On Structured Workflow Modelling*

B. Kiepuszewski[1], A.H.M. ter Hofstede[2] and C. Bussler[3]

[1] *Mincom Limited*
GPO Box 1397, Brisbane Qld 4001, Australia
e-mail: bartek@mincom.com
[2] *Cooperative Information Systems, Queensland University of Technology*
GPO Box 2434, Brisbane Qld 4001, Australia
e-mail: arthur@icis.qut.edu.au
[3] *Netfish Technologies, Inc.*
2350 Mission College Blvd., Santa Clara, CA 95054, USA
e-mail: cbussler@netfish.com

Abstract

Recent years have seen the introduction of many commercial workflow management systems. While there are similarities between the languages of various of these systems, there are also significant differences. One particular area of differences is caused by the fact that different systems impose different syntactic restrictions. For example, some workflow management systems do not allow the use of arbitrary loops. In such cases, business analysts have to choose between either conforming to the language in their specifications or transforming these specifications afterwards. The latter option is preferable as this allows for a separation of concerns. In this paper we investigate to what extent such transformations are possible in the context of various syntactical restrictions (the most restrictive of which will be referred to as *structured workflows*). We also provide a deep insight into the consequences, particularly in terms of expressive power, of imposing such restrictions.

1 Introduction

Recent years have seen the proliferation of workflow management systems developed for different types of workflows and based on different paradigms (see e.g. [Aal96, EN93, EKR95, GHS95, Kou95, Law97, Sch96, DKTS98, Wor96]). Despite this interest in workflow management, both from academia and industry, there is still little consensus about its conceptual and formal foundations (see e.g. [JB96]).

While there are similarities between the languages of various commercially available workflow management systems, there are also many differences. However, it is often not clear whether these differences are fundamental in nature. For example, as different systems impose different

*This research is supported by an ARC SPIRT grant "Component System Architecture for an Open Distributed Enterprise Management System with Configurable Workflow Support" between QUT and Mincom.

syntactic restrictions, one may wonder whether this affects the expressive power of the resulting languages. In addition to that, such variations result in business analysts being confronted with the question as to whether to conform to the target language right away when they specify their workflows, or to transform their specifications in a later stage.

In this paper focus is on syntactic variations in workflow specification languages. Different workflow management systems impose different syntactical restrictions. The most restrictive types of workflows will be referred to as *structured workflows*. Systems such as SAP R/3 and Filenet Visual Workflo allow for the specification of structured workflows only. While enforcing restrictions may have certain benefits (e.g. verification and implementation become easier), the price that may have to be paid is that the resulting language is more difficult to use and has less expressive power.

In this paper, it will be shown that some syntactic restrictions will lead to a reduction of expressive power of the language involved, while other restrictions are of a less serious nature and can be overcome by the introduction of equivalence preserving transformation rules. It will be also shown that even though for certain workflow models it is possible to transform them to equivalent structured forms, the resulting models are less suitable than the original ones. Nevertheless, the automation of such rules could potentially lead to tools giving business analysts greater freedom in workflow specification without compromising their realisability in terms of commercially available (and preferred) workflow management systems.

The paper is organised as follows. In section 2, basic terminology and definitions are introduced. In section 3, the issue of workflow equivalence will be addressed. In section 4, a number of equivalence preserving transformations will be introduced and limitations to removing causes of "unstructuredness" will be investigated. Section 5 will focus specifically on restrictions on loops. Finally, in section 6 the conclusions are presented and some topics for further research identified.

2 Structured Workflows: Definitions

In this section the notion of a structured workflow is formally defined and some elementary properties stated. First however, we need to establish a working definition of the syntax of a workflow in general. Workflows as used in this paper will employ concepts used in most commercial workflow management systems.

The following definition provides the syntax of *arbitrary workflows* and restricts itself to control flow aspects.

Definition 2.1

Syntactically, a workflow $\mathcal{W}$ consists of a set of process elements $\mathcal{P}$, and a transition relation $\mathsf{Trans} \subseteq \mathcal{P} \times \mathcal{P}$ *between process elements. The set of process elements can be further divided into a set of or-joins O_j, a set of or-splits O_s, a set of and-joins A_j, a set of and-splits A_s, and a set of activities $\mathcal{A}$.*

The outgoing transitions of or-splits may have predicates assigned to them through a function $\mathsf{Pred}\colon \mathsf{Trans} \cap (O_s \times \mathcal{P}) \rightarrow \mathit{Predicate}$. *Activities may have a name assigned to them through the partial function* $\mathsf{Name}\colon \mathcal{A} \rightarrowtail \mathit{Name}$. *Activities without names are referred to* *as null activities. And-joins and or-joins should have an outdegree of at most one, and-splits and or-splits should have an indegree of at most one, and all activities have an indegree and outdegree of at most one. Finally, we will call process elements with an indegree of zero initial items ($\mathcal{I} \subseteq \mathcal{P}$) and conversely, process elements with an outdegree of zero - final items ($\mathcal{F} \subseteq \mathcal{P}$).*

□

Although the graphical notation used for representing workflows is irrelevant in terms of the results presented in this paper, we have to agree on one in order to provide examples to illustrate our arguments. Process elements will be represented by large circles; or-joins and or-splits will correspond to small, white circles, while and-joins and and-splits will correspond to small, shaded circles (the indegree and outdegree will always make it clear whether we are dealing with a join or a split).

There are many examples of languages that support the specification of arbitrary workflows, e.g. Staffware[1], Forte Conductor[2] and Verve WorkFlow[3].

Before we continue, the issue of semantics needs to be addressed, as the above definition focuses on syntax only. As the concepts used are well-known, we refer the reader to e.g. [HK99] where a semantics through a mapping to elementary Petri-nets is given. This mapping is straightforward and will not be heavily relied upon in the rest of this paper. To avoid possible ambiguities however, the following provides a summary of the essentials of the assumed semantics:

- All initial activities are started concurrently.

- Any activity that does not have an outgoing transition is a potential final task. The process does not automatically terminate when a final task is reached, rather, it terminates when “there’s nothing else to do”. Naturally, in case of only one concurrent execution thread, completion of a final task will result in termination of the process.

