|+1}, activity_a)\} \end{cases}$$

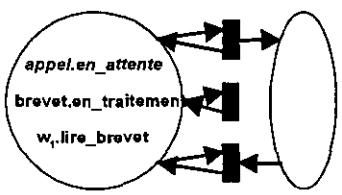
Restore state

$$\begin{cases} I(pt_i) = \{(w_w, activity_{a'}); (doc_d, interrupted); (doc_{d'}, processing); (int_i, activity_a)\} \\ O(pt_i) = \{(w_w, activity_a); (doc_d, processing)\} \end{cases}$$

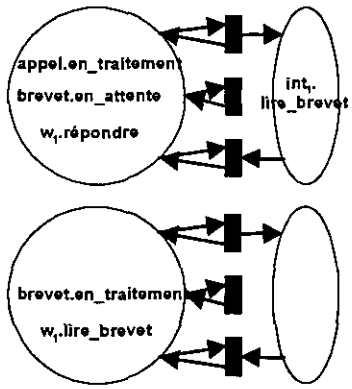
Change state

$$\begin{cases} I(pt_i) = \{(w_w, activity_a); (doc_d, completed)\} \\ O(pt_i) = \{(w_w, activity_{a'}); (doc_{d'}, processing)\} \end{cases}$$

Following example shows an interrupt management:



The actor is reading a patent, the telephone rings, transition 1 is fired, (store state rule is applied).



Activity is interrupted and the actor answers to the telephone

When the telephone conversation is closed, state is restored by the third transition

Fig. 7. Example of an interrupt management

Interrupting one activity for another is related to an actor. In fact, each parameterised system represents an office, decision and priority rules on the activities are different from system to system, i.e. linked to offices or playing roles.

The dynamic of Patent Department procedures has been modelled with these parameterised systems. An example of an application for a loan is shown in figure 1 (the dynamic model of an actual procedure requires two A4 pages). This modelling dynamic is only possible with very detailed activity and actor task analyses. Our dynamic modelling process is a progressive top-down approach, from the activity model and the processing model.

3.3. The simulating phase

Although Petri nets are less accessible to a non-expert, we think that a Petri net based on a simulation tool will allow the dynamic simulation of organisational working practices in a rather didactic way.

A simulator has been built (Gransac, 1997), and is actually under test. It has been built in two parts: one for the rule execution and the other for the interface.

The following structure is an abstract of the internal part:

Activity = {attributes (name, date, associated_roles, run time, interrupted), preconditions, actions }

Token = {attributes (name) , Activities, functions (run, stop)}

Place = {attributes (name, runable), Activities, $Tokens_{actors}$, $Tokens_{documents}$, $Tokens_{interrupts}$, functions (add_token, run, supp_token)}

Transition = {attributes (name, crossable, date), $Place_{Input}$, $Place_{Output}$, Activities }

Net = { attributes (name, nb_actors, nb_doc), Transitions, functions (add_transtion, add_token, ..)}

Each part of the net has its own activities, i.e. its own preconditions and actions. As soon as an event appears in a place (document arrival or actor coming), tokens present

in the place check their preconditions. If an actor interrupts the current job for another one, an interrupt event is raised and the place creates an interrupt.

Transitions have, of course, their own activities too. In our case, transition actions generally consist of documents passing and actors moving. As these actions take time so, our transitions are temporised.

The two parameterised place system is programmed in one place. Activities (preconditions and actions) directly follow from the Petri net used in the dynamic model. The two-place systems appear only at the screen.

The second part of the simulator is the interface. It has been defined classically or nearly so (PPN = {Tokens, Two-place System, Transitions, Arc}). Each object drawn on the screen involves the creation of its twin in the internal part.

The precondition rules are keyed in a window, which show rules in a form close to natural language. (fig 8)

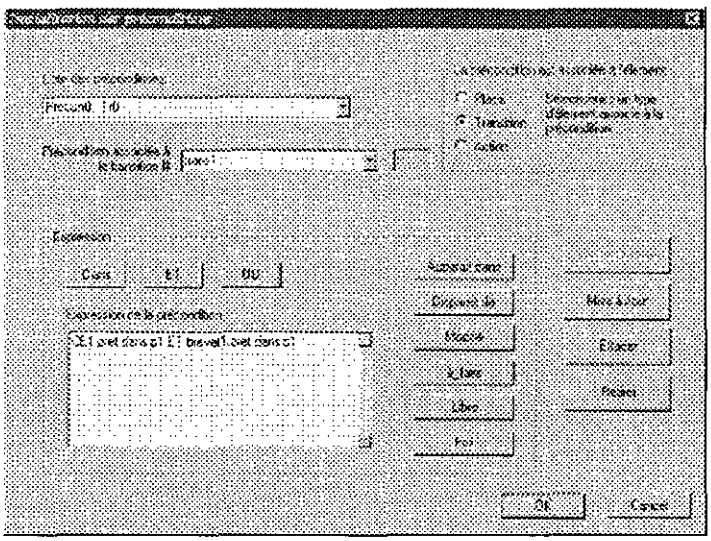


Fig. 8. Keyboarding of preconditions

The analysis part of the net deals only with the net physical aspect (test if all places and transitions are linked). The more thorough aspects, the fact that all places could be reached or that the net does not loop have not yet been programmed. The simulator is based on a modelling of the reality, so it is the people who have made the dynamic model who perform this net validation task.

The simulator has a learning goal, some new work organisation (with or without new tools) can be checked in a didactic way, with loops and jamming appearing clearly.

This simulator has been built in order to be used by people who are not conversant with modelling or Petri net. Simulator initialisation does not take too much time, and first tests allow us to identify 7 elementary activities on documents.

4. First results

As already mentioned, our study case was conducted in a Patent Department involving about thirty people. This department wanted clearly to set up a computerised solution for helping document handling, whilst rendering co-operative work easier.

Today, the complete analysis of the Patent Department processes and activities has been achieved ((Carrere, 1996), (Notte, 1996)).

The highest priority procedure has been built as part of a BPR process. The results are twofold:

- Organisational level: for each procedure, concerned persons have been confronted with static modelling of data, activity and processing. Some organisational solutions have been proposed and have been kept. The simulation allows the anticipation of future jamming and higher workload.
- Computer level: data, data flow, data processing modelling has allowed us to specify a CSCW solution helping actor systems to communicate data. PPN initialisation and its firsts usage aims at specifying in a more detailed way this solution. Rules definitions of parameterised systems, linked to roles, are a first attempt to specify a helping system for each role.

To make this modelling exercise easier, a CAD Software Workshop has been established. (Fig 9)

It proposes a set of four pages (one for each model) and a toolbox including the list of symbols using during the modelling exercise. (Fig 9). It is based on the Visio 3.0 Software.

