Information Control Nets as Processes: Qualitative Analysis

M. Debbabi, Fouzia Bouguerch and Nadia Tawbi

Concurrent Engineering 1997; 5; 47
DOI: 10.1177/1063293X9700500106

CONCURRENT ENGINEERING: Research and Applications

Information Control Nets as Processes: Qualitative Analysis

M. Debbabi,* Fouzia Bouguerch and Nadia Tawbi

Computer Science Department, Laval University, Québec G1K 7P4, Canada

Received 14 April 1996; accepted in revised form 15 September 1996

Abstract: In this paper, we address the formal verification in workflow systems. More precisely, we consider the automatic verification of one of the most prominent workflow models, namely the Information Control Nets (ICNs) model. We present a powerful method for the qualitative analysis of ICNs. The analysis proposed rests on an observed analogy between ICNs and usual process algebra. Actually, ICNs are translated into CCS process algebra agents. The properties to be verified are expressed in a temporal logic, the propositional modal ν-calculus. The verification is done thanks to model-checking algorithms.

1. Introduction

Recently, a set of products commonly referred to as *groupware* started to emerge. These products were of two categories: document-oriented, devised to support shared authorship of documents; and process-oriented, to assist groups of people to carry out a given process. The second category was mainly designed to support ad-hoc activities by professionals and technical staff. Their users have been very enthusiastic about the benefits and wanted to give this concept to people elsewhere in the organization. These systems are now referred to as *workflow* systems. In the literature, some definitions show that a workflow system automates processes, routes documents through a corporation, and coordinates and logs activities performed by actors (people or systems). An actor is said to play a role. One actor may play several roles, and one role may be played by several actors. In addition, a workflow can sometimes be used to redesign existing processes.

Nowadays, it is well known that the use of workflow software contributes significantly to increase productivity and flexibility of administrative organizations. More than fifty commercial products have been launched on the market. Most of the computer builders are represented. For instance, a list of some products is given in Table 1.

Generally these products are made of two main components: a *modeler* that allows the specification of workflow procedures and a *performer* that performs the tasks specified in the model. Each activity or task is represented by a set of procedures. A procedure is made of actions that represent the steps of the corresponding task. Actually, a procedure is represented as an oriented graph whose nodes are either action nodes or control nodes. Action nodes are labeled with action identifiers. Control nodes control the task flow through the procedure.

A great deal of interest has been expressed in the analysis of workflow models. Such an interest is motivated by the following reasons:

- First, there is a need to detect and correct specification errors at an early stage of the design process of workflow procedures.
- Second, once a model is devised, it is judicious to optimize and simplify it. This will be worthwhile and may have significant impacts especially at the dynamic level of its execution.
- Third, an ultimate validation step is required so as to ensure that the optimized, error-free model is exactly the one desired.

Two main analysis techniques have been advanced so far: quantitative analysis, i.e., simulation, and qualitative analysis, i.e., formal verification. Simulation is a relatively simple and efficient analysis technique. However, it is unable to guarantee the correctness of procedure behaviors especially when we deal with safety-critical workflow processes. At this level, there is a crucial need for analyses that formally establish (formal verification) the correctness of the relevant behaviors.

The main novelty of this paper is to address the formal qualitative analysis in workflow systems. More precisely, the intent of this paper is to present a formal qualitative verification method for one of the most prominent workflow models, namely *Information Control Nets* (ICNs). This method emerged from the observation of a symmetry between ICNs and usual process algebra agents. In this paper, we discuss in detail such an analogy and present a translation method that allows one to encode a given ICN as an agent over some process algebra. Furthermore,

*Author to whom correspondence should be addressed.

1063-293X/97/01 0047-11 $10.00/0

Table 1. List of workflow products.

Workflow Product	Company
Workflow Analyst	Action Technologies
ProcessIT	AT and T GIS
BeyondMail	Bayan Systems
Flowpath	Bull
WinWork	Centre-file
EPIC/WF	Computron
Workflow	CSE-Systems
FormFlow	Delrina Corporation
AWD	DST Systems
Visual Workflo	FileNet Corporation
Workflow 2020	Fischer Int. Systems
FlowMark	IBM
POWERFLOW	ICL
LotusForms	Lotus Development
FlowMaster	Olivetti
WorkParty	Siemens Nixdorf
FlowFile	Standard Patforms
PPDM	Tandem Computers
InfoImage	Unisys
OPEN/Workflow	Wang
InConcert	X Soft

we address the formal verification of ICNs via model-checking techniques.

This paper is structured as follows: Section 2 is devoted to the presentation of some basic definitions related to workflows. In Section 3, we present a description of an analogy between ICNs and process algebra. In Section 4, we introduce CCS (Calculus for Communicating Systems) algebra concepts. In Section 5, a description of a new method of translating an ICN into a CCS agent is presented. Section 6 presents our qualitative analysis method of ICNs. Some concluding remarks together with a future work discussion are ultimately sketched in Section 7 as a conclusion.

2. Workflow Systems

Workflow systems are software tools intended to assist a group of people collaborating on the same project. The major aim is to help companies and organizations manage tasks in distributed environments. The "Workflow tools" could assist at different stages of a task's life-cycle: specification, execution, coordination.

Workflow systems differ according to the type of process they are designed to deal with. Thus, three types of workflow systems can be described [7]:

- The first is image-based Workflow Systems. These systems automate the flow of paper through an organization, by transferring the paper to digital "images." They are used in the "imaging" technology, and well represent the routing and processing of digitized images.
- The second is form-based Workflow Systems. These systems are useful to automate the routing of forms throughout an organization. These forms, unlike images, are text-based and consist of editable fields. The automatic routing of forms is directed by the information entered on the form. Furthermore, these form-based systems can notify or remind people when action is due. Thus, form-based Workflow Systems are characterized by a higher level of capability than image-based workflow systems.
- The third is coordination-based Workflow Systems. These systems provide a framework for coordination of actions to facilitate the completion of work. The framework affects the domain of human concerns (business processes), rather than the optimization of information or material processes. Coordination-based Workflow systems ensure the improvement of organizational productivity. They treat the issues necessary for customer satisfaction, rather than automate procedures that are not closely related to customer satisfaction.