- Processes may deadlock (for example when an or-split is followed by an and-join). In this case the process will not terminate (sometimes the notions “successful termination” and “unsuccessful termination” are used in this context to distinguish between normal termination and deadlock).

- Or-splits are assumed to have the semantics of the exclusive or-split construct. This means that predicates are assigned in such a way that only one of them can evaluate to true, or, in case two or more of them evaluate to true, the workflow engine will ensure that only one path is actually taken. Note that the above assumption is without any loss of generality, as general or-splits can be modelled straightforwardly by a combination of and-splits and exclusive or-splits.

- In this paper we are not concerned with the particulars of the implementation of the data perspective. How predicates are evaluated is not relevant for the presented results.

[1] www.staffware.com
[2] www.forte.com
[3] www.verveinc.com

- An and-join will wait for all preceding activities to be finished before the subsequent activity can be started. An or-join will wait for any preceding activity to be finished before the subsequent activity will be started. The subsequent activity will be started for every preceding activity that finishes. This might be several times if one or several preceding activities finish several times.

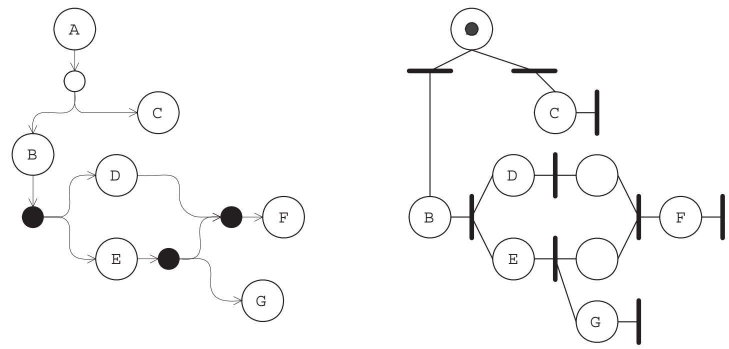

Figure 1: Petri Net mapping of the simple workflow specification

An example of the mapping of a simple workflow to a Petri-net is illustrated in figure 1. Note the need for the auxiliary non-labelled places for the representation of the and-join. It should be remarked that the actual details of the semantics of some constructs may vary slightly from product to product. We abstract from these variations since they do not compromise the presented results in any respect.

A structured workflow is a workflow that is syntactically restricted in a number of ways. Intuitively a structured workflow is a workflow where each or-split has a corresponding or-join and each and-split has a corresponding and-join. These restrictions will guarantee certain important properties shown later in the paper and in some cases correspond to restrictions imposed by commercial workflow management systems.

Definition 2.2

A structured workflow model (SWM) is inductively defined as follows.

1. *A workflow consisting of a single activity is a SWM. This activity is both* initial *and* final.
2. *Let X and Y be SWMs. The concatenation of these workflows, where the final activity of X has a transition to the initial activity of Y, then also is a SWM. The initial activity of this SWM is the initial activity of X and its final activity is the final activity of Y.*
3. *Let $X_1, \ldots, X_n$ be SWMs and let j be an or-join and s an or-split. The workflow with as initial activity s and final activity j and transitions between s and the initial activities of X_i, and between the final activities of X_i and j, is then also a SWM. Predicates can be assigned to the outgoing transitions of s. The initial activity of this SWM is s and its final activity is j.*

4. Let $X_1, \ldots, X_n$ be SWMs and let j be an and-join and s an and-split. The workflow with as initial activity s and final activity j and transitions between s and the initial activities of X_i, and between the final activities of X_i and j, is then also a SWM. The initial activity of this SWM is s and its final activity is j.

5. Let X and Y be SWMs and let j be an or-join and s an or-split. The workflow with as initial activity j and as final activity s and with transitions between j and the initial activity of X, between the final activity of X and s, between s and the initial activity of Y, and between the final activity of Y and j, is then also a SWM. The initial activity of this SWM is j and its final activity is s.

□

The above definition is illustrated in figure 2. Note that the last clause of the definition would correspond to a classic WHILE-loop if X is a null activity and to a classic REPEAT-UNTIL-loop if Y is a null activity. If $n = 2$, the second clause corresponds to a classic IF-THEN-ELSE.

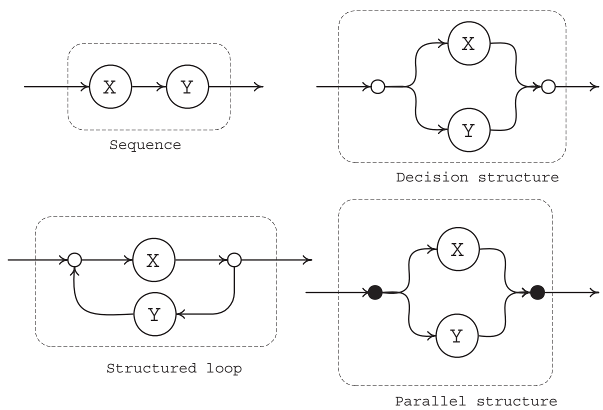

Figure 2: Illustration of structured workflow models

As is clear from definition 2.1, every structured workflow model is also an arbitrary workflow model. Hence, the semantics of the constructs used in structured models is the same as for arbitrary models. The reader may also note that every structured workflow model will always have one initial and one final task. All commercial WfMSs known to the authors allow for the specification of workflow models that are equivalent to structured models as defined in definition 2.2. Some of these WfMSs do not allow for the specification of arbitrary models though and they impose certain levels of structuredness by means of syntactical restrictions typically implemented in the graphical process designer.

The most restricted workflow modelling languages known to the authors with respect to imposing structuredness are the languages of FileNet's Visual WorkFlo[4] (VW) and SAP R/3 Workflow [KT98]. In both languages it is possible to design structured models only. These models resemble the definition provided earlier very closely with some minor exceptions such

[4] www.filenet.com

as that in VW the loops can only be of the form “WHILE p DO X”. In SAP R/3 Workflow no loops are allowed to be modelled in a direct way. An example of syntactical restrictions in the more general area of data and process modelling can be found in UML's activity diagrams (see e.g. [Fow97]) where business modellers are forced to exclusively specify structured models.

