International Journal of Cooperative Information Systems
Vol. 12, No. 3 (2003) 365–391

World Scientific
www.worldscientific.com

THE CASE HANDLING CASE

H. A. REIJERS[*,‡], J. H. M. RIGTER[†,§] and W. M. P. VAN DER AALST[*,¶]

[*] *Faculty of Technology and Management, Eindhoven University of Technology*
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
[†] *Deloitte and Touche Management and ICT consultants*
P.O. Box 12002, NL-1100 AA, Amsterdam, The Netherlands
[‡] *h.a.reijers@tm.tue.nl*
[§] *jaaprigter@deloitte.nl*
[¶] *w.m.p.v.d.aalst@tm.tue.nl*

On the Dutch workflow market, a new and interesting paradigm named "case handling" is emerging. The goal of case handling is to overcome the limitations of existing workflow management systems. By using a data-driven approach combined with implicit routing and carefully avoiding context tunneling, awareness and flexibility are improved. Currently, many organizations are considering case handling systems such as FLOWer (Pallas Athena) rather than the more traditional workflow management systems. This paper provides a critical assessment of this development. The goal is to show the pro's and con's of case handling. Moreover, based on this assessment, an alternative approach using slightly extended workflow management systems is proposed. This approach is being pursued by the Dutch government in a project involving the workflow management system Staffware. Based on our experiences thus far, we provide guidelines for selecting the proper technology.

Keywords: Case handling; workflow management; exception handing; flexibility; case study; tool selection.

1. Introduction

A workflow management system (WfMS) is a software system that supports the specification, execution, and control of business processes.[23,27,29] The promise of a WfMS is fourfold:

- *Less coordination effort*: The WfMS liberates human actors from the routine work they need for coordination.
- *Higher quality*: The WfMS will offer to actors at *least* the work which is required to deliver the preferred quality of service.
- *Higher efficiency*: The WfMS will offer to actors at *most* the work which is required to produce an acceptable result.
- *Higher maintainability*: Ejecting the business control flow from traditional applications and moving it towards a WfMS simplifies the adjustment of both the logistics and the content of work.

On the basis of this promise, it is understandable that the expectations of WfMSs have been high since their appearance.[35,55] However, WfMSs have not attained the success of many other software packages[5] and their practical application has been limited to the support of simple and well-defined business processes.[3]

A fundamental question is whether the manufacturing paradigm that underlies WfMSs is suitable to be applied to an *information process*: a set of activities that create, process, manage, and provide *information*, which are carried out by actors, a mix of human workers and computer applications.[27] The manufacturing paradigm supports the idea that the way activities should be executed can be predefined in a clear, unambiguous and production-like manner.

Case handling systems have been presented as an explicit attack on the manufacturing paradigm.[45] The intention of a case handling system is to empower human actors in two ways. In the first place, it aims to limit the *context tunneling* effect, which may be encountered by actors when a WfMS only provides them the work which they are required to do and does not give them insight into completed and future activities. In the second place, a case handling system simplifies the handling of deviations from the regular process execution, which state-of-the-practice WfMSs do not support adequately.[2,14,15,33]

The promises of case handling system are enticing and their redemption is of obvious practical value. From a theoretical point of view, it is interesting to note that case handling systems exactly claim to offer what Agostini and De Michelis[5] think that current WfMSs lack to provide human actors:*awareness* of the situation and the *flexibility* to handle changes. The lack of flexibility especially to deal with unforeseen situations is a very widely felt shortcoming of many commercially available WfMSs, which has resulted in various research approaches to alleviate the problem.[2,5,11,14,15,19,22,28,32,39,48]

In this paper, we review the claim of case handling systems. We will argue that the specific solution that case handling offers may solve some of the deficiencies encountered with some of the commercially available WfMSs, but that there is a price to be paid: WfMS strengths are impaired or even annulled. As an alternative to the solution case handling systems offer, we propose the use of some concepts that are familiar in a research environment, but surprisingly absent in commercially available WfMSs. The proposal is inspired by the need for flexibility as expressed at the start-up of a *workflow implementation*[27] by the Dutch government. The Staffware corporation has agreed upon the extension of the WfMS Staffware in conformance with the proposal described in this paper.

The structure of this paper is as follows. In Sec. 2 we reflect upon related work. In the first half of Sec. 3, we describe the typical characteristics of case handling systems and their effect on elevating the supposed deficiencies of WfMSs. In the second half of Sec. 3, we focus on some of the undesired side-effects of using case handling systems. Section 4 contains the proposal we mentioned. This section builds upon a debt collection process as carried out by a Dutch government agency. In Sec. 5, we extend our view beyond WfMSs and case handling systems when

we discuss some of our insights on the selection of "workware". We present our concluding remarks in Sec. 6.

2. Related Work

2.1. *Manufacturing paradigm*

Around the time that commercially available WfMSs emerged, the debate intensified on the appropriateness of various design perspectives on human work in an organizational context.[9,52,54] In response to this controversy, Bowers *et al.*[13] indicated on the basis of their fieldwork that the constraints of WfMS process definitions on human work may be felt dearly, but can be acceptable with respect to the gains of solving managerial and administrative problems. Dourish *et al.*[19] noted that despite the problems associated with placing descriptions of processes outside their enactment, workflow technologies increasingly become popular. Nonetheless, their Contraflow and Freeflow prototypes are explicitly aimed at elevating the overly restrictive properties of WfMSs.

The similarities and differences between manufacturing processes on the one hand, with its generally high level of automation and great potential for formalizing business operations, and information processes on the other hand have been studied by e.g. Platier[46] and van der Aalst.[1] They both establish that there are striking similarities, as well as subtle differences. Their general conclusion is that many manufacturing concepts seem to be applicable to the management and execution of information processes.

Recently, the claim that the manufacturing paradigm is unsuitable for information processes was revitalized by Miers and Hutton,[41] van Tol,[53] and various vendors of case handling systems.[45]

2.2. *Flexibility*

The flexibility issue of WfMSs is widely studied, as indicated in the introduction. It is strongly related to the notion of an *exception*, for which the flexibility of a WfMS is seen as a remedy. An exception can be seen as an occasional deviation of normal process behavior.[14,39]

Strong and Miller[50] have presented an influential view on exceptions within office settings. They claim that exceptions are a normal part of business processes. According to them, the view, particularly held by managers, that exceptions are random and uncommon occurrences is not tenable and not supported by research.[26,51] Some exceptions are caused by erroneous designs of automated systems and mistakes by human operators; both sorts can ultimately be fixed. Another category of exceptions is not. This is either because they are caused by typical political tensions within a company, or because their complete (automated) solving would be uneconomic.

Partly building on the above-mentioned perspectives, Kammer *et al.*[32] classify exceptions with respect to their impact on a process definition. They distinguish exceptions that can be tolerated or safely ignored and still produce a satisfactory result as *noise*. *Idiosyncratic exceptions* influence a specific process instance and require a change to its further processing. Finally, *evolutionary exceptions* require changes in the underlying process definition. Casati and Pozzi[14] take a slightly different angle in distinguishing between *unexpected* and *expected* exceptions: the first category is treated on the process instance level, the second on the process definition level. Luo *et al.*[39] provide a three-dimensional framework to analyze exceptions, evaluating whether an exception is *known*, *detectable*, and *resolvable*. van der Aalst and Basten[2] distinguish between changes on the instance and definition level as *ad hoc changes* and *evolutionary changes*.