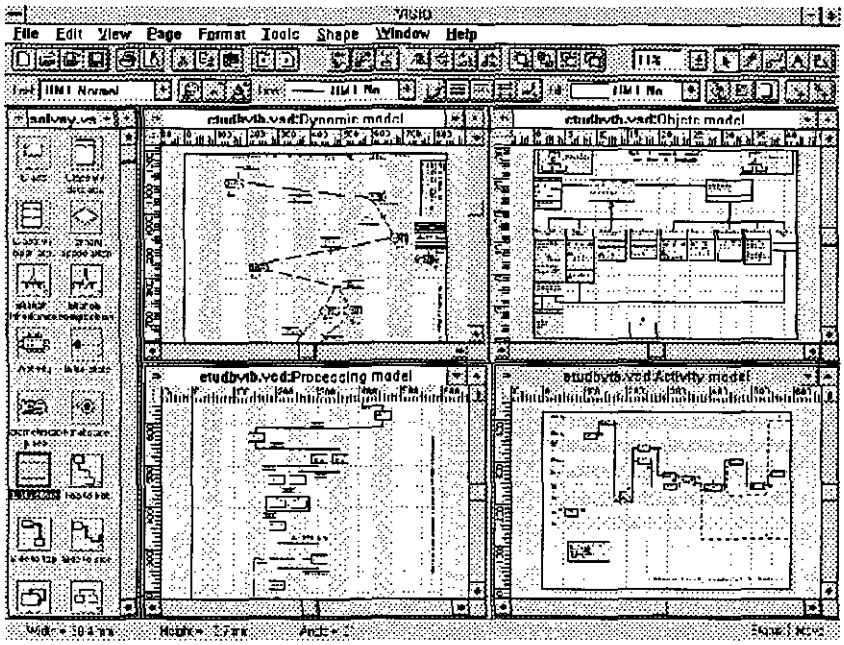


Fig. 9. Screen copy of the workshop

5. Conclusions and perspectives

The proposed model based approach makes the integration of computerised solutions in a complex human organisation easier. It is necessary for all people involved in the exercise to clearly understand their role and their position. The success of co-operative working practice is entirely dependent upon this process.

For the time being, our approach was restricted to organisation modelling. We are now planning to extend into the setting up of Intranet and Internet solutions.

As for a better and more systematic use of models, it is important to construct an easy-to-use computerised system, we believe that emerging CAD software will be of great help in achieving these goals.

Our tool based on the Petri net is currently in a test phase, which will allow us not only to simulate the way organisations work but also to check the effectiveness of the proposed paradigm

Finally, we are planning to «export» our methodology into Research Units.

Acknowledgements

The authors thank Joe Galway for considerable assistance in the translation of this paper, and also Prof. Christophe Kolski for his remarks concerning the first versions of the paper.

Special thanks to Jerome Gransac whose we owe the simulating phase.

References

Abed M. & Angué J.-C. (1994), *A new Method for Conception, Realisation and Evaluation of Man-Machine Interfaces*, Proceedings IEEE , Systems and Cybernetics, San Antonio 2-5/10

Adam E., Vergison E., Kolski C., Mandiau R, (1997), *Holonic User Driven Methodologies and Tools for Simulating Human Organizations*, ESS'97, European Simulation Symposium, University of Passau, Passau, Germany, October 19-23.

Agimont G., Le Strugeon E., Mandiau R., Libert G. (1996), *Parametrized petri nets for organizational simulation and systems design.* Proceedings of the Second International Conference on the Design of Cooperative Systems (COOP'96), COOP Group (Ed.), Juan-les-Pins, France, 12-14 June.

Bowers J.M., Benford S.D. (Eds.), (1991), *Studies in CSCW, Theory, Practice and Design*, Ed.,Nottingham.

Carrere P., (1996), *Production, gestion et diffusion électronique de documents, Ré-ingénierie des processus d'activités d'un département de Propriété Industrielle*, DEA Technical Report - University of Compiègne & SOLVAY S.A.

Connoly J.H. and Edmonds E.A. (Eds.), (1994), *CSCW and Artificial Intelligence*, Springer-Verlag, London.

Diaper D. (1989). *Task analysis for human-computer interaction.* Ellis Horwood Limited, Chichester, United Kingdom.

Dumas P., Charbonnel G. (1990), *La méthode OSSAD, pour maîtriser les technologies de l'information.* Tome 1 : principes. Les éditions d'organisation, Paris.

Ermine J-L. ,(1995) , *MKSM, méhtode de gestion des connaissances*, CEA DIST/SMTI.

Gracanion D., Srinivasan P., Valavanis K.P., (1994), *Parameterized Petri nets and their application to planning and coordination in intelligent systems.* IEEE Transactions on Systems, Man and Cybernetics, 24:1483-1497.

Gransac Jerome, (1997), *Construction d'un simulateur de Réseau de Petri Orienté Objet - Contribution à l'étude de systèmes administratifs complexes*, Mémoire de DESS - Université de Valenciennes & SOLVAY S.A..

I.G.L. Technology (1989). *SADT, un langage pour communiquer.* Eyrolles, Paris.

Jambon Francis, (1996), *Erreurs et interruptions du point de vue de l'ingénierie de l'interaction homme-machine*, Ph. D. dissertation, Grenoble University, France, December 1996

Koestler A. (1969), *The Ghost in the Machine*, Arkana Books, London.

Kolski C. (1997), *Interfaces homme-Machine, application aux systèmes industriels complexes (2ème édition revue et étendue)*. Editions HERMES, Paris.

Notte D., (1996), *The Patent Department : Domain study*, (in French), SOLVAY internal report, Ergodin.

Palanque P., Bastide R., (1995), *Spécifications formelles pour l'ingénierie des interfaces homme-machine*. Technique et Science Informatiques, avril, 1995.

Palanque Philippe, (1992), *Modélisation par objets cooperatifs interactifs d'interfaces homme machines dirigees par l'utilisateur*. Thèse de l'Universite Toulouse 1, Septembre 1992.

Reason James, (1990). *Human Error*. Cambridge : Cambridge University Press, 1990.

Rumbaugh J., Blaha, Premerlani W., Eddy F., Lorensen W. (1991).*Object-oriented modeling and design*. Prentice-Hall.

Shael Thomas, (1997), *Théorie et Pratique du Workflow, des processus métiers renouvelés*, SPRINGER

Tardieu H., Rochfeld O., Colleti R. *La méthode Merise, principes et outils, 2ème édition*. Editions d'Organisation (tome 1), Paris, (1991).

Scrivener Stephen A.R. (Ed.), (1994), *CSCW : The multimedia and networking paradigm*, UNICOM

Wilson J.R., Corlett E.N. (Eds.) (1990). *Evaluation of human works :a practical ergonomics methodology*. Taylor and Francis.

Combining abstraction and context: a challenge in formal approaches to workflow management

S. Donatelli, C. Simone, and D. Trentin

Dipartimento di Informatica,
Università di Torino
simone@di.unito.it

Abstract. Flexibility in workflow management can be obtained through an unanticipated interplay between the identification of recurrent patterns of behaviour and a dynamic use of information about the context where the workflow actors operate. (awareness).
The design of flexible workflow management systems can take advantage of results in process modelling, that combine abstraction and context in all phases of its life cycle. A specifical approach, proposed in [5], is illustrated and interpreted in the light of the above claims. The approach is taken as a starting point for a research agenda aimed at enlarging the set of problems that a formal approach to workflow management system can deal with.