The definition of SWMs guarantees these types of workflows to have certain properties. Specifically, by the use of structural induction it can easily be shown that SWMs do not deadlock (see [HK99]). In addition to that, in SWMs it is not possible to have multiple instances of the same activity active at the same time. This situation is easily modelled in an arbitrary workflow if an and-split is followed by an or-join construct. Similarly, an arbitrary workflow will deadlock if an or-split is followed by an and-join.

Since in the following sections we will regularly pay attention to arbitrary workflow models that do not deadlock and do not result in multiple instances, for terminological convenience we introduce the notion of *well-behaved* workflows.

Definition 2.3

A workflow model is well-behaved *if it can never lead to deadlock nor can it result in multiple active instances of the same activity.* □

Corollary 2.1 Every structured workflow model is well-behaved.

Instead of requiring workflows to be structured, it is more common for workflow languages to impose restrictions on loops only. For example IBM MQSeries/Workflow[5] and InConcert [InC98] do not allow the explicit modelling of loops. Instead they have to be modelled by the use of decomposition. This is equivalent to using a “REPEAT X UNTIL p” loop. In case of MQSeries/Workflow, predicate p is specified as the *Exit Condition* of the decomposition.

Hence, in between arbitrary workflow models and structured workflow models, we recognise a third class of workflow models, referred to as *restricted loop models.*

Definition 2.4

A restricted loop workflow model (RLWFM) is inductively defined as follows:

1. *An arbitrary workflow model without cycles is an RLWFM.*
2. *Let X and Y be RLWFMs with each one initial and one final node. Let j be an or-join and s an or-split. The workflow with as initial activity j and as final activity s and with transitions between j and the initial activity of X, between the final activity of X and s, between s and the initial activity of Y, and between the final activity of Y and j, is then also a RLWFM.*

□

Note that languages that support loops through decomposition are a subset of the class defined by the above definition (in those cases, essentially, Y corresponds to the empty workflow). Naturally, every SWF is an RLWFM and every RLWFM is an arbitrary workflow model.

Example 2.1 Figure 3 illustrates the three different classes of workflows. □

[5] www.ibm.com/software

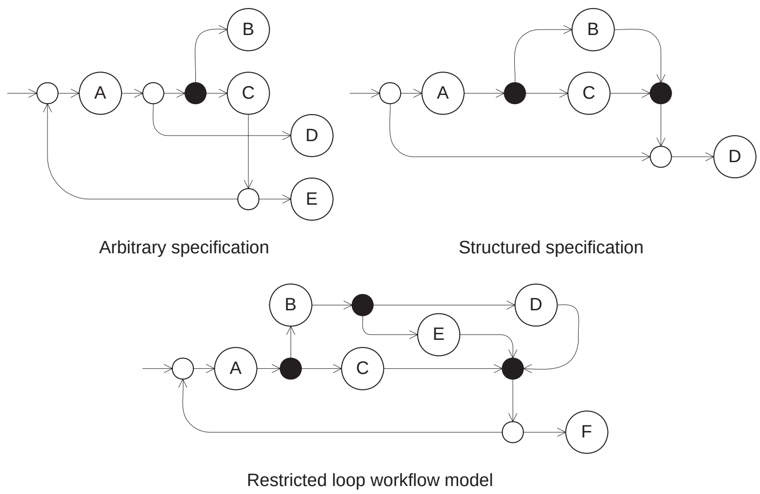

Figure 3: Three different workflow model classes

3 Equivalence in the Context of Control Flow

As there exist workflow languages that do not allow for the specification of arbitrary workflows, business analysts are confronted with the option to either restrict their specifications such that they conform to the tool that is used or specify their workflows freely and transform them to the required language in a later stage. From the point of view of separation of concerns, the latter option is preferable. To support such a way of working it would be best to have a set of transformations that could be applied to a workflow specification in order to transform it to a structured workflow in the sense of the previous section. Naturally, these transformations should not alter the semantics of the workflows to which they are applied, they should be *equivalence preserving*. However, this immediately raises the question as to what notion of process equivalence is the most applicable in the context of workflows (for an overview of different notions of process equivalence the reader is referred to [Gla90]).

One of the most commonly used equivalence notions is that of bisimulation (see e.g. [Mil89]). The formal definition of bisimulation between two different workflow systems, given the fact that they would most likely use different syntax and semantics, would have to be defined using some common formalism that can be applied to both systems. One of the most convenient ways to do it is to define bisimulation formally in terms of their Petri-net representation. That immediately leads to the conclusion that *weak bisimulation* has to be considered since Petri-net representations of workflow models may use many, internal, non-labelled places.

In the context of workflow processes with parallelism, the notion of basic weak bisimulation is not strong enough. Bisimulation is defined in terms of execution sequences, i.e. in terms of arbitrary interleaving. As such, however, bisimulation cannot distinguish between a concurrent system and its sequential simulation. For that reason a stronger equivalence notion is needed. Such a notion is provided in [BDKP91] where it is referred to as *fully concurrent bisimulation*. Given the fact that the formal definition is relatively complex and the details are not particularly useful for the purpose of this paper, we will present fully concurrent bisimulation in the context of workflow specification in terms of the *bisimulation game* (adapted from [Jan94]):

1. There are two players, Player A and Player B, each of which having a workflow model specification (Workflow A and Workflow B).
2. Player A starts the initial activities in his workflow model specification. Player B responds by starting the initial activities in his workflow model specification (which should exactly correspond to those of player A).
3. Player A may choose to finish any of its activities and start a corresponding subsequent activity. Player B responds accordingly by finishing and starting an activity with the same label (possibly performing some internal, non-labeled, steps first).
4. If Player B cannot imitate the move of Player A, he looses. By imitating we mean that at any point in time the same set of activities in workflow B should be completed and started as in workflow A. Player B wins if he can terminate his workflow once Player A has terminated his workflow. Similarly Player B wins if he can deadlock his workflow once Player A has deadlocked his workflow. The case of an infinite run of the game is considered to be successful for Player B too.