Generally, these products have two main components: a modeler that allows the representation of the procedures, and a performer that performs the tasks specified in the model. Each activity or task is represented by a set of procedures. A procedure consists of actions (nodes) that represent the steps of the procedure and of control nodes that control the task flow through the procedure. The procedure is specified in a workflow model. The two prominent workflow models are:

- High level Petri nets (abbreviations and extensions) [14,15].
- Information Control Nets (ICNs).

Notice that ICNs are more expressive than colored Petri nets. ICNs are temporized, colored (typed tokens), and allow the modelization of data aspects of some abstract level. Besides, they have several types of control nodes while Petri nets have only transitions as control nodes.

In the sequel of this paper, ICNs and their formal qualitative analysis are considered. The choice of ICNs is motivated by the fact that initially we have been working on the integration of qualitative analysis in the Bull's workflow system FlowPath. The latter rests on the use of ICNs on a client-server distributed platform.

First of all, let us recall briefly the model of Information Control Nets (ICNs). The steps in a workflow procedure are called *activities*. An activity models an action. Pictorially, an action is represented by a large open circle as in Figures 1 and 2. The action name is located either inside or outside the circle. The model offers the possibility of sequencing activities. This is done graphically by drawing an arc from the first activity to be executed to the sequel of the procedure [see Figure 1(a)]. The alternative (disjunction) of n workflows (or ICNs) $workflow_1, \ldots, workflow_n$ is graphically represented by Figure 1(b). Notice that the small open circles are control nodes and denote the alternative composition. The underlying semantics stipulates that one of the ICNs $workflow_1, \ldots, workflow_n$ will be selected and ex-

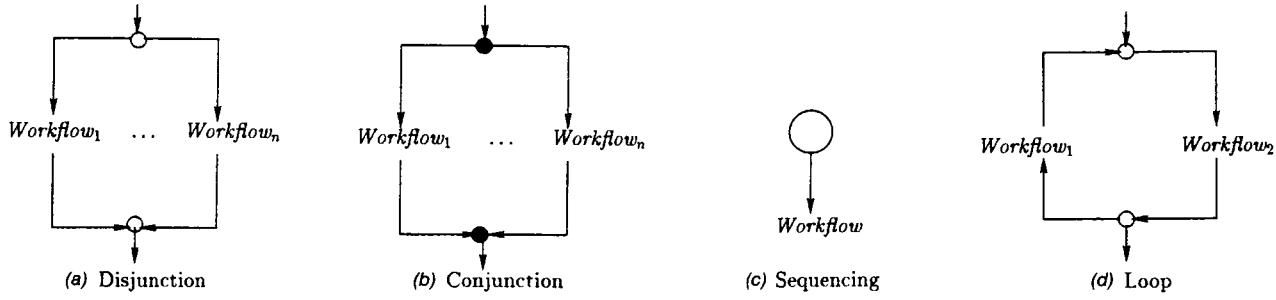


Figure 1. Basic ICN Combinators.

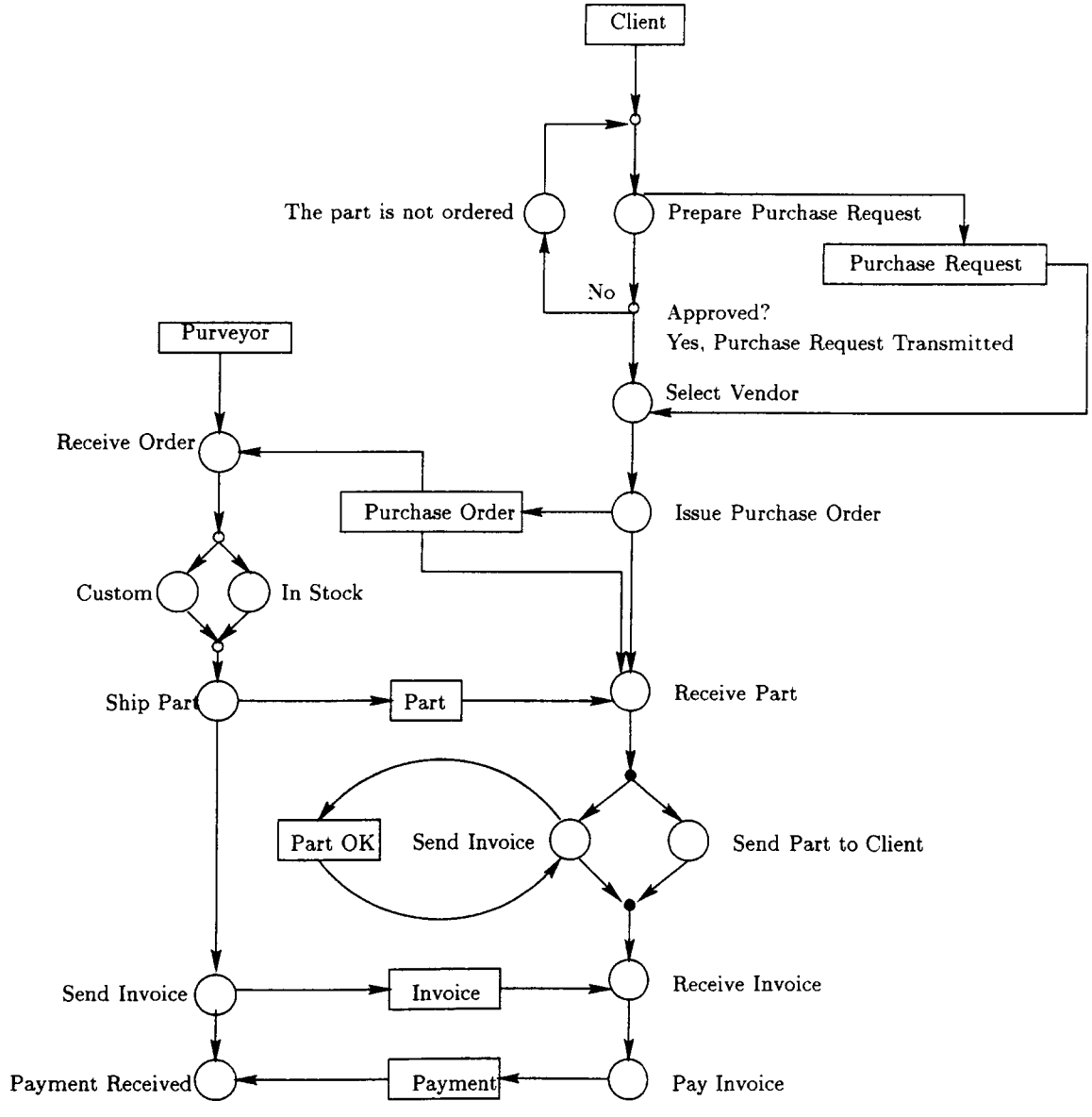


Figure 2. An ICN that models a purchase procedure.