1 Introduction and motivations

The term workflow usually refers to the representation of the part of coordination which is based on recurrent patterns of behavior based on work practices and organization rules. According to the terminology proposed by [6] we distinguish three basic phases in workflow management: definition, enactment and execution.

The theme of Workflow Management (WFM) is common to various domains that take quite different perspectives on it. First of all, the *domain of production* of commercial systems: vendors offer more than 200 WFM systems and have dedicated a remarkable effort in searching for a standardization [6] to guarantee interoperability among themselves as well as with the legacy system environments where they are inserted. Secondly, the domain of *research on process modelling*: here Petri net theory played a relevant role since very beginning and produced results overcoming one of the basic limits of the commercial systems, namely a very limited support to validation when business processes are defined and modified, both during their conception and enactment. Third, the domain of *Computer Supported Cooperative Work* (CSCW) where the development of (prototype) systems supporting cooperation, and among them WFM systems, is enriched with conceptualizations of what cooperative work is. These conceptualizations are often based on field studies aimed at capturing the requirements of cooperative systems (see, e.g., [3] for the case of WFM systems) by analyzing work practices in settings where both a supporting technology is adopted or not.

The three domains developed, indeed, with a very loose interaction. For example, within CSCW, one of the first attempts to build a system based on net models [13], showed very soon its weakness in the way the modeled reality was considered [12]. Among others, the considerations relevant here are the tension between an understandable language and its expressive power, and the fact that each procedure instance is treated separately.

On the other hand, in the framework of Petri net theory, the approach to WFM systems was mainly focused on the definition of tools to support the design of the business process at hand based on various forms of refinement or compositional operators [15, 19]. These techniques provide process designers with useful tools that have, on the other side, the drawback of not supporting adaptive design: that is to say, design as a continuous action which is situated in a *context* and that can last for the all life cycle of the considered business process.

More recently, both commercial products and theoretical framework proposals have devoted much attention to incremental design in order to take into consideration the contingent aspects of process execution. Again, commercial products lack support to validation. The problem of modifying the structure of running instances was the focus of some theoretical work [1, 11]: however, the proposed solutions are only applicable under some constraints on the net structure (e.g., input/output places, no cycles, etc.) which make their application sometimes problematic in real contexts, at least at this present stage.

Adaptation is not the only open issue: recent experiences in empirical studies and in CSCW system design have increasingly emphasized the role of *awareness* in cooperative work and of its impacts on system design (and therefore also on WFM systems design). Most of this paper is indeed devoted to this problem.

Awareness has become a keyword denoting how the knowledge of the context of cooperative actions plays a relevant role in actors coordination. Awareness can take different forms and contents: however, it can generally be defined as the *mutual perception that actors maintain about their different views on the common working space.* These multiple views are necessarily present in cooperative work due to its inherent distributed nature [17], and concern the structure and behavior of the cooperative business processes in which the cooperating actors are involved. Formal approaches to business process design privilege instead, in an attempt to master the complexity of the problem, the view of a process (or any sub-part of it) as uprooted from its context, both when it is defined and when it is enacted and activated. Refinement techniques are a representative example of the approach: all rules we know of can be applied under the basic hypothesis that the module to be substituted for an event is fully disjoint from the net system to be refined. Since the goal is the highest degree of context-independence, these techniques impose severe requirements in order to guarantee a good behavior of the substitution in any context. The limit of this approach was already put in evidence in [7] where it is observed that resources can hardly be seen as local to an action since they are normally shared by definition and they are used across the actions and their refinements. Also the approach proposed in [2], which gives a

small response to the last problem, basically suffers from a limited applicability due to the many constraints imposed on shared objects.

A similar argumentation holds for the run-time modification of process instances. Its correctness and consistency is sought for, rightly, in relation to the process functionality but the impacts of this modification on the interacting cooperative processes are not considered. Apparently, what seems to be relevant here is a sort of input/output behavior that has been so aptly criticized by R. Milner [14] in the early 80's when concurrent systems are concerned.

In summary, the current approaches to business process design and enactment consider the process as a "global" entity that contains all the necessary information, possibly represented at different levels of abstraction. In our view, this approach induces a bias in the above techniques which, though useful within their range of application, are limited in dealing with the distributed nature of cooperative work. On the other hand, the traditional notion of abstraction "as a black box hiding the implementation details" has been questioned [9] in relation to the design of user interfaces which guarantee system extendibility and modularity and at the same time a meaningful information for the user when the system breaks down. While the black-box approach supports system manageability, software reuse and a modular and reliable software maintenance, it lacks to provide the connection between the interface and the inner behavior of the system when the latter is needed by the user to interpret or react to a complex or unexpected behavior. In Dourish's words [9]: "It does not imply to provide a set of *hooks* directly into the implementation ... Instead, ... a rationalized model of the inherent behavior of a system offering its particular functionality".

It is not difficult to transpose this argument from user interface to more general interfaces between cooperative business processes. Basically, this is another way to stress the relevance of that type of information that we formerly denoted as awareness about processes. Now, the point is to decide if and how the management of this type of awareness is hand in gloves with the design of the business process itself, or if it has to remain off-line, as part of some unconnected design or user's activity. In our opinion they have to be strongly connected if awareness information has to be aligned with the current definition and behavior of the system it refers to. In addition, and unlike Dourish's conclusions, we believe that the use of formal design techniques can be helpful to avoid leaving all awareness management up to the users.

The context of a process is the environment in which it is defined and activated: an abstraction of a set of processes both interacting with the considered one and evolving in an autonomous way is part of the context. Every modification in the context or in the process has to do with the constraints defined by the other. The already mentioned classical approach to the abstraction, postulating or guaranteeing modifiability in any context, is not an ultimate answer, since it is too demanding; on the contrary, the possibility to take into account specific context properties allows for a greater flexibility in the description and modification of the process at hand.

Moreover, in the case of WFMS, where awareness management is fundamen-

tal, this last approach seems more appropriate, since context management can serve different purposes at the same time: supporting process design and the mutual awareness of the involved actors executing it. Then, if one is able to adequately combine abstraction and context in the framework of Petri net theory, it is possible to enlarge the use of Petri nets in the design of WFM systems, if not in improving existing techniques to solve well known problems, at least in enlarging the types of problems that WFM systems can contribute to deal with.

The paper is organized as follows: Section 2 discusses the role of awareness, Section 3 illustrates an approach proposed in [5] which goes in the direction of the above requirements as it combines context management with abstraction techniques. The approach originated from a completely different framework, namely performance evaluation and specifically the management of state explosion. Although quantitative aspects are not considered in this paper, the proposed approach leaves open the possibility to consider performance evaluation also in the case of WFM systems. Section 4 discusses how the approach of Section 3 can be used for WFM to take into consideration awareness. Section 5 illustrates the basic ideas through a simple example. This proposal is taken as a witness of a possible research approach and not as a fully satisfactory solution. Section 6 concludes the paper by illustrating a research agenda.