If there is a strategy for defending player (Player B) to always prevent Player A from winning then we say that workflow B can simulate workflow A. If the reverse applies as well (workflow A can simulate workflow B) then we consider the two workflow specifications to be equivalent.

The following figure contains examples of several equivalent and non-equivalent workflow specifications.

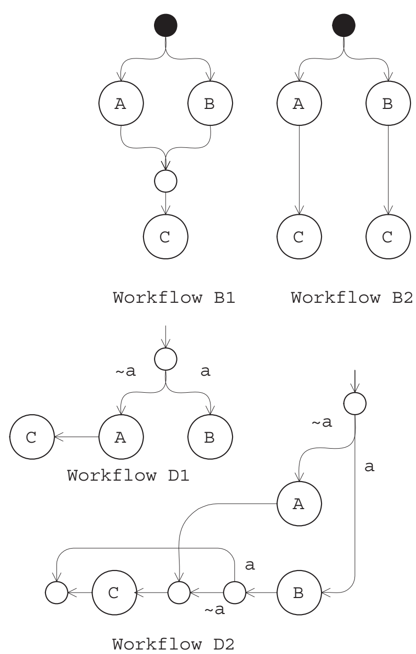

Figure 4: Workflow equivalence examples

1. Workflows $A1$ and $A2$ are not equivalent. Note that after completing activity A in workflow $A1$ there is still a choice to be made whether to proceed with activity B or with

activity C. In workflow $A2$ this option is not present anymore once activity A is completed.

2. Workflows $B1$ and $B2$ are equivalent.

3. Workflows $C1$ and $C2$ are not equivalent. In workflow $C1$ activities B and C can be performed concurrently, while in workflow $C2$ they cannot. Note that these two workflows are equivalent according to the traditional notion of bisimulation (just not in fully concurrent bisimulation).

4. Workflows $D1$ and $D2$ are equivalent provided that activities A and B do not affect the value of α.

4 Transformation of Arbitrary Workflow Models to SWMs

In this section transformations from arbitrary workflow models to SWMs are studied and to what extent such transformations are possible. All transformations presented in this section assume that the workflow patterns they operate on do not contain data dependencies between decisions, in other words for all intents and purposes all decisions can be treated as nondeterministic. This assumption allows us to assume that all possible executions permitted by the control flow specification are possible.

The organisation is as follows. First we concentrate on workflows that do not contain parallelism (to be more precise, we consider workflows that do not contain and-join and and-split constructs). Then we will concentrate on workflows that do contain parallelism, but do not have any cycles. Then we will consider workflow models with both loops and parallelism. Finally, we will comment on the suitability of the presented transformations.

4.1 Simple workflows without parallelism

Workflows that do not contain parallelism are simple models indeed. Their semantics is very similar to elementary flow charts that are commonly used for procedural program specification. The or-split corresponds to selection (if-then-else statement) while the activity corresponds to an instruction in the flow chart. It is well known that any unstructured flow chart can be transformed to a structured one. In this section we will revisit these transformation techniques and present and analyse them in the context of workflow models.

Following [Wil77] we will say that the process of *reducing* a workflow model consists of replacing each occurrence of a base model (i.e. one of the four shown in figure 2) within the workflow model by a single activity box. This is repeated until no further replacement is possible. A process that can be reduced to a single activity box represents a structured workflow model. Each transformation of an irreducible workflow model should allow us to reduce the model further and in effect reduce the number of activities in the model.

The strong similarity of simple workflow models and flow diagrams suggests that if we do not consider parallelism, there are only four basic causes of unstructuredness (see e.g. [Wil77, Oul82]):

- Entry into a decision structure
- Exit from a decision structure
- Entry into a loop structure
- Exit from a loop structure

Entry to any structure is modelled in a workflow environment by an or-join construct. Similarly, an exit is modelled by an or-split. Once parallelism is introduced we will also consider synchronised entry and parallel exit modelled by and-join and and-split constructs respectively.

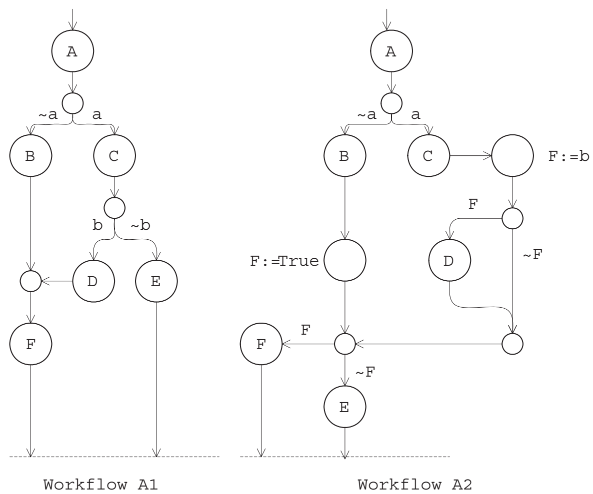

Figure 5: Exit from a decision structure

The first transformation (all transformations in this section are based on [Oul82]), depicted in figure 5, can be performed when transforming a diagram containing an exit from a decision structure. It is important to observe that variable Φ is needed since activity D can potentially change the value of β or, if β is a complex expression, it could change the value of one of its components. This transformation is achieved through the use of auxiliary variables. It should be noted that the models in figure 5 (and in all the following figures) are intended to be fragments of workflows, rather than complete workflows in themselves. The reader should verify for themselves that both models in this figure are indeed fully concurrent bisimulation equivalent.

The transformations as depicted in figure 6 are used when a workflow model contains an entry to a decision structure. Here workflow $B2$ is a transformation of $B1$ achieved through *node duplication*, whereas workflow $B3$ is a transformation of $B1$ achieved through the use of auxiliary variables. Note that as we do not consider parallelism in this subsection, activities A and E can not run concurrently (so there must be an or-split preceding this partial workflow model).

The following two diagrams, depicted in figures 7 and 8, capture transformations to be used when a model contains an entry to, or an exit from a loop structure, respectively.