ecuted. The parallel composition (conjunction) of n workflows (or ICNs) $workflow_1, \ldots, workflow_n$ is graphically represented by Figure 1(c). Notice that the small filled circles are control nodes and denote the parallel composition. The first control node denotes a fork-like operation while the other denotes a join-like operation (a synchronization). Semantically, parallel composition is interpreted as interleaving. The model also allows the definition of looping ICNs. Figure 1(d) illustrates a loop that behaves as follows: first the workflow $workflow_1$ is executed until termination. At that time, either the whole ICN terminates or the workflow $workflow_2$ is executed and upon its termination, we recurse. Data are graphically represented by a rectangle. An outgoing arc that is drawn from a given activity to a corresponding rectangle denotes that some data is produced by this activity. An arc that is issued from a corresponding rectangle to a given activity denotes that some data is used by that activity. Figure 2 is an illustration of the use of all those ICN combinations. It shows an ICN that represents a purchasing procedure where a client needs to write a purchase request so as to order a given part. If the purchase is approved, then a Purchase Order is issued and it is sent to a selected vendor. The vendor supplies the part, packages with a shipping list, and sends it to the purchaser department. When the part is received, it is sent to the client and an invoice is sent to the account payable department after verifying that all parts of the order were received by comparing the shipment list with the shipment contents. Meanwhile, the vendor will have prepared an invoice and sent it to the purchaser's account payable department. When this latter received an invoice and notified that the part was received, then it will issue payment to the vendor. Notice that the ICN in this figure models how work flows through two organizations (the purchasing organization and the vendor organization).

The next section discusses the analogy between ICNs and classical process algebra.

3. From ICNs to Processes

ICNs have been devised as a framework for modeling office work coordination systems. They have been used as the basis for the implementation and use of these systems. ICNs are intended to represent control flow and data flow. ICNs have been used in industry, and are valuable for capturing office procedures [12], for mathematical analyses [4], and for simulation [13]. Actually, ICNs are a general purpose model that could be used to specify plenty of systems that are characterized by being made of activities (actions) and data together with a control structure.

In the literature, several models have been advanced for specifying systems. Among the most prominent models, one can cite process algebra calculi such as CCS (Calculus of Communicating Systems) [8,9], CSP (Communicating Sequential Processes) [1,2,3,6], and the π-calculus [11,10].

A process algebra is an algebraic formalism reduced to basic constructors that allow one to specify concurrent and distributed systems. In general, process algebras are semantically interpreted over a transitional (operational) semantics.

The work presented within this paper rests heavily on an observed analogy between ICNs and process algebra calculi. Actually:

- ICN activities are similar to atomic actions in process algebra.
- The "OR" control nodes are similar to the alternative combinators.
- The "AND" control nodes are similar to parallel composition of agents.
- Finally, loops in ICNs are similar to some extent, to recursively defined agents.

In this paper, we stress the analogy between pure ICNs (without data nodes) and CCS process algebra. Moreover, we present a full formalization of the translation of ICNs into CCS agents. The choice of CCS is motivated by the fact that the latter process algebra is the simplest one that is able to cover the space of pure ICNs.

In the next section, we recall briefly the CCS process algebra.

4. CCS Process Algebra

CCS calculus has been proposed by Robin Milner as a description language of communicating processes. The word process stands for: a machine, a memory, a procedure, etc. The communication is the interaction means between processes.

4.1 Syntax

Let us consider an alphabet Σ denoting atomic actions names ranged by a, b, etc. We denote by $\overline{\Sigma}$ the alphabet of co-names (complementary actions) of actions in Σ ranged by $\bar{a}$, $\bar{b}$, etc. Then we set $\mathcal{L} = \Sigma \cup \overline{\Sigma})$ $\mathcal{L}$ as a set of labels where $\overline{\bar{a}} = a$. $\mathcal{L}$ is extended with a *silent action* denoted τ (has no complement). The resulting alphabet is denoted Act ($Act = \mathcal{L} \cup \{\tau\}$). A CCS process could be denoted as indicated by the following BNF grammar:

$$P ::= nil \,|\, a.P \,|\, P_1 + P_2 \,|\, P_1 \,|\, P_2 \,|\, X = P$$

The constant *nil* models an inactive process, i.e., that does not perform any action without being deadlocked. The term $a.P$ denotes the process that can execute the action a and then its behavior becomes the same as P behavior. The operator "." is the *prefixing* operator. The agent $a.P$ stands for the agent that first performs a and then behaves as P. The agent $P_1 + P_2$ denotes an *external choice* (an alternative) between two processes P_1 and P_2. The agent $P_1 \,|\, P_2$

Table 2. Transitional semantics of CCS.

(Prefix)	$a.P \xrightarrow{a} P$	(Communication)	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$
(Definition)	$\dfrac{P \xrightarrow{a} P'}{X \xrightarrow{a} P'} X = P$	(Interleaving1)	$\dfrac{P \xrightarrow{a} P'}{P\|Q \xrightarrow{a} P'\|Q}$
(Summation1)	$\dfrac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$	(Interleaving2)	$\dfrac{Q \xrightarrow{a} Q'}{P\|Q \xrightarrow{a} P\|Q'}$
(Summation2)	$\dfrac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$		

denotes *parallel composition of* P_1 and P_2. Finally, the agent $X = P$ denotes the *definition* and the *binding* of processes, that is to say the identifier X is bound to the value P which consists of the definition of X. Notice that $X = P$ may be also written *rec* $X.P$ which stands for the definition of a recursive process P. The variable X is the recursion variable. Accordingly, *rec* $X.P$ is equivalent to $P[(rec\ X.P)/X]$, i.e., the process P in which X has been substituted by *rec* $X.P$ (unfolding the recursion).