2 Interplay between workflow and awareness in coordination

The experience derived from the studies on how people achieve coordination in real work setting shows that coordination is based on a continuous interplay of

(a) recurrent patterns of behavior associated to work practices and organization rules: those patterns have been aptly called *precomputation* [16],
(b) ad-hoc forms of behavior to adapt to the needs and constraints of a contingent situation.

The notion of workflow is typically associated to behavior of type (a), but cannot be separated from the second one if one wants to achieve the flexibility required by the real work settings.

Flexibility can be achieved through the partial definition of workflows to be completed, possibly during their execution, through incremental design and run-time modifications. Even in the most standardized work setting, recurrent patterns of behavior cannot be fully specified and "the vagueness of plans" is "ideally suited to the fact that the detail of intention and action must be contingent on circumstantial and interactional particulars of actual situations" [18]. These circumstantial and interactional particulars are exactly what awareness is about. In fact, the completion and adaptation of a workflow can be performed in a more effective way if the involved actors are aware of the state of affairs of the common field of work that is constituted, at least, by the resources and the cooperating workflows that make possible the successful execution of the workflow at hand. The consequence is that a technological support to coordination

should take into account both workflow and awareness management in order to be suited to handle any possible mix of recurrent and ad-hoc behavior and their inherently dynamic specification. This idea is sketched in Figure 1, where the point of view of a "distinct workflow" is taken: its environment is made up of resources and its cooperating workflow. Typically, awareness information can be

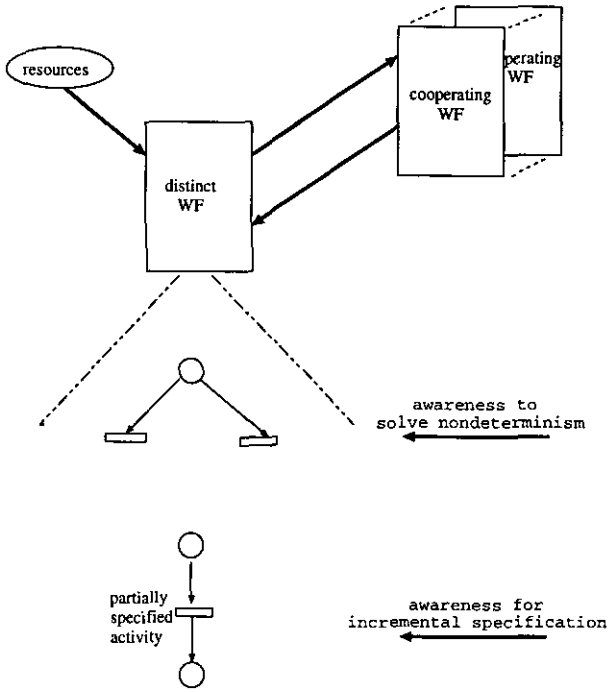


Fig. 1. The use of awareness information.

used by the actors involved in the execution of the workflow to select the more appropriate action, e.g. in solving the non-determinisms inherent in conflicts or in choosing among alternative ways to complete the workflow specification.

Awareness information can be used also to evaluate the impacts of choices local to a workflow in relation to the properties of the workflows cooperating with it. This means that awareness information concerns both the structure and the behavior of the entities constituting the context of the workflow at hand. These aspects will be discussed in the next sections in terms of a formal approach combining abstraction and contexts in workflow management.

3 Combining abstraction and context

The modeling methodology presented by Buchholz in [5] organizes models into a hierarchy. Leaf nodes are called atomic processes and non leaf ones are called

coupling processes. The innovative feature of the approach is that each node is made up of two parts: a description of a local behavior and a description of the environment.

The basic modeling formalism used to specify processes (coupling or atomic ones) is a class of labelled stochastic automata called PMTS (Parametrised Multi-labeled Transition System), that can take into account also time, in particular time as delays associated to activities, but we consider here only its untimed counterpart. For PMTS an operation of Synchronized Product is defined, that, despite its name, allows both synchronous and asynchronous communication among PMTS; we indicate it with the symbol $\otimes$. Labels are used to distinguish state transitions that are local to a PMTS (τ label), transitions that require a synchronous change of state in two or more automata, and transitions that simply send or receive a message in an asynchronous manner.

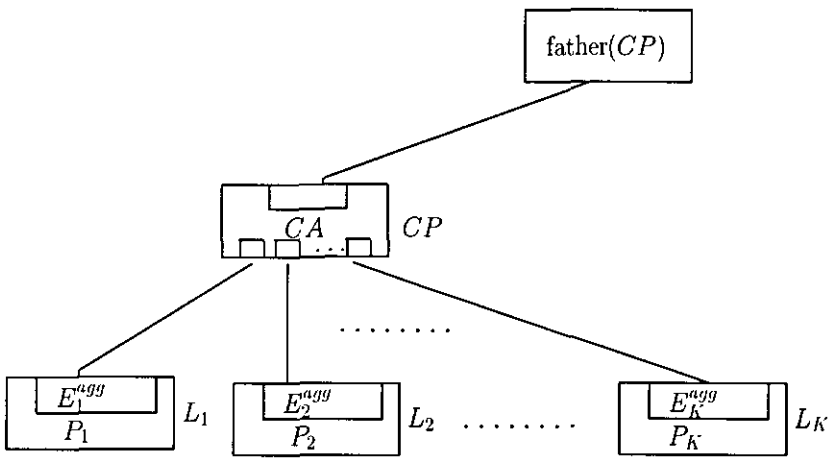


Fig. 2. Hierarchy definition.

Figure 2 shows a portion of a three level hierarchy. Leaf nodes $L_1, \ldots L_K$ are the atomic processes. Each atomic process is the Synchronized Product of two PMTS P_i and E_i^{agg}, the first one representing the detailed model of a portion of the modelled system, and the second one representing an *abstract view* of the rest of the system (called environment).

Abstraction is defined through state space aggregation (each state of the aggregated view represents a set of equivalent states of the detailed model).

A non leaf node CP with K children contains $K + 2$ PMTS: an aggregated representation of each child L_i and of the father of CP, and the local behavior of the coupling activity (denoted by CA in the figure), if any is necessary, describing the interaction among the $K + 1$ children.

Each node defines therefore, to a certain abstraction level, the full model, but the details of the model are contained only in the leaves. For a node to be consid-

ered a good representation of the global model, each aggregated PMTS should be consistent with the corresponding detailed PMTS. The proposal in [5] allows different levels of consistency which are based on different relations among detailed and aggregated PMTSs obtained by means of different types of partitions of their state space.

An aggregated view that preserves the interface behavior is said to be *completely consistent*: this notion is a form of weak bisimulation and preserves, e.g., deadlocks. The state space partition is then obtained by the classical inductive algorithm. Complete consistency guarantees behavioral properties, but may be too strict, as the aggregated view may have the same order of states as the detailed model.