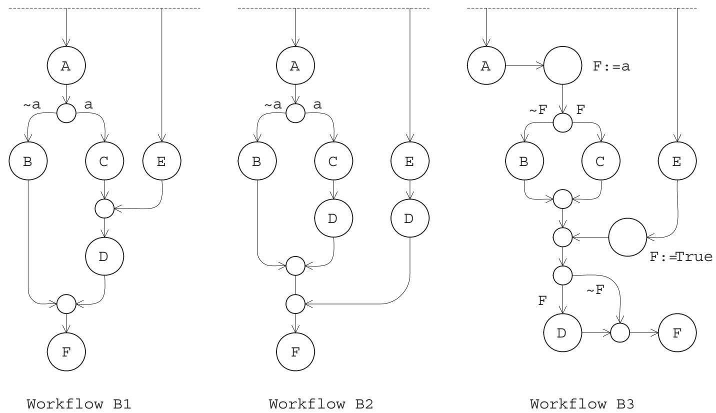

Figure 6: Entry into a decision structure

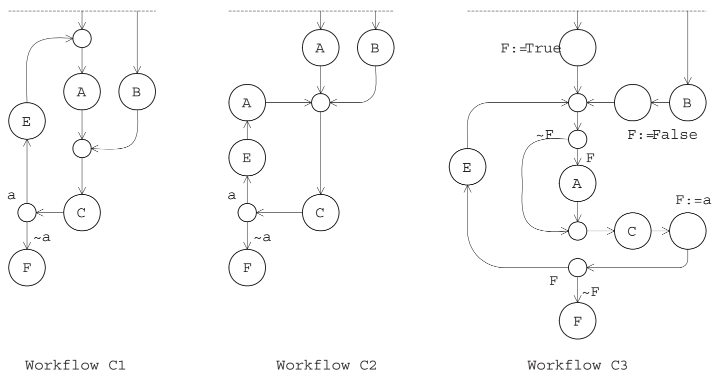

Figure 7: Entry into a loop structure

Repeated application of the transformations discussed in this section can remove all forms of unstructuredness from a workflow. All unstructured workflows without parallelism have an

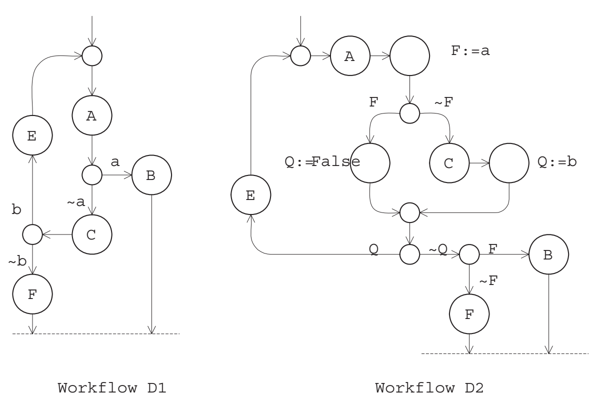

Figure 8: Exit from a loop structure

equivalent structured form. Finally, it should be remarked that in some cases we have presented alternative transformations (not using auxiliary variables) and in some cases we have not. In later sections, we will show that this has a reason: in the cases where no extra transformations (not using auxiliary variables) are presented, such transformations turn out not to exist.

4.2 Workflows with parallelism but without loops

Addition of parallelism immediately introduces problems related to deadlock and multiple instances. As noted in section 2, structured workflow models never result in deadlock nor multiple instances of the same activity at the same time. Hence the following lemma is obvious.

Lemma 4.1 Structured workflow models are less expressive than arbitrary workflow models.

This lemma immediately raises the question as to whether well-behaved workflow models can be transformed to structured workflow models. As the next theorem shows, the answer to this question is negative.

Theorem 4.1 There are arbitrary, well-behaved, workflow models that cannot be modelled as structured workflow models.

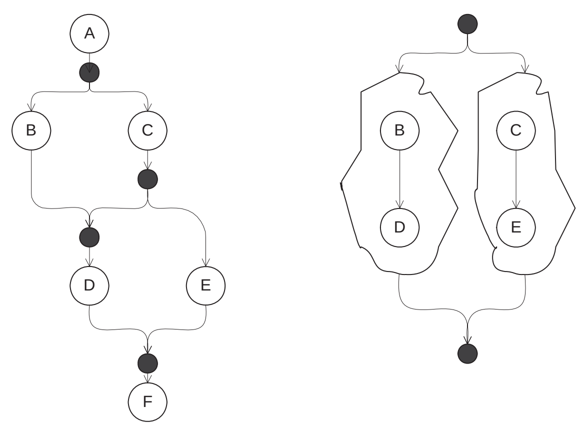

Figure 9: Arbitrary workflow and illustration of its essential causal dependencies

Proof:

Consider the workflow fragment in figure 9. The first observation is that as activities B and C are causally independent (that is, they can be executed concurrently) they have to be in different branches of some parallel structure in a corresponding structured workflow. As activities C and E are causally dependent (E is always performed after C) there must be a path from C to some activity named E. This activity has to be in the same branch as C as it cannot be outside the parallel structure as that would make it causally dependent on B. By applying similar reasoning, an activity named D has to be in the same branch of a parallel structure as B. Now we have that as C and D are in different branches of a parallel structure they are causally independent. However, in the original model they are causally dependent. Contradiction. No corresponding structured workflow exists. □

To find out which workflow models can be effectively transformed into SWMs, let us concentrate on the causes of unstructuredness that can occur when parallelism is added. If loops are not taken into account, these causes are:

- Entry to a decision structure
- Exit from a decision structure
- Entry to a parallel structure
- Exit from a parallel structure
- Synchronised entry to a decision structure
- Parallel exit from a decision structure
- Synchronised entry to a parallel structure
- Parallel exit from a parallel structure

In the remainder of this section we will concentrate on which of these structures can be transformed to a structure model.

Entries and exits from decision structures are dealt with in section 4.1 and can obviously be transformed to a structured model.

As a synchronised entry to a decision structure and an exit from a parallel structure leads to a potential deadlock (i.e. there are instances of the model that will deadlock), it follows that if the original workflow contains any of these patterns, it cannot be transformed into a SWM.