An unexpected exception in the sense of Casati and Pozzi[14] is typically handled by halting the process instance and modifying the process at the definition or instance level, after which a consistent continuation is pursued.[5,48] Heinl *et al.*,[28] Kradolfer and Geppert[36] and van der Aalst and Basten[2] among others address how cases can be migrated from one variant/version of the process definition to another in such a case. Another approach for unexpected exceptions is to tolerate an inconsistency and to "excuse" it before it can proceed.[15] Case-based reasoning and knowledge-based approaches are proposed among others by Luo *et al.*[39] and Klein and Dellarocas.[33]

An expected exception is typically countered by rolling back a process instance until a decision point has been reached from which forward execution can proceed along a different path than previously chosen.[14] Current research mainly focuses on these types of exceptions. Early work in this area[8,20,38] has triggered improvements to allow for more sophisticated workflow compensation and re-execution in a broader range of application domains.[31,42]

An exhaustive treatment of the various approaches mentioned so far is beyond the scope of this paper. A special issue of the CSCW journal contains some of the most interesting directions in current research on flexibility.[34] The editors distinguish in their introduction two main approaches that are currently taken by researchers in handling exceptions: "One track explores principled approaches for modifying 'normal' process definitions to handle exceptions as they occur." The earlier work of e.g. Cugola[15] and papers in this special issue by Ellis and Keddara[21] and Divitini and Simone[18] exemplify the first approach. The editors continue with: "The other takes the approach of utilizing process definitions that are less prescriptive in the first place." Examples of the latter approach are e.g. by Dourish *et al.*[19] and within the same issue by Agostini and De Michelis[5] and Faustmann.[34] The Milano system that Agostini and De Michelis[2] also describe in this special issue incorporates the concept of jumps. This is the basis for one of the proposed extensions described in Sec. 4. Another paper in this issue by Kammer *et al.*[32] gives a wide overview of the various approaches that were only highlighted in this section.

It is interesting to note that despite the broadness of the research into WfMS flexibility few research results have made their way to commercially available WfMSs. Very few commercially available WfMSs provide support at all for handling exceptions, as observed by e.g. Cugola,[15] Casati and Pozzi,[14] Klein *et al.*[34] and van der Aalst and Basten.[2] The proposal that we discuss in Sec. 4 is illustrative for this phenomenon, as it combines existing research concepts and is yet an extension to the world's leading WfMS.

2.3. *Context tunneling*

Different authors have identified the need for actors in a cooperative process to refer to the history of cases by some form of representation.[44,47,56] Sometimes this context needs to be transparent, other times visible.[7] With respect to the danger of context tunneling Kueng[37] states in conclusion to an empirical study that "through the use of a workflow system, jobs become more structured and more routine. Additionally, individuals are forced to stay within given limits. Since a larger proportion of work is programmed, it becomes harder to exercise and integrate creativity and ingenuity." Kueng stresses the importance of organizational measures (e.g. job rotation, a redefinition of actor's roles, etc.) to counter-balance these effects.

2.4. *Case handling systems*

Case handling systems have been studied before, but not many research papers have been devoted to them yet. Miers and Hutton[41] provide a very positive evaluation of case handling technology, just like van Tol[53] and Bayens and Tönissen.[10] Their arguments are very similar to that of case handling system vendors, e.g. Pallas.[45] Deen[16] takes a more critical approach. He focuses on the supposedly simplified effort of modeling process definitions for case handling systems. He concludes that no richer model expressiveness or flexibility has been reached by them. Actors may more easily interpret the process definition of a case handling system, although this will only be of real value when these models have a high level of detail. A contribution of our paper is that we extend Deen's single point of evaluation to a complete review of the supposed, general advantages of case handling systems and the effect of their specific properties.

3. Case Handling

3.1. *Characteristics*

Few commercial case handling systems exist. Known case handling systems are FLOWer from Pallas Athena, ECHO from Digital, CMDT from ICL and Vectus from London Bridge Group. There are also other case handling tools which are used on top of existing WfMSs. For example, the Activity Manager developed by BPi Software can be used on top of WfMSs such as Staffware, COSA, FileNET's Panagon eProcess Services, and IBM MQSeries.

There is some consensus that the general features of case handling systems are unique.[3,43,53] In particular, despite the similar goal of managing and executing business processes, the properties of a case handling system (CHS) are seen as different from those of a WfMS. We distinguish the following three characteristic properties of a CHS:

- The *system's focus is on the case.*[3,10,53]
- The *process is data-driven.*[3]
- *Parts of the process definition are implicit.*[3,16,43]

We will subsequently address each of these properties and compare them to those of WfMSs. If characteristics of CHSs are discussed in some detail, the system FLOWer is used as a frame of reference. FLOWer is inspired by the ECHO (Electronic Case-Handling for Offices) system whose development started in 1986 within Philips and later moved to Digital. We will also indicate in the coming sections which shortcomings of WfMSs the properties of CHSs are aimed for to solve. At this point, it is important to state that these shortcomings are generalizing and may not apply to individual systems.

3.1.1. *System focus is on the case*

To clarify relevant notions within a system that supports the management and execution of information processes, van der Aalst and van Hee[4] propose the following dimensions: the *process*, the *case*, and the *resources*. Within a WfMS, the process dimension is usually dominant in the clarification of most notions. A WfMS consists of a *build-time* and a *run-time* part.[29] The build-time part concerns the specification of the process definition. The run-time part uses this definition to execute and manage the process. The dominance of the process dimension is best seen on the run-time part of the system. The workflow engine, the heart of the run-time part, provides lists of work items to the human actors.[25] The allocation of a work item to an actor usually does not depend on the specific case, but on the capability of the actor to execute the corresponding activity. As soon as an actor completes a work item, the engine establishes which new work items arise. It does this on the basis of the dependencies within the activities of the process definition[25] — which is again strongly focused on the process.

On the other hand, within a CHS the human actor at run time is not faced with this potentially fragmentary, activity-centered view. Instead, an *entire* case — complete with all the information that it embodies — is handed out to him. The actor can add or update information of the case, or just simply browse and view its properties. What is more, it is not necessary to wait for the CHS to make an allocation decision; the case is always accessible for the actor with the proper rights. Note that in various WfMSs and other systems that offer process support the awareness issue has been identified and addressed differently. Lotus Notes supports partially contextualization through the possibility of attaching a conversation to

a document. The WfMS LinkWorks offers functionality to conceptually integrate a process instance with the documents that are created during its execution. The WfMS Cosa provides a network displayer to actors, which shows the dynamic current state of the process and in particular the open work items. The research WfMS Milano can provide a complete overview of the historic activities with respect to a process instance, where actors can modify this view to their needs, e.g. by selecting what information is relevant.[6]

This specific property of a CHS is primarily aimed at providing human actors a better overview of each of the cases that is handled. In other words, it should avoid *context tunneling*. An actor, for example, can inspect information that has been established earlier by other actors. Also, it is possible to inspect which activities still need to be executed. The risk of loosing this overview is that human actors may become alienated from the case. The idea is that if actors see only a part of the larger picture, they cannot determine their contribution to the outcome. As a result, the quality of their work possibly drops, as well as their job satisfaction.[37]

Aside from this primary aim, a case-centered system provides simpler access to the information of the CHS on the status of cases. In this, it resembles Customer Relationship Management systems that try to collect all information on a client within one view. In a WfMS, the overall status of a case certainly present in the system but often more difficult to reveal to an actor.

3.1.2. *Process is data-driven*

A WfMS usually manages the *state of a case* or, alternatively, the *process state* as the progress with respect to the (partially) ordered activities in a process definition.[25] Within an event-oriented WfMS such as Staffware from Staffware Corporation, the state is the set of activities that have been completed at any moment. Within a state/event-oriented WfMS such as COSA from Thiel/Software Ley, a more fine-grained view on this state is possible. The state of a case is then the collection of milestones that have been reached, completed with the values of specific case data. In both types of systems, a strong distinction exists between *logistic information and content*.[4] The logistic information is managed by the WfMS. What is more, the WfMS has no knowledge about the information that has no logistical effect. For example, if the name of a client is not used for routing purposes, this data is exclusively stored in other applications. The Workflow Management Coalition (WfMC) makes a similar distinction, namely between *workflow control data* on the one hand and *application data* on the other.[25]

The state of a case within a CHS consists of the complete collection of information on the case, i.e. both workflow control data and application data. All this information is managed by the CHS. By doing this, the CHS is capable of determining on a very detailed level what has been established for each case. This property addresses the *context tunneling* danger, as all information about a case can be provided that an actor may be interested in and is authorized to see. It also enables

the definition of fine-grained relations between activities. Instead of merely using precedence relations, it is possible to specify on a data level what is sufficient for an activity to count as completed and which activities then still need to be performed. This facilitates a much less rigid execution of the process definition than is the case in a WfMS. In other words, it counters the *flexibility* problem of WfMSs. Note that this functionality is comparable to the possibilities of Freeflow,[19] which incorporates an activity level both user and system states can be used for specifying intricate activity dependencies.