4.2 CCS Transitional Semantics

Robin Milner presents in Reference [9] a transitional or operational semantics for CCS. Actually, this operational semantics is a description of a transition system that behaves as an interpreter of the language CCS. It is based on a ternary relation denoted "$\_ \xrightarrow{\_} \_$". A sequent of the form $P_1 \xrightarrow{a} P_2$ stands for the evolution of P_1 into P_2 after the execution of action a. The inference rules defining the transitional semantics are given in Table 2.

4.3 Expansion Theorem

The expansion theorem allows the reduction of a process to its normal form [9]. Thanks to this theorem, the parallelism combinators are rewritten into choices. The constructors of the normal form are prefixing ".", sum and choice "Σ".

Theorem 4.1 *Let P be $P = (P_1|\ldots|P_n)$, where $n \geq 1$. Then we have:*

$$P = \Sigma\left\{a.(P_1|\ldots|P_i'|\ldots|P_n) : P_i \xrightarrow{a} P_i'\right\} + \Sigma\left\{\tau.(P_1|\ldots|P_i'|\ldots|P_j'|\ldots|P_n) : P_i \xrightarrow{a} P_i',\ P_j \xrightarrow{\bar{a}} P_j',\ i < j\right\}$$

The symbol Σ stands for the agent sum "+". The agent $P_1 + \ldots P_n$ is represented by $\Sigma_{i=1}^{n} P_i$.

4.4 Bisimulation

In CCS, we find a notion called observation equivalence of processes, which expresses the equivalence of processes whose external communications follow the same pattern but whose internal behavior may differ widely. The equivalence is related to a notion called bisimulation, which we introduce in this section. Let us first introduce the relation "$\_ \stackrel{=}{\Rightarrow} \_$" which consists of a generalization for the transition relation "$\_ \xrightarrow{\_} \_$".

Definition 4.2 *Let μ be $\mu = a_1, \ldots, a_n \in \mathscr{L}^*$; we can write $P \stackrel{\mu}{\Rightarrow} Q$ if:*

$$P(\xrightarrow{\tau})^* \xrightarrow{a_1} (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* \xrightarrow{a_1} (\xrightarrow{\tau})^* Q$$

Now, we can introduce the bisimulation relations commonly called observational equivalence.

Definition 4.3 *We define inductively the family of equivalence relations $\approx_n$ ($n \geq 0$) as follows:*

1. $\forall P, Q$: $P \approx_0 Q$
2. $P \approx_{n+1} Q \Leftrightarrow \forall \mu \in \mathscr{L}^*$:
   (a) $P \stackrel{\mu}{\Rightarrow} P' \Rightarrow \exists Q' | Q \stackrel{\mu}{\Rightarrow} Q' \wedge P' \approx_n Q'$
   (b) $Q \stackrel{\mu}{\Rightarrow} Q' \Rightarrow \exists P' | P \stackrel{\mu}{\Rightarrow} P' \wedge P' \approx_n Q'$

Definition 4.4 *Let P and Q be two CCS processes CCS; P and Q are said to be observationally equivalent, or bisimilar, and we write $P \approx Q$, if and only if:*

$$\forall n \geq 0, \quad P \approx_n Q$$

Thus, the observational equivalence, or the bisimulation is defined as follows:

$$\approx = \bigcap_{n=0}^{\infty} \approx_n$$

In the next section, we present a description of our method of translating an ICN into a CCS agent.

5. Translation into CCS Algebra

In order to perform qualitative analysis of ICNs, we have elaborated a new method that is based on the translation of a given ICN into a CCS agent. Afterwards, the process generated is subjected to formal verification using model-checking. The properties to be verified are expressed into the modal propositional μ-calculus.

The approach underlying the translation of an ICN into a CCS agent is based on the analogy between ICNs and CCS process algebra mentioned previously. Although this analogy is attractive, some technicalities need to be clarified. The first one is a matter of expressiveness. The ICNs model allows sequencing of tasks, while there is no sequencing of processes in CCS. The process algebra CCS has a prefixing operator "." which is less expressive than the usual sequencing. To overcome this limitation, we expressed the sequencing of two processes using recursion, choice, prefixing, and/or nil constructors. This is done by replacing all the nil subexpressions of the first process by the whole expression that represents the second process. Graphically, a process can be viewed as a finitely-branching rooted graph (a synchronization tree following Milner's terminology [8,9]). Hence, the sequencing of two processes amounts to the concatenation of the second process graph to all the terminal branches of the first process (those branches that all terminate with leaves). The second technicality to resolve is related to the simultaneous presence of the parallel composition (AND nodes) and task sequencing in the ICNs model. The application of the sequencing technique above seems to be adequate. However, its application confines us to use only nil, prefixing, choice, and recursion. In other words, the sequencing technique is inapplicable in the presence of parallel composition. In order to eliminate the CCS concurrency operator, one has to compute merely an equivalent normal form of the original process using the expansion theorem of CCS. Figure 3(a) is a graphical illustration of the sequencing technique. For instance, given a process $P = a.nil + b.c.nil + d.e.P$ and given a process $Q = f.nil + g.nil$, then the sequencing of the two processes is $P; Q = R = a.(f.nil + g.nil) + b.c.(f.nil + g.nil) + d.e.R$. Figure 3(b) is a graphical illustration of the use of sequencing in the presence of parallel composition. Let P be a process $a.nil + b.nil$ and Q a process $c.nil$. The parallel composition of the two processes is $P|Q = (a.nil + b.nil)|c.nil$. We apply the expansion theorem on the latter process to compute its normal form. This process is then reduced into its normal form that is $a.c.nil + b.c.nil + c.(a.nil + b.nil)$. Let R be the process $d.f.nil + e.nil$. The sequencing of the two processes $P|Q$ and R is the process $(P|Q); R = a.c.(d.f.nil + e.nil) + b.c.(d.f.nil + e.nil) + c.(a.(d.f.nil + e.nil) + b.(d.f.nil + e.nil))$.

Finally, let us mention that our ICNs correspond to finite state processes which are appropriate for model-checking.