To overcome this problem, a notion of *weak consistency* has been introduced: a weakly consistent aggregated view is a pair of PMTS, one describing an *almost consistent* view, and the other describing an *at least consistent* view of detailed process. The almost consistent view can be constructed by partitioning the state space according to any equivalence relation, with the only constraint that it is a refinement of the equivalence induced by the states of the aggregated view of the environment: state transitions among equivalence classes is such that the communication language of the detailed view is a subset of the communication language of the almost consistent view. The at least consistent view can be constructed on top of the almost consistent state space aggregation by eliminating the state transitions of the aggregated view according to the following idea[1].

Let s and s' be states of the detailed view, z and z' be states of the aggregated view, and α a communication label: then the state transition $z \xrightarrow{\alpha} z'$ is eliminated if states s and s', belonging to the equivalence classes represented respectively by z and z', exist such that $\neg\ (s \xrightarrow{\alpha} s')$. In this case the communication language of the detailed view is a superset of the communication language of the at least consistent view. To sum up, the three notions of consistency define the following relations among the languages $\mathcal{L}$ of the different views:

$$\mathcal{L}_{at-least-consistent} \subseteq \mathcal{L}_{complete-consistent} = \mathcal{L}_{detailed-view} \subseteq \mathcal{L}_{almost-consistent}$$

In [5] some operations on the static hierarchy are discussed. First of all, the refinement operation can be applied just to leaf nodes, which contain the detailed views. In order to keep the desired consistency, either complete or weak, the original and refined leaf should be indistinguishable from the environment, i.e., both can be represented by the aggregated view which is completely (weakly) consistent for both. The contrary operation is aggregation, of which the above discussed aggregated view is an example preserving behavioral properties. These operations modify the state space and the state transitions in the affected nodes. Other operations, called abstraction and reduction, modify the structure of static hierarchy leaving unchanged the underlying state space: they are not considered here because they seem to be of little use in the framework of workflow modeling.

[1] It is behind the scope of this paper to give all the technical details, specifically about the problems arising from the presence of synchronous and asynchronous communication.

4 Applying the approach to WFM

This section gives an intuition of how the above approach could be used for the modeling of cooperative workflows. To this aim we reconsider the portion of hierarchy illustrated in Figure 2 in the light of the arguments discussed in Section 2, that is shown in Figure 3.

Let P and Q be two cooperative workflows of alphabet $\alpha(P)$ and $\alpha(Q)$, that is, workflows that interact by means of a non empty communication alphabet $A(P,Q) = \alpha(P) \cap \alpha(Q)$. In the atomic processes incorporating them they are combined with the aggregated view of the environment in which they operate. The flexible definition of aggregation criteria, in combination with the hierarchical description, allows the workflow designer to organize the information in a way that can help the management of the workflow description complexity.

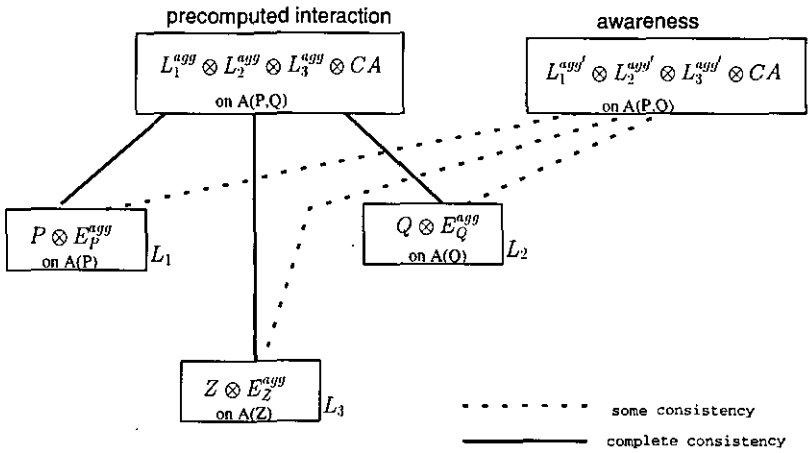


Fig. 3. Hierarchy and workflow.

Let $A(P)$ ($A(Q)$) be the actions of P (Q) not belonging to $A(P,Q)$. The portion of the aggregated environment that refers to $A(P)$ ($A(Q)$) is called the "local environment" of P (Q), denoted LE_P (LE_Q). Actions in $A(P,Q)$ are considered as "potentially possible" both in P and Q, as the constraints governing them are specified in the coupling process. Before moving to coupling, let us consider the possibility of an additional leaf node representing a process Z that does not interact directly with P and Q, but nevertheless belongs to their environment. In the WFM framework this process can be either a totally unconneted one or, more interestingly, a process that has to monitor the behavior of P and Q for some unspecified reasons (e.g., for sake of auditing or for sake of awareness).

The construction of the aggregated is what is left to be specified: the consistency criteria can support an articulated methodological approach in relation to

the already discussed interplay of workflow and awareness in coordination, and to the meaning and use of the contextual information in the two cases. Since both workflow and awareness mechanisms have to be explicitly designed, their specification has to appear in the coupling process node: however, they can be governed by different consistency criteria.

As far as workflows are concerned, the context plays the role of constraint of refinements/modifications local to the workflow or a role of trigger of propagation of changes from a workflow to the workflows cooperating with it. This happens when the modification violating the constraints represented by the context cannot be avoided and the latter has to be realigned together with all processes generating it. In this case, complete consistency has then to be considered the appropriate criteria: in fact, modifications/refinements are the typical ways to reduce the complexity of workflow definition where strong behavioral properties have to be taken into account. Of course, this is not a complete solution of the problem: that is, providing refinement or modification techniques preserving good behaviour in relation to a given context. What is provided is a support to the workflow design in terms of allocation of functionality between context and process and of "a posteriori" check of the correctness of the realized refinement/modification. Moreover, a reduced representation of the context can make it easier for the designer to identify the extent to what a modification affects the cooperating workflows, in a sort of domino effect, and to define accordingly the appropriate propagation strategy. One has to notice that also in the execution phase modifications local to an instantiated process (typically, the enforcement of a transition to a new state) can affect the cooperating workflows. Finally, if a WFM system is able to keep trace not just of the context structure but also of its behavior, then the above reasoning can govern the application of the techniques mentioned in Section 1 for run-time modifications.

On the other hand, weaker notions of consistency can be used for dealing with awareness mechanisms: in this case, again according to the arguments proposed in [9], the required information can be less precise, in favour of a smaller representation that users can more easily understand and take advantage of. Here the problem is to decide what information can be considered as sufficient to promote the suitable awareness for each specific recipient. This choice is in charge of the users/designers: however, notions like almost consistency and at least consistency give examples of different techniques to provide it. This point will be further illustrated in the next section, where an example is discussed.

Coming back to the construction of the coupling process, we can say that in the framework of WFM the coupling node can be viewed as consisting of two parts, represented as separated coupling nodes in Figure 3.

The first part relates to the handling of the "precomputed" interaction of cooperating workflows, the second to the interaction in terms of awareness. In the first part, the coupling processes node contains the completely consistent aggregated views of the atomic process nodes related to P and Q and their local contexts. In addition it contains the representation of the local coupling activity defining the constraints on the interaction between P and Q in terms of

the actions belonging to the $A(P,Q)$. In this way, the coupling processes node contains all the information concerning "precomputed" interaction among the cooperating workflows.