3.1.3. *Implicit modeling*

The process definition in a WfMS contains a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as actors, associated IT applications and data, etc.[25] Alternative paths through a process definition are generally required to manage variations in the properties and demands of customers (e.g. with respect to the requested delivery speed of a product), several organizational policies (e.g. a handling policy of a claim may distinguish between a low and a high-risk procedure), and evolving external circumstances (e.g. the conditions on the money market may influence a financial decision). Typical for a process definition in a WfMS is that the designer has to specify what is *permitted* (e.g. Cosa, Staffware, MQSeries). Any routing which is not specified at build time will not be supported by the WfMS at run time. Deen[16] and van Tol[53] refer to this approach as *explicit modeling*.

In a CHS, the process definition is modeled in a different way. Similarly to the WfMS approach, various activities in the process definition are distinguished, but only a *preferred* or *normal* control flow between these activities is modeled. By default, at run time an actor is able to execute, redo, or bypass these activities, possibly diverging from the normal flow. The designer of the process definition can limit the possibilities of redoing or bypassing by assigning proper authorizations for the respective actions. The modeling mechanisms of the CHSs differ. ECHO uses the language CTDL (Case Type Definition Language) to specify the process definition.[41] FLOWer has a graphical design environment STUDIO, which supports "execute", "redo" and "skip" primitives.[45] At the activity level, the roles of actors are specified that can perform these specific primitives for each activity, aside from the standard execution authorization. If an actor "skips" an activity, a work item is offered to the actor that should perform the following activity. If an activity is redone, the process state returns to the state before its execution. The way that actors can decide for themselves which activities are appropriate resembles the characteristics of groupware and *ad hoc* workflow systems,[4] although it is not possible to define with a CHS a completely new way of working at run time. We will address these other types of technology in Sec. 5. Note that the redo and bypass actions resemble the backward and forward jumps of the Milano system,[5] although in a CHS they are defined on the activity level instead of on the process state.

The modeling approach of CHSs is called *implicit modeling*.[16,53] Typical for a process definition in a CHS is that the designer has to specify what is *not permitted*. In other words, actors will be allowed in principle to engage activities opportunistically, but they have to respect predefined *constraints*. This idea is similar to the principles underlying the Contraflow prototype which evolved into Freeflow.[19] Contraflow is the result of pursuing the specification of continuous activity relations rather than instantaneous ones. The system PROSYT[15] allows the use of *invariants* to distinguish between acceptable and unacceptable sets of process states. Invariants on a local level may be violated (e.g. for a document) as long as the overall invariants (for the process instance) are satisfied. On a related note, Bernstein[11] describes a system which offers a variable degree of support, including to have actors plan their actions freely while automated agents monitor the imposed constraints. Characteristic for FLOWer is that at run time a normal, predefined temporal ordering is suggested.

The principal advantage aimed for with implicit modeling is that it will be easier to develop a process definition (at build time) which in exceptional situations will conform to the wishes of an actor to select and perform the activities that conform with the local situation (at run time). This is to counter the problem which e.g. Heinl *et al.*[28] and Klein *et al.*[34] describe: the inclusion of all alternative paths blows up the explicit process definition. If a path is not included in the explicit process definition and an exception occurs which justifies this path, then the actor is not adequately supported. In other words, implicit modeling should support a better *flexibility* of the system. Note that this type of flexibility aims at expected exceptions (see Sec. 2). The CHS approach fits within one of the two main approaches currently being taken by adaptive workflow researchers: utilizing process definitions that are less prescriptive[34] (see Sec. 2).

Aside from the primary flexibility aim, process definitions that are implicitly modeled should be easier to interpret by human actors and easier to maintain by designers than explicitly modeled ones.

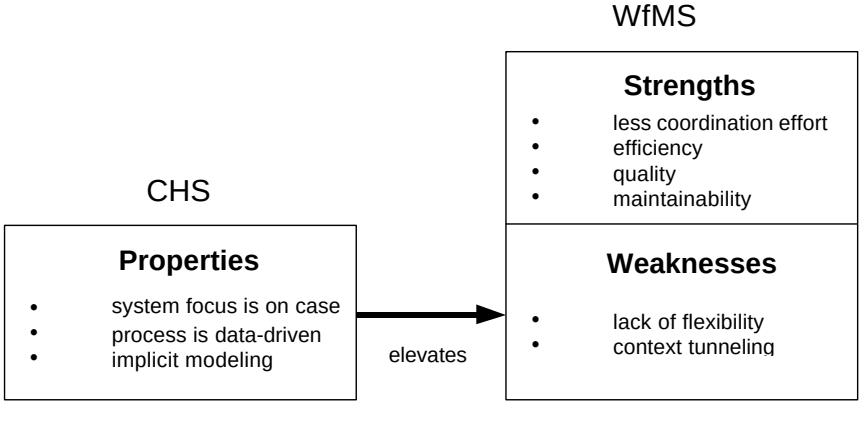

Fig. 1. Conceptual effects of CHSs.

The three discussed characteristic properties of CHSs are targeted to elevate the two typical weaknesses of commercially available WfMSs, as visualized in Fig. 1. Note that we also mentioned some secondary advantages of the use of a CHS in this section.

3.2. *Review*

In this section, we once again discuss the characteristic properties of a CHS, but try to show how they may conflict with the primary strengths usually attributed to WfMSs. To clarify and illustrate this review, we will try to focus on the essential and commonly shared characteristics of CHSs. For the sake of illustration, we will refer to the CHS FLOWer.[45]

3.2.1. *System focus is on the case*

As stated before (see Sec. 3.1), context tunneling is not a problem without existing solutions. Having said that, we are inclined to believe that the system focus of a CHS on the case will actually give an actor a good view on the history of the case, the information actually available, and the work that has to be done still. In other words, it will probably reduce the context tunneling effect. However, the only way for a CHS to do this is by maintaining *complete* data of the cases it handles. It is not sufficient to hold only a part of the case-related data. After all, sooner or later an actor will be interested in a piece of information (as part of the state of the case) that has been omitted from the control of the CHS.

This phenomenon places the people responsible for the role of the CHS in an overall organizational IT infrastructure in a difficult position. There are two principle choices they can make. The first option is to let the CHS maintain a rather large set of information. This set initially consists of exactly the same case information that is available in the information systems to which the CHS integrates and subsequently with all its additions and updates. In this scenario, there is a fully redundant administration of case data. The second option is to try and find a *selection* of all the available case data, which is then maintained by the CHS for its purposes.

Maintaining a redundant case data administration by the CHS is obviously a very inefficient and costly solution. The risk of inconsistencies also arises between data in the CHS and other systems, with obvious negative effects. The question may indeed be posed why other information systems are still maintained. It would make more sense to abandon the other information systems, and fully rely on the CHS for process execution and data storage. In the extreme case, this will lead to the situation where the CHS will be the sole, integrated system for supporting and executing the business process. However, this is exactly the situation that WfMSs were out to improve on: by a division between the logistics and the content of work, both parts would be easier to maintain.[4]

The other option, a selective storage of case data by the CHS, is troublesome for another reason. Provocatively stated: if it is difficult to think of all possible paths in a process definition an actor may want to follow in different situations, it must be similarly difficult to distinguish what is relevant state information in which an actor is interested in different situations. If we suppose that in a practical situation this distinction is possible, then we can ask ourselves why the same case state could not have been generated by a WfMS. After all, a WfMSs such as COSA and Staffware is equipped with case variables. These case variables could be used to present with a WfMS the same type of information that a CHS with a selective knowledge of the case can. In other words, the specific characteristic of the CHS then becomes non-distinctive.