Parallel exits and synchronised entries to a parallel structure are dealt with in theorem 4.1. The reasoning of this theorem can be applied to any model that contains these patterns. Hence such models, even though they may be well-behaved, cannot be transformed into SWMs.

Before analysing the two remaining structures let us define a syntactical structure called an *overlapping structure*. This structure has been previously introduced in the context of workflow reduction for verification purposes in [SO99]. A specific instance of it is shown in figure 10. An overlapping structure consists of an or-split followed by i instances of and-splits, followed by j instances of or-joins and finally by an and-join. The structure of figure 10 has both i and j degrees equal to two. The overlapping structure contains both an entry to a parallel structure and a parallel exit from a decision structure and it never results in a deadlock. It is possible to transform an overlapping structure into a SWM as shown in figure 10.

A thorough analysis of the causes of deadlock and multiple instances in workflow models (see e.g. [SO99]) leads to the conclusion that workflow models containing a parallel exit from a decision or an entry to a parallel structure will cause a potential deadlock unless they form a part of an overlapping structure or the exit path from the decision does not join the main execution path.

Hence we conclude:

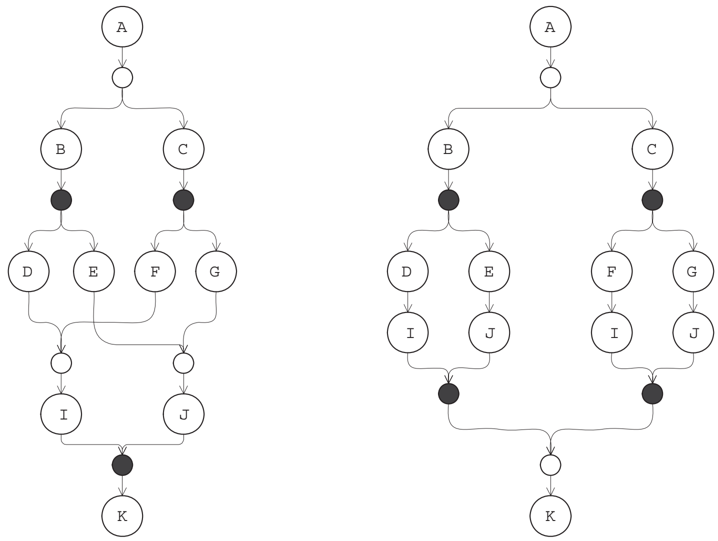

Figure 10: Overlapping structure

- An entry to a parallel structure can cause a potential deadlock unless it is part of an overlapping structure (in which case it can be transformed as shown).
- Similarly, a parallel exit from a decision structure can cause a potential deadlock and cannot be transformed into a SWM unless it is part of an overlapping structure or if the exit path does not join the main path (figure 11 illustrates the second case and the corresponding transformation).

Table 1 gives an overview of the main results of this section.

The observations in this section have led us to the following conjecture:

Conjecture 4.1 Any arbitrary well-behaved workflow model that

1. does not have loops,
2. when reduced, does not have a parallel exit from a parallel structure, and
3. when reduced, does not have a synchronised entry into a parallel structure,

can be translated to a SWM.

4.3 Workflows with parallelism and loops

Finding out whether a workflow can deadlock or not in the context of loops is much more complex and conjecture 4.1 cannot be automatically applied. To expose potential difficulties let us concentrate on what kind of loops we can encounter in a workflow model once and-join and and-split constructs are used. Every cycle in a graph has an entry point that can be either an or-join or an and-join and an exit point that can be either an and-split or an or-split. Cycles

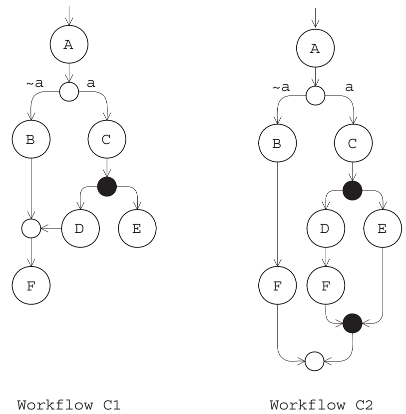

Figure 11: Exit path not joining main path in parallel exit from decision structure

without an entry point cannot start and cycles without an exit point cannot terminate. The latter case can be represented by a cycle with an exit point where the exit condition on the or-split is set to false.

Most cycles will have an or-joins and or-splits as entry and exit points respectively (note that there may be many exit and entry points in the cycle) provided that the workflow is well-behaved. The transformation of such cycles is straightforward using transformations as presented earlier in this section.

If the cycle has an and-join as an entry point, the workflow will most likely deadlock. Examples of two workflows containing cycles with and-join as an entry-point that do not deadlock are shown in figure 12.

Conversely, most workflows that have an and-split as an exit point will most likely result in multiple instances. Our previous observation that any workflow resulting in deadlock or multiple instances cannot be modelled as a structured workflow certainly holds whether or not the workflow has loops. The major impact of introducing loops though is that finding out if the workflow deadlocks or results in multiple instances becomes a non-trivial task (see [HOR98, HO99]).

pattern	*transformation possibility*	*comments*
Entry to par. struct.	transf. not always possible	only if overlapping struct.
Exit from par. struct.	no transf. possible	deadlock
Synchr. entry to a decision	no transf. possible	deadlock
Par. exit from a decision	transf. not always possible	overlapping struct. or paths do not merge
Synchr. entry into a par. struct.	no transf. possible	
Par. exit from a par. struct.	no transf. possible	
Entry to decision	transf. possible	
Exit from decision	transf. possible	