The second part contains the aggregated views of the atomic process nodes related to P and Q and their local contexts according to the selected consistency criteria for sake of awareness management. In this case, the local coupling activity can be omitted by interpreting it as the most flexible way to access the information contained in the related aggregated views: this local activity does not need to be explicitly designed as it can be automatically provided by the engine supporting workflow execution, a standard component of any WFM system.

It is worthwhile to notice that the presence of a massive use of aggregation does not lead to any overhead to the users/designers. In fact, the construction of completely consistent aggregations can be automated when the state aggregation is based on a notion of equivalence implying complete consistency and for which an aggregation algorithm exists (e.g., in the case of the largest weak bisimulation, as proposed in [5]). This is a valuable support in refinement and modification: in fact, once the refinement/modification is performed on the leaf nodes, the related aggregation can be dynamically computed. For sake of validation, a check of isomorphism between the aggregated views of the source and refined processes (according to the same aggregation criteria) can then be performed, on a possibly radically reduced representation. The same holds when aggregations are based on the weaker notion of state space partition: also in this case the check of at least consistency can be automated.

5 A simple example

In order to show how the above concepts can be applied to the design and enactment of a workflow, we discuss a simple example which is adapted from [5]. Consider a business process which serves two types of requests coming from the outside, distinguished by the (high or low) amount of resources they require and indicated by Rh and Rl, respectively. Only a single request belonging to Rh is allowed to be in the system, at any time, while a finite number (let us denote it by max) of requests of the second type can be simultaneously present and satisfaction of low priority requests is not checked by the outside. Service can be suspended at any time for some internal reasons while its resumption depends on external conditions: for example, the suspension might concern some asynchronous operation (like an inventory or an auditing process) or more traditional exceptional situations requiring external interventions.

Now, one could decide to define the service process by incorporating in it all the above requirements. Let's suppose that we can trust the environment of being able to control the flow of requests belonging to Rh and in so doing, it is able to guarantee that no Rh arrives while one is already in. By using the approach presented in the previous section we give an explicit representation of

both the context (environment) and the process in such a way that the latter can deal just with the requirements under its responsibility.

For sake of presentation we describe the state space of the business process by the labelled P/T system generating it, shown in Figure 4 (this is not to say that we claim that the proposed approach operates at the system level, as it will be discussed in the concluding section). Label h_{in} (l_{in}) indicates the arrival of a Rh (Rl) request; h_{out} indicates the termination of a Rh request (observe that there is no corresponding l_{out} since there is no feedback for Rl requests); as/ex indicates instead the event that causes a diversion from the regular activity. Internal actions are, as usual, represented by τ (tau).

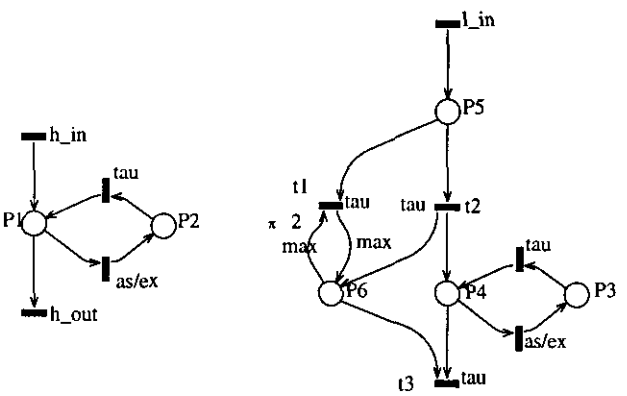


Fig. 4. Nets for the generation of the state space.

Transition $t1$ has priority over $t2$, and this allows to eliminate all requests that exceed max. Transition $t3$ represents the end of a Rl request. Observe that the left portion of the net in Figure 4 represents an unbounded system, while the net on the right has all places bounded by max.

To complete the atomic process we need to define the context, as seen from the processes. The context is of course not complete: rather, it mentions just what is needed for the process at hand to behave according to the above requirements. The aggregated state space can be described by the labelled state machine of Figure 5. States Y and N denote the situations in which a high priority request is present or not, respectively.

In the corresponding state space, each state is described as a triple: $\langle z, n_l, n_h \rangle$, where z is the aggregated context state (Y or N), n_l is the number of tokens in $P6$, and n_h is the sum of tokens in $P1$ and $P2$. Its structure is quite obvious from the above labelled net system: we want to recall only that in all states where the first component equals max, any transition labelled by l_{in} will leave these states unmodified. This is modeled in Figure 6 (only the pertinent part of

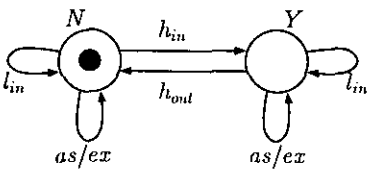


Fig. 5. The aggregated context of the process.

the state space is shown) where x denotes any non-negative number.

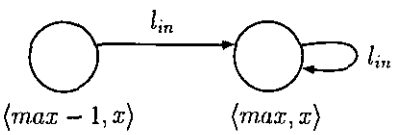


Fig. 6. State space border conditions.

Once the workflow is defined, it has to be enacted in order to be executed. One basic part of the enactment is the assignment of resources to the process together with their possible constraints. Let us assume that two actors are assigned to it in order to answer the various requests. Moreover, as part of the above mentioned interference with the request handling, the resources can become temporarily unavailable, e.g., employees can be sick or assigned to other duties. The enactment of the process can then be performed by enriching the workflow description in the above atomic process with an additional component modeling the resources, again described as a net system in Figure 7.

Each state of the enriched atomic process contains an additional component to represent the number n_{act} of available actors: $\langle z, n_{act}, n_l, n_h\rangle$. The resulting state space, for $max = 3$, contains 24 states, listed in Table 1.

As illustrated at the end of the previous section, the atomic process has to become part of the context of other atomic processes cooperating with it (through the coupling process level). This problem can be dealt with both from the design and enactment perspective. Since the resulting argumentation is quite similar, we consider the second perspective.

How to construct the aggregated view of the above 24 states? As anticipated, the selection of the aggregated view depends on its use within the coupling process node. For what concerns the part of the node related to the precomputed interaction, the choice is towards completely consistent aggregations. The only partition that does not lead to a trivial aggregation (i.e., that makes a true reduction of the state space) and that is completely consistent, is the one based on the state of the environment and number of actors. This aggregation, shown in Figure 8, fully maintains the interface behavior of the detailed view.

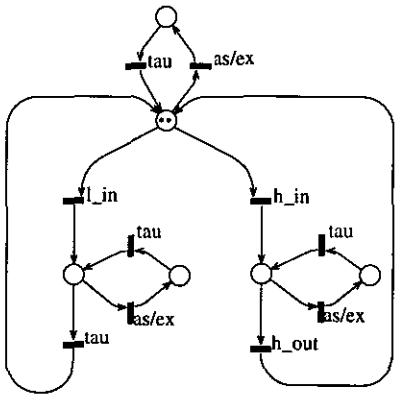


Fig. 7. Adding resources.