3.2.2. *Process is data-driven*

It is necessary for implementing a data-driven process that all case information is available to the CHS. We already named this requirement for obtaining the specific system focus of a CHS (see previous subsection). The same questionable consequences in the fields of efficiency, quality, and maintainability therefore apply. There is yet an additional effect which has to do with the *concurrency* issue.

By having the state of a case driven by the actual values of the case data, it is very important to avoid inconsistent situations. Suppose that there are two actors with the same responsibilities who are working independently from each other on the same case. Because of various skipping and re-doing of activities and the access to the entire case data, it is possible that they both want to update the same piece of information, for example, the decision whether a loan application is granted. If they make a different decision, what is the new state of this specific case? Even if complete concurrency of the update actions is ruled out, it is clear that unwanted situations may arise if there is multiple access to the same data. One feasible way to prevent this is to support a system of *locking*: case data is completely locked by the first interested party and released only after an explicit log-off. This mechanism is, for example, implemented in FLOWer, which prohibits the simultaneous handling of a single case by different actors. Note that the Lotus notes system takes another approach in a similar situation. It allows the creation of different versions of the same piece of information. It is the responsibility of human actors to use the right one and possibly combine them at some time again to one.

The effect of a locking mechanism is that effectively all concurrency in handling the case is ruled out. At best, it is possible to exploit arbitrary execution orderings of activities, but simultaneous activities cannot be performed. Note that in a WfMS concurrent execution of activities is fully supported because of the strict control of data that is available to users at specific points.

The embargo on complete concurrency is a drawback, although perhaps not a serious one. van der Aalst and van Hee[4] do distinguish the possibility of concurrent work in an administrative environment as one of the prime methods to speed up

processing. Nonetheless, such a speed up is accomplished rather by the interleavings of activities that used to be sequentially ordered. This can be explained from the fact that the total response time of a case consists of less than 5% out of service time.[46] In other words, complete concurrent execution is, although not impossible, a rarity.

A straightforward patch for the concurrency problem in a CHS is that mutually exclusive sets of data may be distinguished, which can be concurrently processed by different actors. However, this undermines the unique system focus and case data availability of a CHS.

3.2.3. *Implicit modeling*

Deen[16] already established that the expressiveness of implicit modeling does not exceed that of explicit modeling. The same run-time flexibility can be offered by a WfMS on basis of an explicitly modeled process definition. It is not hard to show that the execute, skip, and redo primitives as provided by FLOWer can be expressed in modeling constructions that WfMSs support. For example, the COSA WfMS uses Petri nets[17,49] as modeling technique. In Fig. 2, four Petri net constructions are visualized respectively modeling an activity that can and should be executed (A), an activity which can be executed or skipped (B), an activity that can be executed and possibly redone (C), and an activity that can be executed, skipped or redone (D).

Therefore, the interesting question is not whether a CHS can offer *more* flexibility than a WfMS at run time — it cannot — but whether it can provide this flexibility *with less effort*. More in specific, we are interested to know the following:

(1) Whether it is easier for the workflow designer to use implicit modeling for the initial model and the subsequent updates (maintenance).
(2) Whether the end-user can interpret an implicit process definition more easily.

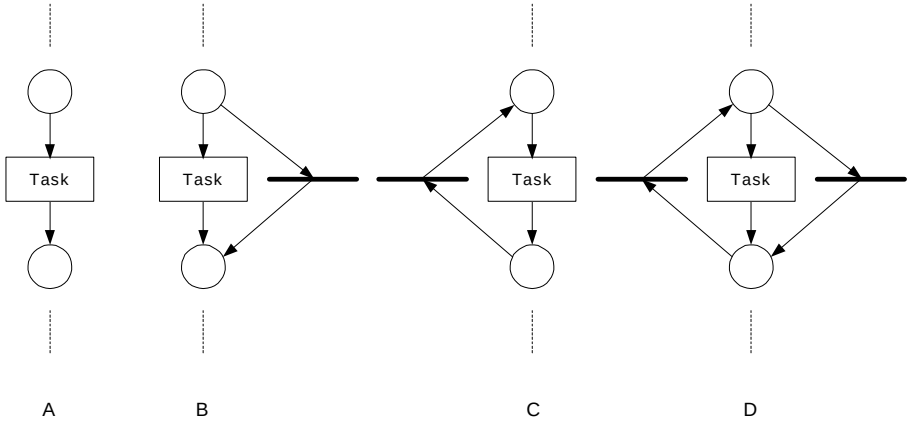

Fig. 2. CHS primitives expressed in Petri nets.

The answers to these questions largely seem to be a matter of taste. Empirical research is probably most suited to derive some objective support for answering these questions. Some comments are nonetheless due.

Deen[16] states that for modelers it is easier to use implicit modeling and for users it is easier to interpret these models. This advantage is more apparent in situations where very fine-grained activities are modeled. Miers and Hutton[41] observe that the development of a process definition is much faster due to the implicit modeling approach. They acknowledge, however, that modelers should have a deep understanding of the process at hand, as well as the used modeling language.

The primary reason why an implicitly modeled process definition should be easier to make, to maintain, and to interpret is that the process definition is *simpler*. More in specific, compared to a traditional process definition as used by COSA or Staffware, in a process definition of e.g. FLOWer there will be less dependencies between activities that obstruct the view.

Let us now consider the example in Fig. 3 to examine this supposed ease. In this figure, the left-hand model represents a CHS process definition. Alongside each activity, its activity status is shown, which is normally no part of the graphical depiction of the model. We are interested to know whether it is possible to jump back to the execution of activity A when activity C has been completed. To determine this for a FLOWer model, we have to know the redo statuses of activities A, B, and C. After all, the semantics of the FLOWer engine is such that all activities between

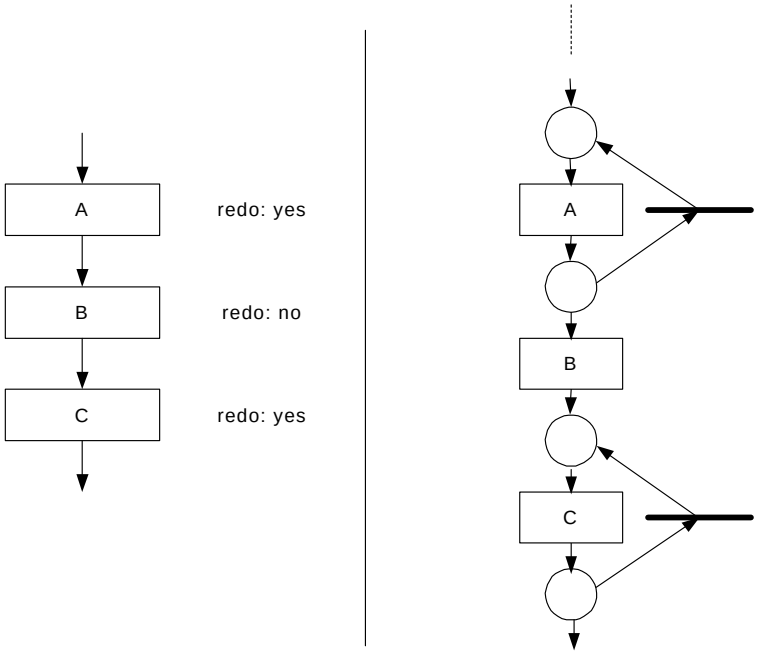

Fig. 3. Examples of a CHS and WfMS process definition.

A and C must be redo-able to turn back to A. If C could be redone but B cannot, then it is impossible to redo A once C is completed, regardless of the redo status of A itself. The explicitly modeled equivalent process definition is in the right-hand side of the figure, which could have been modeled in e.g. COSA. Note that this model is completely graphical.