Table 1: Summary of results

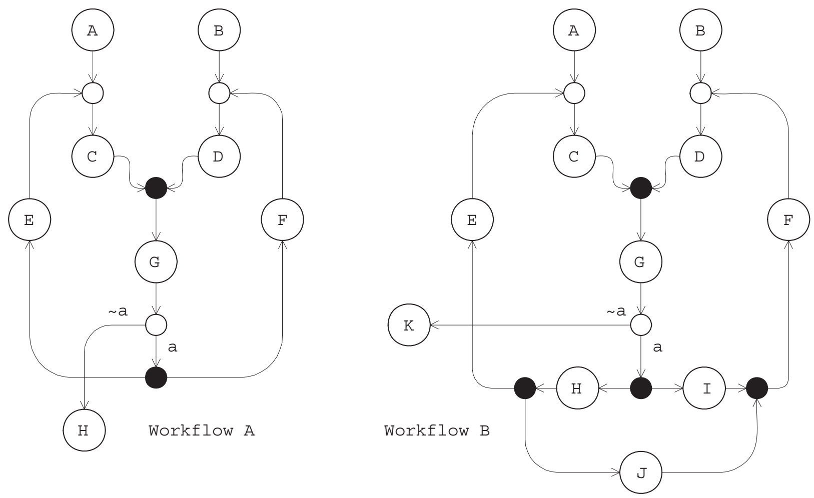

Figure 12: Two workflow models with arbitrary loops

In rare cases when a cycle has an and-join as entry and an and-split as exit point and the workflow involved does not deadlock nor result in multiple instances, theorem 4.1 is helpful when determining if such a workflow can be remodelled as a structured workflow. In figure 12 for example, workflow A can be remodelled as a structured workflow whereas workflow B cannot. The equivalent workflow to workflow A is shown in figure 13.

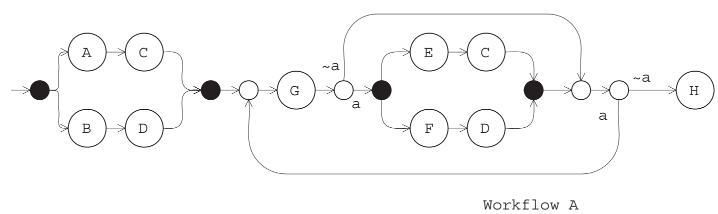

Figure 13: Structured version of leftmost workflow of figure 12

4.4 Suitability of transformations

The transformations presented earlier in this section are using two major techniques: 1) node duplication and 2) use of auxiliary variables to control conditions. In this section we will comment on the suitability of these solutions.

Suitability in general refers to the relation between concepts offered in the specification technique and concepts required by the problem domain. There are a number of aspects in a workflow specification, e.g. data and control flow, and there are a number of ways in which the same underlying model can be presented, e.g. data flow and control flow "view". Yet, conceptual models, in general, are required to convey a certain amount of information which should not be split up, if the model is to be effective (this corresponds to the *Cognitive Sufficiency Principle* promulgated by [BH98]). For example we believe that the model that conveys all control flow interdependencies between activities in a control view is a better model than the model that requires both the control flow view and data flow view to understand relationships between activities. Consider for example the three models from figure 6. In models $B1$ and $B2$ it is clear that activities B and D are exclusive in the sense that they will never be both executed in any process instance. On the other hand, in model $B3$, it seems that activity D can follow the execution of activity B. Only close inspection of the or-splits' predicates as well as implicit knowledge that activity B does not change the value of variable Φ can lead to the conclusion that activities B and D are indeed exclusive.

To retain the suitability of a certain workflow model, transformations should avoid using auxiliary variables to control or-splits through predicates. Unfortunately, this is not always possible.

Theorem 4.2 There are forms of unstructuredness that cannot be transformed without the use of auxiliary variables.

Proof:

Consider the workflow model of figure 8. This workflow model contains multiple exits from a loop and as such is unstructured. Now consider another workflow model equivalent to this model, which is structured. The first observation is that as workflow representations are finite, this structured workflow model needs to contain at least one loop as the associated language is infinite. On one such loop there has to be an occurrence of both activities A and C. Activities B and F should be outside any loop (as we cannot use predicates anymore to prevent paths containing these activities to be chosen if they are included in the body of the loop). Playing the bisimulation game yields that after each instance of activity A one should be able to choose to perform either C or B. Since B is outside any loop, there has to be an exit point from the loop sometime after activity A (but before activity C, as one cannot use predicates that guarantee that activity C will be skipped after the decision has been made to exit the loop). Similarly, after each instance of activity C one should be able to choose to perform either activity E or activity F. As F is outside any loop, we also have an exit point from this loop after activity C (but before activity E). Hence, the loop under consideration has at least two exit points and the workflow cannot be structured. Contradiction. Hence a structured workflow equivalent, not using auxiliary variables, to the workflow of figure 8 does not exist. □

An alternative technique to transform arbitrary models into a structured form requires node duplication. As has been proved earlier, it cannot be used for every model, but even when it can be used, it is not without associated problems. Consider once again the model in figure 6. If activity D in the left model is followed by a large workflow specification, the transformation presented in the right model would need to duplicate the whole workflow specification following activity D. The resulting workflow will be almost twice as big as the original and will therefore be more difficult to comprehend.

5 Restricted Loops

In this section we will focus on languages that impose restrictions on loops only. Examples of such languages are MQSeries/Workflow and InConcert. The main reason these languages impose restrictions on loops is that the introduction of cycles in their workflow specifications would result in an immediate deadlock because of their evaluation strategy. MQSeries/Workflow for example propagates true and false tokens and its synchronizing or-join expects tokens from every incoming branch before execution can resume; this results in deadlock if one of these branches is dependent on execution of the or-join itself. Note that the semantics of the synchronising or-join is different from the semantics of the or-join as presented earlier in this paper, but that does not compromise the obtained results.

The approach chosen in MQSeries/Workflow and InConcert guarantees that their specifications are well-behaved (for MQSeries/Workflow this is formally proven in [HK99]).

Even though one may ask the question whether any arbitrary workflow specification can be translated to a specification that uses restricted loops only, the more practical question would be to ask whether any well-behaved arbitrary specification can be translated to a specification using restricted loops only. As the next theorem shows, the answer to this question is negative.

Theorem 5.1 There are well-behaved arbitrary workflow specifications that cannot be expressed as RLWFMs.