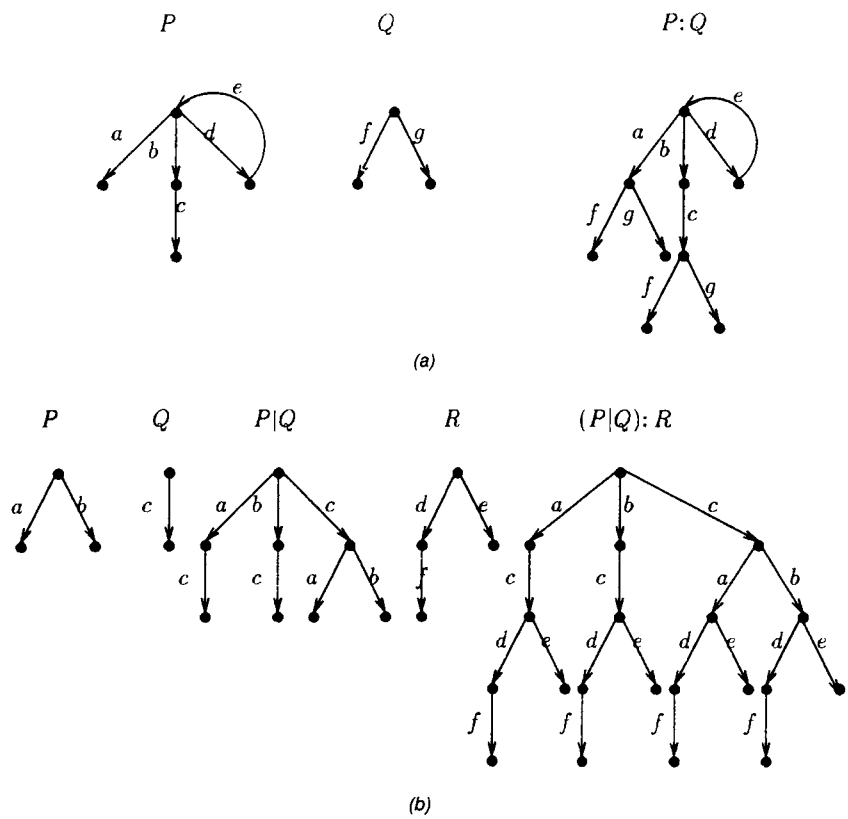


Figure 3. The translation schema of processes sequencing.

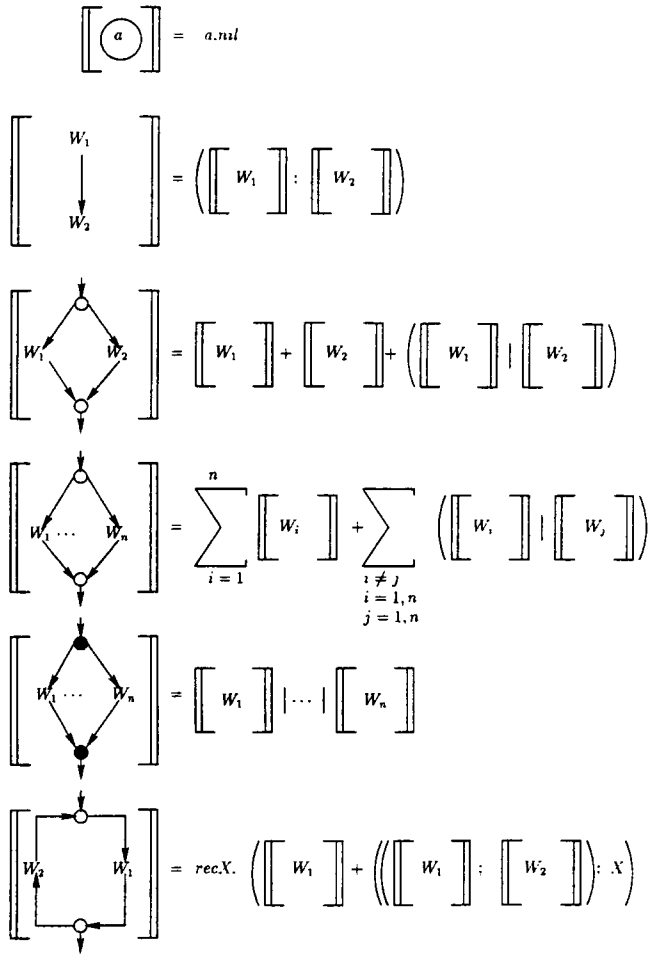


Figure 4. The translation schema of ICNs into CCS agents.

In fact, this is achieved by providing joins on the AND or OR nodes for ICNs. These joins play a role of a good parenthesizing on these nodes. Once we open a parenthesis (outgoing arcs issued from an OR or an AND node), the syntax stipulates that we must close it (in-going arcs to OR or AND nodes). This is a crucial condition which guarantees the termination of the translation and also provides a simple and efficient model-checking during the analysis.

The translation is compositional and syntax directed. A given task a is translated into the CCS agent $a.nil$, i.e., the agent executes action a then successfully terminates. The sequencing of two workflows $W1$ and $W2$ is translated as the sequencing of the corresponding processes. The translation of the disjunction of two workflows $W1$ and $W2$ is interpreted as a choice between the execution of the process attached to $W1$, or the execution of the process attached to $W2$, or the simultaneous (parallel) execution of both processes. The translation of the conjunction of a finite list of ICNs amounts to the parallel execution of the corresponding processes. A looping ICN is translated into a recursive process that executes the process that corresponds to the first sub-ICN and then either terminates successfully, or executes the second sub-ICN and recurses. The translation algorithm of ICNs into CCS agents is presented in Figure 4.

The next section presents a description of our qualitative analysis method with a brief recall of the modal μ-Calculus.

6. Qualitative Analysis

In this section, we present our implemented qualitative analysis method. Almost all workflow systems use quantitative analysis (simulation). Our analysis is useful to verify the traditional properties, understand the behavior of sophisticated nets, and prove equivalences between ICNs in order to optimize them. The analysis method is based on model-checking algorithms.

The first use schema is illustrated in Figure 5. In this case, the intent is to verify that a given ICN, say ICN1, satisfies some property. This is carried out as follows: first, we translate ICN1 into the CSS agent, say agent1. Second, we express the property to be verified into the modal propositional μ-calculus. This property and the CCS agent1 are then submitted as an input to a model-checker. Once the formal verification of the property on the CCS agent1 is completed, the model-checker provides as an output a boolean result (True or False) corresponding, respectively, to the satisfaction or not of this property by the CCS agent1. A second use schema is illustrated in Figure 5. In this case, the intent is to verify that two ICNs, say ICN1 and ICN2, are equivalent up to some CCS equivalence. The analysis schema is interesting, especially when one of the two ICNs is suspected to be an optimized form of the other. In this case, establishing the equivalence confirms the previous statement. The two ICNs, ICN1 and ICN2, are submitted to the translator of an ICN into a CCS agent and we get, respectively, two processes, say agent1 and agent2. The two processes are submitted as inputs to the model-checker, and again the answer is the boolean value.