Although we used a specific example, we believe that the simplicity of the CHS model is reached by omitting information which that is relevant to understand what is really happening. We conjecture therefore that possible paths through the process definition cannot be identified more easily in an implicit process definition. Rather, it is easier to distinguish a semi-formal notion of the *normal flow* of the process. By focusing exclusively on the normal flow, it is easily forgotten that there are more paths provided by the CHS than can be seen at first glance. Even an experienced modeler of implicit process definitions may find it hard to decide whether an *unwanted* sequence of activities may occur on basis of an arbitrary model. This is a quality aspect, which a WfMS typically tries to improve upon (see Sec. 1).

There are also two other effects worth mentioning, which arise when implicit process definitions are used which are only slightly prescriptive. This is, in other words, the situation where almost full possibilities to bypass and rollback on activities are maintained in the process definition. It is a natural tendency to build this type of models with a CHS, as implicit modeling comes down to modeling *what is not permitted*. The first effect is that actors will spend more time on run-time coordination of their activities than is the case in a WfMS. More important is the effect of the possibility to reiterate for as many times as an actor likes. There will be hardly any "points of no return". Although this appeals to the demand for more flexibility, it should be noted that this contradicts the idea of efficient process execution, as the CHS will not put any pressure on the actor for the completion of the process.

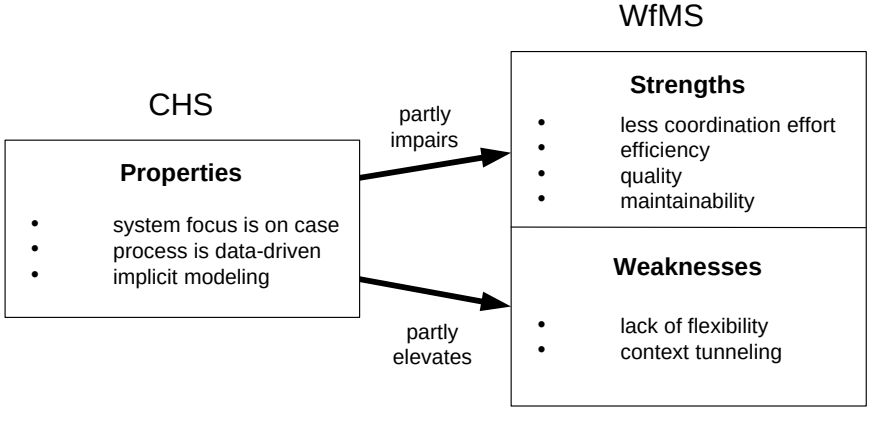

Fig. 4. Actual effects of CHSs.

We have named the quality and efficiency aspects among the primary promises of WfMSs in Sec. 1: precisely those activities are required which are due, no more and no less. Once again, in fixing the problems of a WfMS the characteristics of a CHS may be impairing the primary advantages of WfMSs.

In summary, we have argued in this section that the characteristic features of a CHS indeed will decrease the risk of context tunneling, but do not offer increased flexibility. Moreover, the specific nature of a CHS may threaten the primary strengths of WfMSs with respect to the decreased coordination effort, maintainability, efficiency, and quality. This is visualized in Fig. 4. A general superiority of CHSs over WfMS is therefore not tenable. Rather, the trade-off between the discussed effects should be made for each actual project to decide between the use of a WfMS and a CHS.

4. Yet Another View on Flexibility

4.1. *Handling addressable exceptions*

In spite of the abundance of views on the flexibility issue in relation to WfMSs, we present in this section yet another. However, our proposal combines *existing* views on exceptions, *existing* technology, and *existing* research concepts. The only innovative aspect is that this combination is unique and will actually be implemented in one of the world's leading WfMSs. The occasion that triggered this development is an actual application of workflow technology for a Dutch government agency with respect to debt collection. Within this agency, some 600 business professionals are purely involved with debt collection. Currently, this agency collects 10 million debts yearly, while this number is expected to grow to 20 million in 2004. In total, 5% of these cases requires human mediation: they are the exceptions on the normal procedure. Exceptions that occur generally have to do with changing insights into who is responsible for the debt and the changing legal status of the debtor(s) in question. The remaining 95% is fit for Straight-Through-Processing (STP), i.e. their handling can be completely automated.[40] The cycle time of an individual process instance can take up from four weeks to over a year. In light of the rapidly growing volume of work, the agency in question was looking for one integrated solution to coordinate both the bulk of standard work and the much smaller fraction of exceptions. Although the number of options to proceed when a typical debt collection exception occurs is not so extensive, the agency's management made very clear how important it is that a legitimate option is selected. In other words, careless selection of activities must be avoided because of the legal implications of mistakes. For both tracks, the additional integration with several governmental back-office systems was a very important requirement to ensure a legitimate and accountable process execution.

From the start of the project, the basic characteristics of some commercially available WfMSs seemed adequate both to handle the bulk of the work that is involved with debt collection and to ensure seamless integration with the various

back-office systems. However, experimentation with existing systems indicated too little support for the handling of debt collection exceptions. For each candidate, one of the following observations could be made:

- The process definitions became overly complex when they incorporated all possible kinds of alternative routings in response to expected exceptions. As a result, these definitions became hard to understand, hard to validate, and probably hard to manage in future.
- If less restrictive process definitions were used from the start, then the support provided by the system easily became *too* liberal. Users could "wander" through the process, making legal mistakes when they moved out of the scope of their own work, and spent too much time on coordination.

We looked for extensions that somehow address these issues — in the spirit of the solutions provided by CHSs and known research concepts — while trying to keep intact the obvious strengths of a WfMS. Our starting point was an existing WfMS that was found to be satisfying all other requirements, while we looked further for extensions to handle the remaining 5%. We will first describe the various aspects of our proposal, illustrate these with examples from our case study, and subsequently argue in Sec. 4.2 why a WfMS extended in this way can help to support the process of debt collection. The proposal we describe incorporates both organizational and technical measures.

4.1.1. *Organizational measures*

We cannot over-emphasize the importance of the organizational context in which exceptions occur. In the various workflow projects we participated in so far, this issue is often underrated when it comes to discussing ways of handling exceptions. Because a WfMS is to support the execution of an information process, it may seem logical for some to expect from a WfMS to handle each disruption of normal business operations. However, it is unreasonable to expect a WfMS to cope with *unexpected exceptions* (see Sec. 2.2). A large terrorist attack is a sad but realistic example of an event that could hardly be foreseen, let alone that the response can be predefined. A process definition used to handle insurance claims related to such an event will most probably require very specific build-time modifications.

If we consider *expected exceptions*, then we can make a further categorization. As Strong and Miller[50] have argued, exceptions may be caused by political tensions within a company. No matter how sophisticated the technology, such an underlying cause of disruptions will not be solved without organizational measures. Therefore, in discussing the flexibility of a WfMS we think it is only meaningful to consider expected exceptions that are in principle technologically solvable. We will refer to these exceptions as *addressable.* In a sense, this is comparable to the *resolvable* dimension by Luo *et al.*[39] The first part of our proposal in managing business processes is to invoke organizational procedures to analyze each occasion as it occurs,

to define appropriate ways of handling these in general, and to maintain and update this knowledge in the form of process definitions. We will not discuss this organizational side in this paper, but focus on the second part of the proposal which is technical in nature.

4.1.2. *Technical measures*

The main idea of our proposal is to distinguish each addressable exception explicitly, much in the spirit of the execution mapping that took place in the WIDE project.[14] Instead of specifying process definitions that are inherently flexible, we propose to describe how each type of addressable exception should be handled *before* the WfMS is taken in production, i.e. at build time. For each type of case there will be one main process definition that describes the normal flow. For each addressable exception on this normal flow, one or both of the following actions take place:

(1) The main process definition will be executed in *a specific way.*
(2) Another, *separate process definition* will be executed.