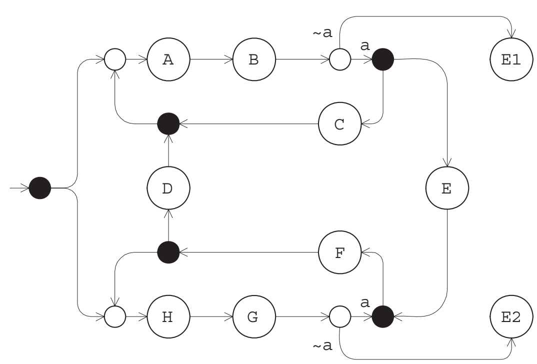

Figure 14: Well-behaved arbitrary workflow

Proof:

By showing that the workflow from figure 14 cannot be modelled as an RLWFM.

Observe that after completion of the initial activity and as long as α evaluates to true, there will be at least two tokens in the corresponding Petri-net. That means that in an equivalent workflow specification that has restricted loops only, there have to be two concurrent restricted loops running in parallel (if there was only one loop, the moment the exit condition was evaluated there would be only one token in the corresponding Petri-net).

One of the restricted loops would have to contain activities A, B, C, and E, and the other loop would have to contain activities D, F, G, and H. In the original workflow specification A is causally dependent on D. That means that there must be a path between A and D but that is impossible if A belongs to a different restricted loop than D according to the definition of a restricted loop. □

The careful reader may have noticed that in the workflow model of figure 14 data is used to make sure that both loops are exited at the same time (otherwise deadlock would occur). It is an open question as to whether there exist well-behaved arbitrary workflow specifications that do not contain decision dependencies and that can not be transformed into an RLWFM.

6 Conclusions

The transformation of arbitrary workflow models to workflows in a structured form is a necessity typically faced by either an application programmer who has to implement a non-structured workflow specification in an environment supporting structured specifications only (e.g. SAP R/3 workflow or Filenet Visual Workflo), or by a business analyst who is trying to capture real-world requirements in a structured workflow specification technique (e.g. UML's activity diagrams).

In this paper we have shown that even simple transformations require the use of auxiliary variables which results in the introduction of dependencies between decisions in a workflow graph. As a result the transformed workflow specification is typically more difficult to understand for end-users. Moreover, some arbitrary specifications cannot be transformed at all to a structured form. Hence in general, structured models are less expressive and less suitable than arbitrary models.

For these reasons it is our contention that any high-end workflow management system should support the execution of arbitrary workflow specifications. To some, this might seem to contrast with the common consensus of avoiding GOTO statements (and using WHILE loops instead) in procedural programming languages, but, as shown throughout this paper, the presence of parallelism as well as the nature of workflow specifications provide the essential difference.

As a consequence, the good workflow modelling environment should be supported by a powerful verification engine that would help process modellers detect syntactical problems such as potential deadlock or unwanted multiple instances. Using sophisticated verification tools for these purposes (incorporating techniques from state-of-the-art Petri-net theory) seems feasible from a practical perspective (see [AH98]).

References

[Aal96] 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, Cambridge, Massachusetts, November 1996.

[AH98] W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of Workflow Task Structures: A Petri-net-based Approach. Forschungsbericht Nr. 380, Universität Karlsruhe, Institut AIFB, Karlsruhe, Germany, 1998.

[BDKP91] E. Best, R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in Petri nets. *Acta Informatica*, 28:231–254, 1991.

[BH98] A.P. Barros and A.H.M. ter Hofstede. Towards the construction of workflow-suitable conceptual modelling techniques. *Information Systems Journal*, 8(4):313–337, October 1998.

[DKTS98] A. Doğaç, L. Kalinichenko, M. Tamer Özsu, and A. Sheth, editors. *Workflow Management Systems and Interoperability*, volume 164 of *NATO ASI Series F: Computer and Systems Sciences*. Springer, Berlin, Germany, 1998.

[EKR95] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock and C. Ellis, editors, *Conference on Organizational Computing Systems*, pages 10–21, Milpitas, California, August 1995. ACM SIGOIS, ACM.

[EN93] 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, Berlin, Germany, 1993. Springer-Verlag.

[Fow97] M. Fowler. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley, Reading, Massachusetts, 1997.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, April 1995.

[Gla90] R.J. van Glabbeek. The linear time-branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR'90. Theories of Concurrency: Unification and Extension*, pages 278–297, Berlin, Germany, 1990. Springer-Verlag.

[HK99] A.H.M. ter Hofstede and B. Kiepuszewski. Formal Analysis of Deadlock Behaviour in Workflows. Technical report, Queensland University of Technology/Mincom, Brisbane, Australia, April 1999. (submitted for publication).

[HO99] A.H.M. ter Hofstede and M.E. Orlowska. On the Complexity of Some Verification Problems in Process Control Specifications. *Computer Journal*, 42(5):349–359, 1999.

[HOR98] A.H.M. ter Hofstede, M.E. Orlowska, and J. Rajapakse. Verification Problems in Conceptual Workflow Specifications. *Data & Knowledge Engineering*, 24(3):239–256, January 1998.

[InC98] InConcert Inc. InConcert Process Designer's Guide, release 3.7, edition IC3.7.1. Cambridge, Massachusetts, 1998.

[Jan94] P. Jančar. Decidability Questions for Bismilarity of Petri Nets and Some Related Problems. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, editors, *STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 581–592, Caen, France, February 1994. Springer-Verlag.

[JB96] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, United Kingdom, 1996.

[Kou95] T.M. Koulopoulos. *The Workflow Imperative*. Van Nostrand Reinhold, New York, New York, 1995.

[KT98] G. Keller and Th. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley Longman, Harlow, United Kingdom, 1998.

[Law97] P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, New York, 1997.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[Oul82] G. Oulsnam. Unravelling Unstructured Programs. *Computer Journal*, 25(3):379–387, 1982.

[Sch96] T. Schäl. *Workflow Management for Process Organisations*, volume 1096 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany, 1996.

[SO99] W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In *Proceedings of the 11th Conf on Advanced Information Systems Engineering (CAiSE'99)*, pages 195–209, Hildeberg, Germany, June 1999.

[Wil77] M. H. Williams. Generating structured flow diagrams: the nature of unstructuredness. *Computer Journal*, 20(1):45–50, 1977.

[Wor96] Workflow Management Coalition. Terminology & Glossary. TC-1011, June 1996.