We experimented with the model propositional μ-calculus as a logic for expressing properties. This logic is based on an extension of Hennessy-Milner logic [5,9]. The latter is a

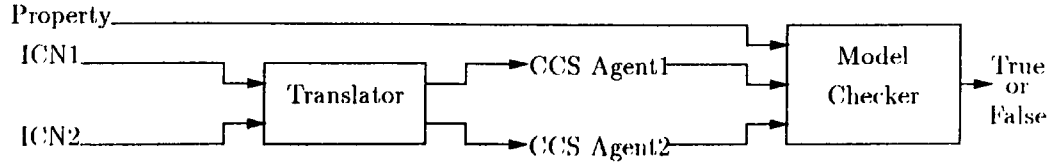


Figure 5. The qualitative analysis schema.

modal logic dedicated to labelled transition systems. This logic introduces modalities parameterized by actions expressing the ability to execute these actions. Many extensions to this logic have been proposed. These extensions have led to the logic commonly called modal μ-calculus [16]. We adopted the last logic as a verification framework. A brief introduction of the modal propositional μ-calculus logic is given as follows: formulas in μ-calculus are constructed using the common boolean operators, modal operators expressing the ability of a process to execute some actions as well as fix-point operators in order to express properties on infinite paths. The syntax of formulas is given by the following BNF grammar:

$$\begin{array}{rl} \Phi ::= & \mathit{True} \mid \mathit{False} \\ \mid & \neg\Phi \mid \Phi \vee \Phi' \mid \Phi \wedge \Phi' \\ \mid & [K]\Phi \mid \langle K\rangle\Phi \\ \mid & \mu Z.\Phi \quad \nu Z.\Phi \end{array}$$

The constants *True* and *False* stand for the usual boolean constants, $\neg$ for the negation, $\wedge$ for the conjunction, and $\vee$ for the disjunction. The formula $[K]\Phi$ means that each process resulting from an execution of some action in the set K, satisfies the formula Φ. The formula $\langle K\rangle\Phi$ means that there exists a process that is the result of executing an action in the set K, and that satisfies the formula Φ. The operator $\nu Z.$ is a binder, binding free occurrences of Z in Φ. Actually, the fixpoint formulas are abbreviated. They could be unfolded as follows:

$$\mu Z.\Phi = \bigvee_{i=0}^{\infty} \Phi^i$$

$$\Phi^0 = \mathit{False}$$

$$\Phi^{i+1} = \Phi[\Phi^i/Z]$$

$$\nu Z.\Phi = \bigwedge_{i=0}^{\infty} \Phi^j$$

$$\Phi^0 = \mathit{True}$$

$$\Phi^{j+1} = \Phi[\Phi^j/Z]$$

The semantics of this logic is given by a satisfaction relation. For any formula Φ, we define when a process P has, or satisfies, the property Φ, and we write $P \models \Phi$ to denote this statement. If P fails to have the property Φ, we write $P \not\models \Phi$. The satisfaction relation $\models$ is defined inductively on the structure of formulas:

$$\begin{array}{l} P \not\models \mathit{False} \\ P \models \mathit{True} \\ P \models \neg\Phi \quad \text{iff } P \not\models \Phi \\ P \models \Phi \wedge \Psi \text{ iff } P \models \Phi \textit{ and } P \models \Psi \\ P \models \Phi \vee \Psi \text{ iff } P \models \Phi \textit{ or } P \models \Psi \\ P \models [K]\Phi \quad \text{iff } \forall Q \in \{P' \mid P \xrightarrow{a} P' \\ \qquad\qquad \text{and } a \in K\}.Q \models \Phi \\ P \models \langle K\rangle\Phi \text{ iff } \exists Q \in \{P' \mid P \xrightarrow{a} P' \\ \qquad\qquad \text{and } a \in K\}.Q \models \Phi \end{array}$$

Every process has the property *True*. A process has the property $\neg\Phi$ when it fails to have the property Φ. It has the property $\Phi \wedge \Psi$ when it has both properties Φ and Ψ, while it has the property $\Phi \vee \Psi$ when it has the property Φ or Ψ. Finally, a process satisfies $[K]\Phi$ if after every performance of any action in K, all the resulting processes have the property Φ, while it satisfies $\langle K\rangle\Phi$ if after every performance of any action in K some of the resulting processes have the property Φ. In the context of this work, we experienced the verification of several kinds of properties:

- Verification of behavioral equivalences of workflows. There are many equivalence checks, e.g., bisimulation, test equivalence, etc.
- Liveness and safety properties expressed in modal propositional μ-calculus. Some examples of properties we verified are the following:

$$\begin{array}{ll} \mathit{Even}\ (X) & = \mu Y.X \vee \langle - \rangle Y \text{ Eventually } X \\ \mathit{Never}\ (X) & = \neg \mathit{Even}\ (X) \text{ Never } X \\ \mathit{Always}\ (X) & = \nu Y.X \wedge [-]Y \text{ Always } X \end{array}$$

- Computing the size of a behavioral graph associated with an ICN.
- Computing the execution traces of an ICN.

As an example, let us consider the ICN in Figure 6. This ICN has four nested parallel compositions. This makes the rigorous understanding of all its behaviors almost impossible as will be shown in the sequel of this paper. Even though with a graphic representation, understanding the behavior of this ICN is far from being obvious. The complexity grows exponentially with the level of nested parallel compositions. A combinatorial explosion of the number of states is observed and this is due to the interleavings generated by the AND nodes and control nodes. Once this ICN is loaded in our tool, it is possible to translate it into a CCS agent. We then have part of the equational characterization listed in Table 3.