Note that for each newly distinguished addressable exception at run time, a solution in the form of these actions will be defined for further use. The exception itself will be handled in an *ad hoc* manner using the standard administration and definition functions of the WfMS, i.e. an authorized and knowledgeable workflow administrator will manually update the case state and/or extract it from the system for further processing, possibly with a new process definition. This is similar to the approach described by Cugola,[15] who refers to such a person as the process manager.

To illustrate both types of actions, let us consider the debt collection procedure in more detail. In our view, the main process definition consists of the notification of the open debt to the debtor and subsequently of increasingly ponderous measures to collect the debt, e.g. serving summons, send in the bailiffs, imprisonment for debt etc. So, the normal work is considered to be the flow of activities if repayment is not received after (repeated) summons. The repayment is here — rather cynically — the expected, addressable exception that may take place. A *separate process definition* is concerned with handling these repayments. When a repayment is received, this latter process definition is executed. The instance of the main process definition is then completed without superfluous extra summoning, which is *a specific way* of executing the main process definition.

To properly implement this general idea we propose the following three extensions of a WfMS:

(1) The use of *case variables and preconditions.*
(2) Direct access to the *workflow execution status.*
(3) *Jump facilities.*

We will discuss each of these extensions in some more detail.

4.1.3. *Case variables and preconditions*

Case variables are predefined data attributes. They carry information about the case in question. This information may be used to select an alternative predefined path through the main process definition at run time. We recall the debt collection process to illustrate its use: when a repayment is received a separate process definition will be instantiated to handle this repayment. This instantiation sets a general case variable to "repayment received". Each activity summoning the debtor in the main process definition has as a *precondition* that the case variable should evaluate to "repayment not yet received". This precondition is very similar to the notion of a *transition condition* by the WfMC.[25] As a consequence, each of these activities will then be skipped in further executing the main workflow. Other activities may still be executed, e.g. the registration of the last summoning date.

Case variables and preconditions are already known constructs in some commercially available WfMSs, such as COSA. Its addition is comparable to the color that is added to the classical Petri net to make it more manageable.[30]

4.1.4. *Workflow execution status*

Each execution of the main process definition will be assigned a special status, which is observable for each actor working on the case. Actors with sufficient authorization may alter this status in one of possibly many predefined values. This status does not refer to the progress in handling the case, but indicates the *level of processing activity*. We will consider in this paper three basic values of this status. When a process instance is *active*, normal support of the WfMS holds, i.e. work items are allocated to actors in conformance with the main process definition. An active process instance may be turned into a *suspended* mode. As soon as this status is set, all current activities may still be finished by actors but no new work items are assigned to actors anymore. As soon as there are no activities in progress, i.e. all assigned work items are completed, the status of a suspended process can be changed to *stopped*. A suspended or stopped process instance can be turned into the active mode again. Note that the reference model of the WfMC distinguishes a much larger number of common states a process instance may take, namely: initiated, running, active, suspended, complete, terminated, archived.[25] The suspended and stopped modes of our proposal can be seen as substates of the suspended state of the WfMC. The idea of various levels of suspension during enactment can also be found back in the PROSYT system described by Cugola.[15] Aside from stopping enactment on the process level, PROSYT also supports the halting of enactment on the object level (e.g. a document).

The use of the execution status may be illustrated by another example of an addressable exception in the case of the debt collection process. If a notice is received during the execution of the main process definition that a debtor has passed away, this exception is in our view to be handled by instantiating a separate process definition. An actor that initiates the latter immediately suspends the main

process instance and checks in a legally sound way whether the debtor is indeed deceased. There is still the option to resume the execution of the main process definition by turning it into the active status when the notification turns out to be ungrounded, i.e. a fake notification. Note the superiority of this solution with respect to terminating the main process instance.

Not many commercially available WfMSs distinguish a separate workflow execution status apart from the states to monitor the progress of cases. It was already defined as a useful construct in the Mobile framework underlying a research WfMS by Jablonski and Bussler.[29]

4.1.5. *Jump facilities*

When an addressable execution takes place, it may be desirable to continue an instance of the main process definition in an *ad hoc* way. The extension of a WfMS with a jump facility enables an actor to move from one state of the case to another, thereby changing the current set of available work items. This jump is no part of the normal execution flow specified with the main process definition.

We recall the addressable exception of a death announcement of a debtor. If we suppose that the main process definition incorporates paths for collecting debts from "normal" and criminal citizens it may be desirable that issuers of false death announcements are treated as criminal citizens — even if the offense takes place *after* the track for normal citizens was selected at run time. An actor that has suspended the main process instance may now also move the case state in the main workflow to another, so that the activities will be performed that are devised for issuing debts in less lenient ways.

The jump facilities we propose explicitly use the *reachability graph* that can be derived from a process definition.[49] The reachability graph contains all possible states and all possible transitions from one state to another. The use of the reachability graph for jumps was proposed and clearly described by Agostini and De Michelis[5] in their design of the Milano research WfMS. Jumps correspond to additional transitions not possible according to the process definition, but should not introduce new states. Moreover, in our proposal forward and backward, jumps at run time are only permitted by authorized personnel and must be selected from a *predefined* list of *tolerated* transitions from one state in the reachability graph to another. This strictness is in fact different from the approach in Milano, where run-time negotiation can occur between involved parties or a supervising actor to approve of a jump.[5] We will return to this issue at the end of the next section.

4.2. *Review*

We claim that the extension of a WfMS in the way presented in the previous section:

(1) will provide WfMSs with clear and easy-to-use facilities to better handle addressable exceptions,
(2) the important strengths of WfMSs are not seriously impaired.

We will shortly review these claims in this section with respect to our case study. Considering our first claim, a WfMS extended with case variables, preconditions, execution statuses, and jumps is assessed by the technical staff of the governmental agency as an adequate answer to the agency's required ability to deal with exceptions. The existing functionality of the WfMS already offers sufficient functionality for STP. The extensions provide support for addressing exceptions in a very controlled way, which answers the need of the Dutch government to increase the flexibility of work execution while maintaining the (legal) quality of the work and the efficiency of processing. Dividing the main workflow from the separate workflows for the addressable exceptions also helps to make the models more manageable and understandable (see the observations in Sec. 4.1). It will anyhow be a major improvement to the existing situation, where process logic is hard-coded in existing administrative applications. Practical experimentation with the extended WfMS in pilot projects in the last part of 2002 will point out whether our assessment of the actual exceptions and their treatment is correct. The pilot project will also be used to evaluate the applicability of our framework in a practical setting.

Let us take a look at the second claim we make. If we consider one main process definition with its supporting separate process definitions and the interaction mechanisms as discussed, it can be imagined that one large process definition could be designed that integrates these flows, that has precisely the same functionality, and that could be executed by any common WfMS. Therefore, a similar run-time support in terms of efficiency and quality may be expected from either a WfMS or a WfMS extended according to our proposal. What is more, we do not require derivations of the normal way a WfMS operates in the way CHSs do. In particular, there is no need for developing a large set of redundant case data — a small set of predefined attributes implementing the case variables will do — no locking facilities on the case will be required, and no jumps will be allowed that are not explicitly specified at build time. As we have seen in Sec. 3.2, these are precisely the factors that cause a possible impairment of the WfMS strengths by a CHS.

Note that the extensions we propose will not elevate the context tunneling effect in the way a CHS can be expected to do. In general, a smartly chosen set of case variables may improve upon the context support that any WfMS offers. Although many WfMSs already incorporate case variables, their disclosure to actors is not always a standardized facility. This problem can be tackled by developing specific query forms or organization-specific *worklist handlers*.[25] Many WfMS vendors provide an environment of Application Programming Interfaces (APIs) to realize these with limited efforts. It is this direction that we take for limiting the context tunneling effect in the case of the debt collection agency.