1	$\langle N,0,0,0\rangle$	13	$\langle Y,0,0,1\rangle$
2	$\langle N,1,0,0\rangle$	14	$\langle Y,1,0,1\rangle$
3	$\langle N,2,0,0\rangle$	15	$\langle Y,2,0,1\rangle$
4	$\langle N,0,1,0\rangle$	16	$\langle Y,0,1,1\rangle$
5	$\langle N,1,1,0\rangle$	17	$\langle Y,1,1,1\rangle$
6	$\langle N,2,1,0\rangle$	18	$\langle Y,2,1,1\rangle$
7	$\langle N,0,2,0\rangle$	19	$\langle Y,0,2,1\rangle$
8	$\langle N,1,2,0\rangle$	20	$\langle Y,1,2,1\rangle$
9	$\langle N,2,2,0\rangle$	21	$\langle Y,2,2,1\rangle$
10	$\langle N,0,3,0\rangle$	22	$\langle Y,0,3,1\rangle$
11	$\langle N,1,3,0\rangle$	23	$\langle Y,1,3,1\rangle$
12	$\langle N,2,3,0\rangle$	24	$\langle Y,2,3,1\rangle$

Table 1. State space of the leaf node.

As far as the management of awareness is concerned, there are various possibilities, that, on the basis of different semantics represented by the selected partition, and guarantee different levels of consistency. A first choice is to aggregate the states according the states of the environment (that is, the resulting aggregated state space has just two states corresponding to the value Y and N): in this case, the aggregation is obviously almost consistent but not completely consistent with the atomic process since a high priority request can be satisfied (h_{out} can fire) also when no actor is available. This behavior is not possible in the detailed view. More generically, the choice to focus on the local environment of a workflow disregards the information about any type of constraint contained in the workflow description. Consequently, this choice can be considered, again in general, not satisfactory from the point of view of the workflows cooperating

with the considered one but it can be acceptable from the point of view of other parts of the organization which can have different purposes, e.g., to have awareness information just on the presence of high level requests for starting some monitoring activity concerning them. Now, there are different ways to take into account information about the specific behavior of the workflow. For example, one may want to get information about all the current requests, irrespective of their priority: in this case, the aggregation can be based on the state of the environment to capture high priority requests and the number of low priority requests. This choice reduces the state space (to 8 states) and leads to an almost consistent aggregation telling about the dynamics governing the number of the requests of the two types.

The not completely consistent aggregation based on the existing requests can be transformed in an at least consistent aggregation by deleting the state transitions labelled by h_{out}. In fact, in this case, the interface behavior is maintained on a subset of the language of the detailed view. This type of information focuses just on the arrival of requests and not on their completion.

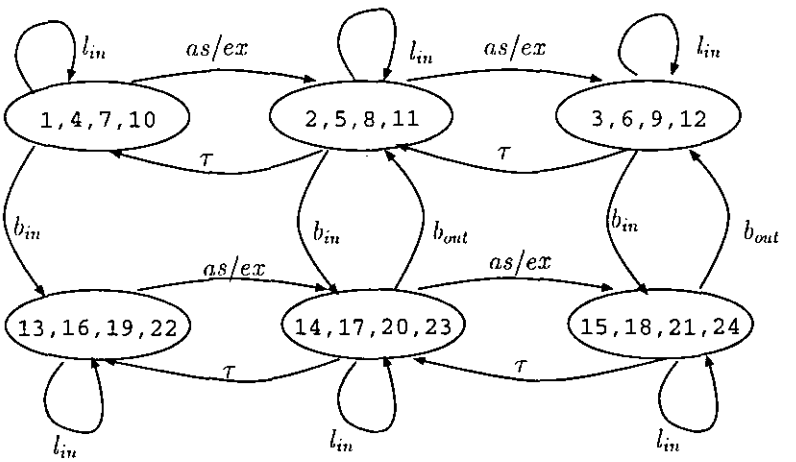


Fig. 8. Aggregated state space.

6 Research agenda

The approach shown in the previous sections has not to be considered as conclusive, rather as inspiring a new research agenda. In fact, while the basic idea is quite interesting, the current achievements show severe limitations. First of all, the approach strongly relies on a behavioral description and not on the net system level representation. Some results have being obtained for a restricted class of net system [4]. On the other hand, the emerging synthesis techniques

(rooted in the seminal notion of region [10]) could, to some extent, allow one to accept this approach. An interesting point could be the combination of synthesis techniques with the hierarchical approaches, e.g. the one presented in this paper. Secondly, the consistency criteria should be better understood in order to provide system modeler with aggregation policies that preserve the desired behavior.

On the other hand, other techniques to get more abstracted representations [8], while very powerful to check behavioral properties or construct well behaved models, exhibit the basic drawback of producing abstract representations of the target system which can be hardly interpretable as such by designers/users for sake of directing their future action (both in system design and use).

In any case, we foresee an interesting merge of interests in system design, coming from different application domains and different goals. In fact, the need of combining abstraction and context is an emerging requirement in the modeling of real systems, which are "open" by definition, and is a mandatory prerequisite when performance evaluation is concerned.

References

1. A. Agostini, G. De Michelis, and K. Petruni. Keeping workflow models as simple as possible. In G. De Michelis, C. Ellis, and G. Memmi, editors, *Proc. of the Workshop on Computer-Supported Cooperative Work, Petri Nets and related formalisms*, pages 11–29, Zaragoza, 1994.
2. L. Bernardinello, L. Pomello, and C. Simone. A class of morphisms for the refinement of EN systems. Research report, Dipartimento di Scienze dell'informazione, Università di Milano, Italy, 1996. submitted for pubblication.
3. J. Bowers, G. Button, and W. Sharrock. Workflow from within and without: technology and cooperative work on the print industry shopfloor. In H. Marmolin, Y. Sundblad, and K. Schimdt, editors, *Proc. of the Fourth European Conference on Computer-Supported Cooperative Work*, pages 51–66. Kluwer Academic Publishers, 1995.
4. P. Buchholz. A hierarchical view of GCSPN's and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15(3):207–224, July 1992.
5. P. Buchholz. A framework for the hierarchical analysis of discrete event dynamic systems. Habilitationsschrift, Universitaet Dortmund, 1996.
6. Workflow Management Coalition. Workflow reference model. 1994.
7. F. Di Cesare and M. Der Jeng. Synthesis for manifacturing systems integration. In F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F.B. Vernadat, editors, *Practice of Petri Nets in Manufacturing*. Chapman & Hall, London, 1993.
8. F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F.B. Vernadat, editors. *Practice of Petri Nets in Manufacturing*. Chapman & Hall, London, 1993.
9. P. Dourish. Accounting for system behaviour: representation, reflection and resourceful action. Technical report, Rank Xerox Research Center, 1995.
10. A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures: I and ii. *Acta Informatica*, 27(4):315–368, 1990.

11. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proc. of the 1995 conference on Organizational Computing Systems*, pages 10–21, San Jose', CA, 1995.
12. T. Kreifelts, E. Hinrichs, K. H. Klein, P. Seuffert, and G. Woetzel. Experiences with the DOMINO office procedure system. In L. Bannon, M. Robinson, and K Schmidt, editors, *Proc. of the Second European Conference on Computer-Supported Cooperative Work*, pages 117–130. Kluwer Academic Publishers, 1991.
13. T. Kreifelts and G. Woetzel. Distribution and error handling in an office procedure system. In G. Bracchi and D. Tsichritzis, editors, *Proc. of Office Systems: Methods and Tools*, pages 197–208. North-Holland, 1987.
14. R. Milner. *A calculus of Communicating systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg, 1980.
15. R. Pareschi, G. De Michelis, and S. Sarin. *Proceeedings of the first International Conference on Practical Aspects of Knowledge Management- Workshop on Adaptive Workflow*. Basel, 1996.
16. K. Schmidt. Of maps and scripts: the status of formal constructs in cooperative work. In *Proc. of the ACM conference on supporting group work*, pages 138–147, Phoenix', AZ, 1997. ACM Press, NY.
17. C. Simone and K. Schmidt. Taking the distributed nature of cooperative work seriously. In *Proc. of the 6th Euromicro Workshop on Parallel and Distributed Processing*, pages 295–301. IEEE-CS, 1998.
18. L.A. Suchman. *Plans and situated actions: the problem of human-machine communication*. Cambridg University Press, Cambridge, 1987.
19. W. Van der Aalst. Verification of workflow nets. In P. Azema and G. Balbo, editors, *Proc. of the* 18^{th} *Intern. Conference on Applications and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin Heidelberg, 1997.

This article was processed using the LaTeX macro package with LLNCS style

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.
96/02	P. de Bra and A. Aerts	Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12
96/03	W.M.P. van der Aalst	Parallel Computation of Reachable Dead States in a Free-choice Petri Net, p. 26.
96/04	S. Mauw	Example specifications in phi-SDL.
96/05	T. Basten and W.M.P. v.d. Aalst	A Process-Algebraic Approach to Life-Cycle Inheritance Inheritance = Encapsulation + Abstraction, p. 15.
96/06	W.M.P. van der Aalst and T. Basten	Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
96/07	M. Voorhoeve	Structural Petri Net Equivalence, p. 16.
96/08	A.T.M. Aerts, P.M.E. De Bra, J.T. de Munk	OODB Support for WWW Applications: Disclosing the internal structure of Hyperdocuments, p. 14.
96/09	F. Dignum, H. Weigand, E. Verharen	A Formal Specification of Deadlines using Dynamic Deontic Logic, p. 18.
96/10	R. Bloo, H. Geuvers	Explicit Substitution: on the Edge of Strong Normalisation, p. 13.
96/11	T. Laan	AUTOMATH and Pure Type Systems, p. 30.
96/12	F. Kamareddine and T. Laan	A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.
96/13	T. Borghuis	Priorean Tense Logics in Modal Pure Type Systems, p. 61
96/14	S.H.J. Bos and M.A. Reniers	The I^2C-bus in Discrete-Time Process Algebra, p. 25.
96/15	M.A. Reniers and J.J. Vereijken	Completeness in Discrete-Time Process Algebra, p. 139.
96/17	E. Boiten and P. Hoogendijk	Nested collections and polytypism, p. 11.
96/18	P.D.V. van der Stok	Real-Time Distributed Concurrency Control Algorithms with mixed time constraints, p. 71.
96/19	M.A. Reniers	Static Semantics of Message Sequence Charts, p. 71
96/20	L. Feijs	Algebraic Specification and Simulation of Lazy Functional Programs in a concurrent Environment, p. 27.
96/21	L. Bijlsma and R. Nederpelt	Predicate calculus: concepts and misconceptions, p. 26.
96/22	M.C.A. van de Graaf and G.J. Houben	Designing Effective Workflow Management Processes, p. 22.
96/23	W.M.P. van der Aalst	Structural Characterizations of sound workflow nets, p. 22.
96/24	M. Voorhoeve and W. van der Aalst	Conservative Adaption of Workflow, p.22
96/25	M. Vaccari and R.C. Backhouse	Deriving a systolic regular language recognizer, p. 28
97/01	B. Knaack and R. Gerth	A Discretisation Method for Asynchronous Timed Systems.
97/02	J. Hooman and O. v. Roosmalen	A Programming-Language Extension for Distributed Real-Time Systems, p. 50.
97/03	J. Blanco and A. v. Deursen	Basic Conditional Process Algebra, p. 20.
97/04	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra: Absolute Time, Relative Time and Parametric Time, p. 26.
97/05	J.C.M. Baeten and J.J. Vereijken	Discrete-Time Process Algebra with Empty Process, p. 51.
97/06	M. Franssen	Tools for the Construction of Correct Programs: an Overview, p. 33.
97/07	J.C.M. Baeten and J.A. Bergstra	Bounded Stacks, Bags and Queues, p. 15.

97/08	P. Hoogendijk and R.C. Backhouse	When do datatypes commute? p. 35.
97/09	Proceedings of the Second International Workshop on Communication Modeling, Veldhoven, The Netherlands, 9-10 June, 1997.	Communication Modeling- The Language/Action Perspective, p. 147.
97/10	P.C.N. v. Gorp, E.J. Luit, D.K. Hammer E.H.L. Aarts	Distributed real-time systems: a survey of applications and a general design model, p. 31.
97/11	A. Engels, S. Mauw and M.A. Reniers	A Hierarchy of Communication Models for Message Sequence Charts, p. 30.
97/12	D. Hauschildt, E. Verbeek and W. van der Aalst	WOFLAN: A Petri-net-based Workflow Analyzer, p. 30.
97/13	W.M.P. van der Aalst	Exploring the Process Dimension of Workflow Management, p. 56.
97/14	J.F. Groote, F. Monin and J. Springintveld	A computer checked algebraic verification of a distributed summation algorithm, p. 28
97/15	M. Franssen	λP-: A Pure Type System for First Order Loginc with Automated Theorem Proving, p.35.
97/16	W.M.P. van der Aalst	On the verification of Inter-organizational workflows, p. 23
97/17	M. Vaccari and R.C. Backhouse	Calculating a Round-Robin Scheduler, p. 23.
97/18	Werkgemeenschap Informatiewetenschap redactie: P.M.E. De Bra	Informatiewetenschap 1997 Wetenschappelijke bijdragen aan de Vijfde Interdisciplinaire Conferentie Informatiewetenschap, p. 60.
98/01	W. Van der Aalst	Formalization and Verification of Event-driven Process Chains, p. 26.
98/02	M. Voorhoeve	State / Event Net Equivalence.
98/03	J.C.M. Baeten and J.A. Bergstra	Deadlock Behaviour in Split and ST Bisimulation Semantics, p. 15.
98/04	R.C. Backhouse	Pair Algebras and Galois Connections, p. 14
98/05	D. Dams	Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers. P. 22.
98/06	G. v.d. Bergen, A. Kaldewaij V.J. Dielissen	Maintenance of the Union of Intervals on a Line Revisited, p. 10.