We implemented our tool in *C* on a Unix workstation. Our tool is structured as follows: two lexical analyzers, implemented in LEX, are used to split and generate from a textual representation of an ICN a list of tokens or lexical units and to perform syntactic substitutions in the case of an equational representation. This form enables an easy entry of big size ICNs, syntactic analyzers, implemented in YACC, in order to check the conformance with the grammar, and furthermore, to perform a semantical analysis for the generation of CCS agents. This generation is based on semantic actions associated with the grammar production rules. A *C* library for the handling of CCS agents has been im-

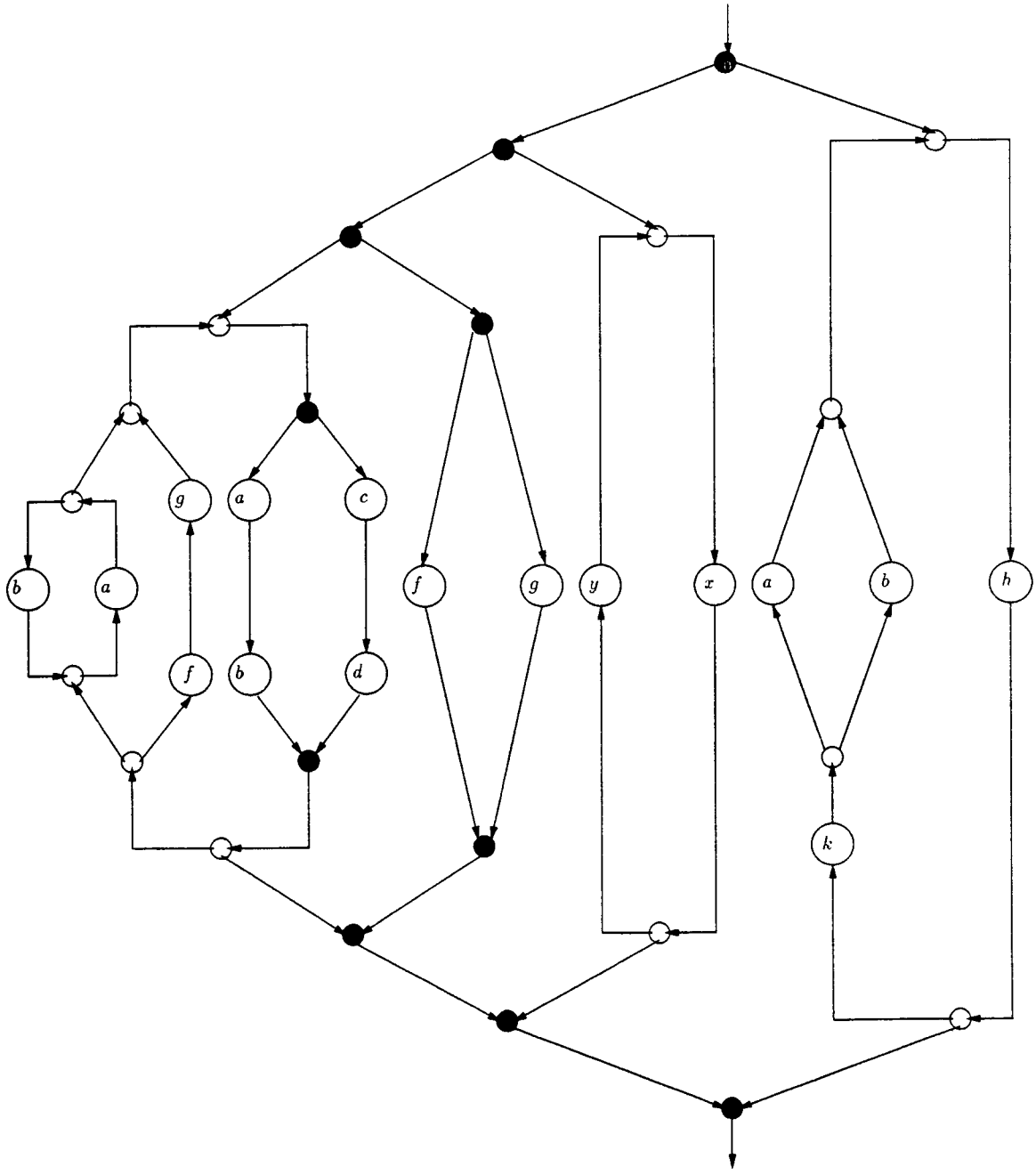


Figure 6. A sophisticated ICN.

Table 3. Equational characterization.

$$
\begin{aligned}
X1 &= a.X2 + c.X3 + a.X4 + c.X5 + f.X6 + g.X7 + x.X8 + x.X9 + h.X10 + h.X11\\
X2 &= b.X12 + c.X13 + f.X14 + g.X15 + x.X16 + x.X17 + h.X18 + h.X19\\
X3 &= a.X13 + d.X20 + f.X21 + g.X22 + x.X23 + x.X24 + h.X25 + h.X26\\
X4 &= b.X27 + c.X28 + f.X29 + g.X30 + x.X31 + x.X32 + h.X33 + h.X34\\
X5 &= a.X28 + d.X35 + f.X36 + g.X37 + x.X38 + x.X39 + h.X40 + h.X41\\
X6 &= a.X14 + c.X21 + a.X29 + c.X36 + g.X42 + x.X43 + x.X44 + h.X45 + h.X46\\
X7 &= a.X15 + c.X22 + a.X30 + c.X37 + f.X42 + x.X47 + x.X48 + h.X49 + h.X50\\
X8 &= a.X16 + c.X23 + a.X31 + c.X38 + f.X43 + g.X47 + h.X51 + h.X52\\
X9 &= a.X17 + c.X24 + a.X32 + c.X39 + f.X44 + g.X48 + y.X1 + h.X53 + h.X54\\
X10 &= a.X18 + c.X25 + a.X33 + c.X40 + f.X45 + g.X49 + x.X51 + x.X53\\
X11 &= a.X19 + c.X36 + a.X34 + c.X41 + f.X46 + g.X50 + x.X52 + x.X54 + k.X55\\
X12 &= c.X56 + f.X57 + g.X58 + x.X59 + x.X60 + h.X61 + h.X62\\
X13 &= b.X56 + d.X63 + f.X64 + g.X65 + x.X66 + x.X67 + h.X68 + h.X69\\
X14 &= b.X57 + c.X64 + g.X70 + x.X71 + x.X72 + h.X73 + h.X74\\
X15 &= b.X58 + c.X65 + f.X70 + x.X75 + x.X76 + h.X77 + h.X78\\
&\vdots \quad \vdots\\
X1033 &= a.X360 + a.X1021 + x.X1040 + x.X1041 + a.X947 + b.X947\\
X1034 &= a.X361 + a.X1022 + g.X1040 + a.X948 + b.X948\\
X1035 &= a.X362 + a.X1023 + g.X1041 + y.X1010 + a.X949 + b.X949\\
X1036 &= a.X363 + a.X1024 + f.X1040 + a.X952 + b.X952\\
X1037 &= a.X364 + a.X1025 + f.X1041 + y.X1015 + a.X953 + b.X953\\
X1038 &= b.X1040 + a.X961 + b.X961\\
X1039 &= b.X1041 + y.X1021 + a.X962 + b.X962\\
X1040 &= a.X597 + a.X1038 + a.X1002 + b.X1002\\
X1041 &= a.X598 + a.X1039 + y.X1033 + a.X1003 + b.X1003
\end{aligned}
$$