There is, however, a price to be paid for our proposed extensions. The main process definition and its separate supporting workflow will not give full information on their possible executions. This is mainly caused by the jumps that can be made through the main workflow at run time. So, the list of the tolerated jumps should accompany the traditional flow diagrams of the process definitions to understand

the overall process. This drawback is similar to that of interpreting a CHS process definition as discussed in Sec. 3.2. However, while the default strategy of modeling a process in a CHS is to model what is *not allowed*, the emphasis in our model is to model what *is allowed*. We believe that the latter is a more natural way to specify and understand models.

The selection of the more meaningful transitions among all the possible transitions through the reachability graph is probably the most time-consuming design effect of our proposal. One may expect a less efficient way of modeling. After all, the number of *possible* transitions is exponential in the number of states in the reachability graph. However, we do not propose to consider each of them. Workshops with experts will be held using the list of addressable exceptions to identify the relevant transitions. So, the search for jumps is sharply focused. An additional advantage of explicitly defining the tolerated jumps is of course that it can be *verified* at build-time whether they lead to logically correct workflow executions. For example, using the techniques as described by Agostini and De Michelis[5] and van der Aalst and Basten.[2] Legal experts can also determine whether these jumps are indeed legitimate, i.e. the jumps can be *validated* at build time. We recall the importance of this latter issue (see Sec. 4.1). Because of these advantages, we expect the real effort of modeling reachability graph transitions to be acceptable.

The fixed character of the tolerable jumps is the most likely aspect to expect difficulties in our work for the Dutch government. Bowers *et al.*,[13] for example, have shown with their fieldwork in the print industry how restrictions imposed by a WfMS may seriously hinder a previously "smooth flow of work". The situation of the Dutch government *with* WfMS support however will probably be experienced as less restrictive than the current process support, that is, as stated before, hard-coded in the government's administrative applications. Nonetheless, we reserve a "second chance mechanism", which resembles the authorization role of the Agency director in the example of the use of the Milano system.[5] If an actor wishes to jump to a state that is not previously identified as correct and valid, an expert may grant run time permission to this jump. In this way, we hope to guard the efficiency and quality of the work being performed. Hands-on experiences will be needed to point out if this mechanism is indeed required and, if so, whether it is feasible given the large number of cases.

Finally, we would like to point out that the proposed system is most likely not the single acceptable solution for the situation we sketched to circumvent the problems sketched in Sec. 4.1. For example, we can imagine that a proper constraint monitoring mechanism in the spirit of Freeflow[19] could also have resulted in transparent process definitions which are not overly liberal. Still, in a practical case as the one described it is always a mix of corporate, commercial, and technical requirements that is used to select a WfMS and its vendor. We showed in the spirit of existing research concepts and the ideas behind CHSs how this existing commercial technology may be extended to offer more flexibility and less context tunneling, while maintaining typical workflow advantages.

5. Selecting the Right Technology

Thus far we have only considered production WfMSs such as Staffware, COSA and MQSeries, case handling systems such as FLOWer and ECHO, and WfMSs extended with the capabilities described in the previous section. We did not address two additional categories of "workware": *groupware systems* such as Lotus Notes (IBM) and *ad hoc workflow systems* such as InConcert (TIBCO), Ensemble (Filenet) and TeamWARE Flow (TeamWARE Group). Although these latter two are not always seen as separate categories,[27] we feel that their different process awareness justifies a distinction. In this section, we first discuss groupware systems and *ad hoc* workflow systems and their potential applicability to our case. Then we characterize all types of "workware" to aid a general selection of the right technology.

The most notable groupware system is Lotus Notes. Systems like Lotus Notes are widely used and provide a lot of operational flexibility. The two basic paradigms are the sharing of information and documents and the communication between actors. Groupware typically involves small teams of professionals and are intended to support short term activities which require a rapid workflow solution.[27] Groupware is usually applied to support non-routine processes.[11] When it comes to the support of work *processes*, we can rule out these systems. Pure groupware systems are unaware of the processes taking place and, therefore, cannot be expected to offer process support.[11,32] Note that it is possible to use groupware systems to implement support for workflow processes. However, in this case either the processes are hard-coded in applications or handed over to specific workflow modules (e.g. Lotus Domino Workflow). In both cases, the groupware system itself is not offering process support. For the case of the Dutch government agency, groupware was not considered as a serious alternative because of the large numbers of cases, the large number of business professionals involved, the similarities between process executions for various cases, and the long cycle times of process instances.

Ad hoc workflow systems such as InConcert, Ensemble and TeamWARE Flow allow for the on-the-fly creation and modification of process definitions. Each case has a private process definition and therefore the traditional problems encountered when changing a process definition can be avoided. *Ad hoc* WfMSs allow for a lot of flexibility. The WfMS InConcert even allows the user to initiate a case having an empty process definition. When the case is handled, the process definition is extended to reflect the work conducted. Another possibility is to start using a template. The moment a case is initiated, the corresponding process definition is instantiated using a template. After instantiation, the case has a private copy of the template which can be modified while the process is running. InConcert also supports "workflow design by discovery": the routing of any completed process instance can be used to create a new template. This way, actual workflow executions can be used to create process definitions. *Ad hoc* WfMSs are attractive from a flexibility point of view. However, there are two important requirements for the

successful application of *ad hoc* WfMSs. The first requirement is that actors are aware of the processes they are dealing with. This means that the processes should only be defined or modified by actors having a good overview of the *whole* process. The second requirement is that actors have the ability to use advanced modeling tools and have a good understanding of process definition techniques. It is essential that modelers can think in terms of sequential, parallel, conditional, and iterative routing. The two requirements often inhibit the application of this technology. In the case of the Dutch government, the business professionals working on debt collection with its various legal implications do not have a complete overview of the actions that are legitimate under all circumstances. Moreover, the volume of cases is too high to design process definitions individually.

Figure 5 shows four categories of workware systems: groupware, *ad hoc* WfMSs, CHSs, and production WfMSs. As argued before, groupware systems are not process-driven and focus exclusively on communication and the sharing of information. Therefore, they are particularly suitable for unstructured processes. On the other end of the spectrum, we find the production WfMSs which are process-driven and only use data for routing purposes. A production WfMS only supports routes through the process that are modeled *explicitly* at *build time*. In Fig. 5, this is referred to as explicitly structured. *Ad hoc* WfMSs allow for on-the-fly changes of a single process instance. At any time the process definition for a case is structured, but the model can be changed while the case is being handled. In Fig. 5, this is referred to as *ad hoc* structured. CHSs support implicitly structured processes. Using the skip and redo roles (cf. Sec. 3), it is possible to allow for routes not directly visible in the graphical representation of the process definition. Moreover, the case automatically moves to a subsequent state if all mandatory data elements are present. Note that we have indicated the *dominant* characteristics of the various systems in Fig. 5.

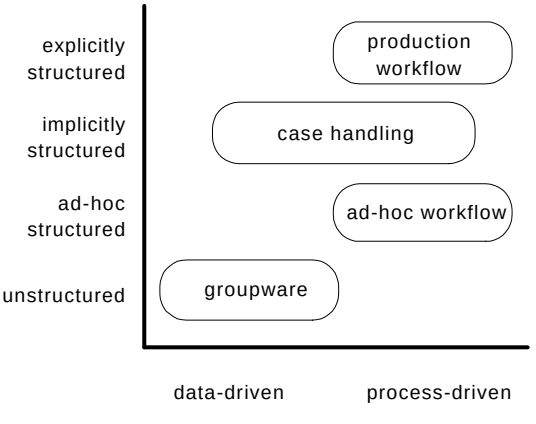

Fig. 5. Positioning four categories of workware.

Georgakopoulos *et al.*[27] express with a similar figure how both the complexity of individual tasks and the complexity of the process relate to the different types of workware. They do not focus on the data or the process drivers of the technology.

Figure 6 shows a comparison of four categories of workware systems with respect to the criteria flexibility and support. Groupware systems offer a lot of flexibility but hardly any support for the processes. If a system is not aware of the processes being executed, one should not expect any process support. Note that groupware systems will support the execution of individual activities, but not the management and enactment of processes. Production WfMSs offer a lot of process support but can only deal with situations that have been modeled explicitly. Groupware and production workflow are two ends of the spectrum. Clearly, *ad hoc* workflow and case handling are in-between these ends. *Ad hoc* WfMSs offer more flexibility than CHSs because it is possible to allow for execution paths that could not be anticipated at design time. However, *ad hoc* WfMSs offer less support than CHSs because the actor is expected to redesign the workflow to enable an alternative route.