plemented and enables all the authorized compositions in CCS (sequencing, choice, recursion, parallel composition, etc). The internal representation is based on graphs (represented using binary decision diagrams). These graphs correspond to the theoretical synchronization tree notion introduced by Milner in order to build a denotational semantics for CCS.

7. Conclusion

We reported in this paper a qualitative analysis technique for Information Control Nets. The technique elaborated is based on an analogy between ICNs and process algebras. Actually, ICNs are translated into CCS agents, properties are expressed into the propositional modal μ-calculus, and verification is performed using model-checking algorithms. As a future work, we plan to extend the analysis that we described above to consider the temporal and data constructs of the ICNs. We note that the process algebra CCS is not able to express them. So in order to realize this extension, we need to consider more expressive process algebras.

Acknowledgements

We would like to thank David Lifchitz for the implementation of the first prototype and also for his valuable comments that allowed us to significantly improve the ideals in this paper.

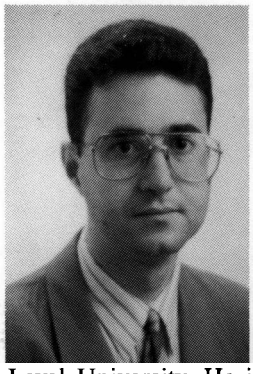
References

1. Brooks, S. D. 1985. "On the Relationship of CCS and CSP." Technical Report, Carnegie-Mellon University.
2. Brooks, S. D., C. A. R. Hoare, and A. W. Roscoe. 1984. "A Theory of Communicating Sequential Process," *ACM*, 31(3):560–599.
3. Brooks, S. D. and A. W. Roscoe. 1985. "An Improved Failure Set Model for Communicating Processes," In *Seminar on Concurrency*, Springer-Verlag, pp. 281–305.
4. Cook, C. 1980. "Office Streamlining Using the ICN Model and Methodology," In *National Computer Conference.*
5. Hennessy, M. and R. Milner. 1985. "Algebraic Laws for Nondeterminism and Concurrency," *Journal of the ACM*, 32(1):137–161.
6. Hoare, C. A. R. 1985. *Communicating Sequential Processes*, Prentice-Hall.
7. Menascé, D., V. Almeida, and L. Dowdy. 1994. *Capacity Planning and Performance Modeling: from Mainframes to Client-Server Systems*, Prentice-Hall.
8. Milner, A. J. R. G. 1980. "A Calculus of Communicating Systems," In *Lecture Notes in Computer Science 92*, Springer-Verlag, pp. 281–305.
9. Milner, A. J. R. G. 1989. *Communication and Concurrency*, Prentice-Hall.
10. Milner, R. 1991. The Polyadic π-Calculus: A Tutorial. Technical Report, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh.
11. Milner, R., J. Parrow, and D. Walker. "A Calculus of Mobile Processes," Technical Report, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1989.

12. Nutt, G. J. 1990. *A Simulation System Architecture for Graphs Model,* Springer-Verlag, pp. 417–435.
13. Nutt, G. J. and C. A. Ellis. 1979. "Backtalk: An Office Environment Simulator," In *ICCI'79, Conference Record,* pp. 22.3.1–22.3.5.
14. Petri, C. A. 1962. *Kommunikation mit Automaten.* PhD thesis, Univ. Bonn, West Germany.
15. Reisig, W. 1985. *Petri Nets. An Introduction,* Volume 4 of *Monographs of Theoretical Computer Science,* Springer-Verlag.
16. Stirling, C. 1987. "Modal logics for Communicating Systems," *Theoretical Computer Science,* 49:311–347.

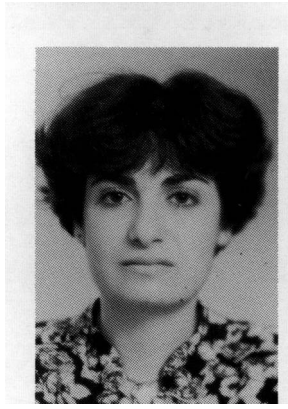
Biographies

Mourad Debbabi



Mourad Debbabi is a professor at the Computer Science Department of Laval University, Quebec, Canada, since 1994. From 1990 to 1994, he was a permanent researcher of the Bull Corporate Researcher Center, Paris, France. He obtained a Ph.D. in Computer Science from the University Paris XI-Orsay, France in 1994. He is the leader of the LSFM group (Languages, Semantics and Formal Methods) at Laval University. He is the author of many papers on the semantics of specification and programming languages and higher-order process algebras. His research interest include semantics, concurrency models, type theory, and formal specification and verification.

Nadia Tawbi



Nadia Tawbi is a professor at the Computer Science Department of Laval University, Quebec, Canada since 1996. She was a permanent researcher of the Bull Corporate Researcher Center, Paris, France, where she headed a research group on static analysis and formal verification for five years. She obtained a Ph.D. from the University Pierre et Marie Curie Paris VI in 1991. She is co-leader of the LSFM group. Her research interests include static analysis, formal verification, and parallelizing compiling.

Fouzia Bouguerch

Fouzia Bouguerch has been an M.Sc. student at the Computer Science Department of Laval University, Quebec, Canada, since 1994. She is a member of the LSFM research group. She is finishing a thesis on the qualitative analysis in workflow systems. She investigated the translation of workflow models into process calculi and their formal verification using tableaux-based methods. She is supervised by Pr M. Debbabi.