Georgakopoulos *et al.*[27] characterize workware in a similar figure along a continuum from human-oriented to system-oriented. Their characterization coincides with the one we presented when we realize that situations where technology cannot offer much support although there is a great need for flexibility typically are very human-oriented. The other end of our spectrum is typically very system-oriented.

How does the discussion in Sec. 4 relate to Figs. 5 and 6? The extended WfMS has not been depicted in Fig. 5. A WfMS extended to handle addressable exceptions will move the WFMS into the direction of CHSs. This means that in Fig. 5 the oval "production workflow" is extended into the direction of "data-driven" and "implicitly structured". Figure 6 shows that there is trade-off between flexibility and support. The goal of the extended WfMS as discussed in Sec. 4 is to find the right balance between these two criteria for the case we discussed.

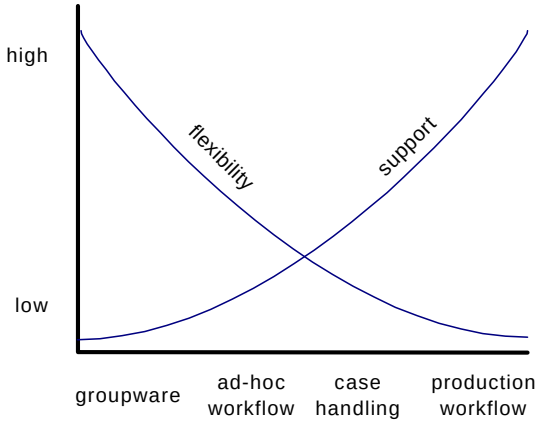

Fig. 6. The trade-off between flexibility and support.

6. Conclusion

We have treated the advantages and drawbacks of CHSs and concluded that they are not universally superior to WfMSs. We presented a proposal to extend the facilities of WfMSs for a specific case with constructs known from real systems and existing research to handle addressable exceptions. We expect that the proposed extensions will be a less serious impairment of WfMS strengths.

At the beginning of this year, the proposal has been accepted by the Staffware corporation as a functional specification for the extension of their WfMS. At the time of writing this paper, the tests of the adapted WfMS have been positively completed. During the second half of the year 2002, pilot projects for the Dutch government will be initiated to test the concept. It is our aim to follow these development carefully to assess the validity of our claims.

References

1. W. M. P. van der Aalst, On the automatic generation of workflow processes based on product structures, *Computers in Industry* **39** (1999) 97–111.
2. W. M. P. van der Aalst and T. Basten, Inheritance of workflows: An approach to tackling problems related to change, *Theoretical Computer Science* **270**, 1–2 (2002) 125–203.
3. W. M. P. van der Aalst and P. J. S. Berens, Beyond workflow management: Product-driven case handling, *Proc. Int. ACM SIGGROUP Conf. Supporting Group Work* (ACM Press, New York, 2001) 42–51.
4. W. M. P. van der Aalst and K. M. van Hee, *Workflow Management: Models, Methods and Systems* (MIT Press, Cambridge, 2002).
5. A. Agostini and G. De Michelis, A light workflow management system using simple process definitions, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 335–363.
6. A. Agostini, G. De Michelis and M. A. Grasso, Rethinking CSCW systems: The architecture of milano, *Proc. Fifth European Conf. Computer Supported Cooperative Work* (Kluwer, Dordreacht, 1997) 33–48.
7. A. Agostini, G. De Michelis, M. A. Grasso, W. Prinz and A. Syri, Contexts, work processes and workspaces, *Computer Supported Cooperative Work* **5**, 2–3 (1996) 223–250.
8. G. Alonso, M. Kamath, D. Agrawal, E. Abbadi, R. Gunter and C. Mohan, *Failure Handling in Large Scale Workflow Management Systems*, IBM Research Report RJ9913, IBM Almaden Research Center, 1994.
9. L. Bannon, Series of 14 responses to the articles of Suchman and Winograd, *Computer Supported Cooperative Work* **3**, 1 (1995) 29–96.
10. G. I. H. M. Bayens and J. B. M. Tönissen, Provision of service automation does not go smoothly, *Automatisering Gids*, 49 (2000) (in Dutch).
11. A. Bernstein, How can cooperative work tools support dynamic group processes? Bridging the specificity frontier, *Proc. Conf. Computer Supported Cooperative Work* (ACM Press, New York, 2000) 279–288.
12. A. Borgida and T. Murata, Tolerating exceptions in workflows: A unified framework for data and process, *Proc. Int. Joint Conf. Work Activities Coordination and Collaboration* (ACM Press, New York, 1999) 59–68.
13. J. Bowers, G. Button and W. Sharrock, Workflow from within and without: Technology and cooperative work on the print industry shopfloor, *Proc. Fourth European Conf. Computer-Supported Cooperative Work* (Kluwer, Dordreacht, 1995) 51–66.

14. F. Casati and G. Pozzi, Modeling exceptional behaviors in commercial workflow management systems, *Proc. Fourth IFCIS Int. Conf. Cooperative Information Systems* (IEEE Computer Society, Brussels, 1999) 127–138.
15. G. Cugola, Tolerating deviations in process support systems via flexible enactment of process models, *IEEE Trans. Software Engineering* **24**, 11 (1998) 982–1001.
16. R. Deen, Flexible handling: The use of case handling, *Workflow Magazine* **6**, 4 (2000) 10–12 (in Dutch).
17. J. Desel and J. Esparza, *Free Choice Petri Nets, Cambridge tracts in Theoretical Computer Science* 40 (Cambridge University Press, Cambridge, 1995).
18. M. Divitini and C. Simone, Supporting different dimensions of adaptability in workflow modeling, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 365–397.
19. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen and A. Zbyslaw, Freeflow: Mediating between representation and action in workflow systems, *Proc. Conf. Computer Supported Cooperative Work* (ACM Press, New York, 1996) 190–198.
20. J. Eder and W. Liebhart, The workflow activity model WAMO, *Proc. Third Int. Conf. Cooperative Information Systems* (1995) 87–98.
21. C. A. Ellis and K. Keddara, ML-DEWS: Modeling language to support dynamic evolution within workflow systems, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 293–333.
22. C. A. Ellis, K. Keddara and G. Rozenberg, Dynamic change within workflow systems, *Proc. Conf. Organizational Computing Systems* (ACM Press, New York, 1995) 10–21.
23. C. A. Ellis and G. J. Nutt, Modeling and enactment of workflow systems, *Application and Theory of Petri Nets*, Lecture Notes in Computer Science 691 (Springer-Verlag, Berlin, 1993) 1–16.
24. G. Faustmann, Configuration for adaptation — A human-centered approach to flexible workflow enactment, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 413–434.
25. L. Fischer, *Workflow Handbook 2001* (Future Strategies, Lighthouse Point, 2001).
26. L. Gasser, The integration of computing and routine work, *ACM Trans. Office Information Systems* **4**, 3 (1986) 205–225.
27. D. Georgakopoulos, M. Hornick and A. Sheth, An overview of workflow management: From process modeling to workflow automation infrastructure, *Distributed and Parallel Databases* **3** (1995) 119–153.
28. P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein and M. Teschke, A comprehensive approach to flexibility in workflow management systems, *Software Engineering Notes* **24**, 2 (1999) 79–88.
29. S. Jablonski and C. Bussler, *Workflow Management: Modeling Concepts, Architecture, and Implementation* (International Thomson Computer Press, London, 1996).
30. K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use* 2, *EATCS Monographs on Theoretical Computer Science* (Springer-Verlag, Berlin, 1992).
31. M. Kamath and K. Ramamritham, Failure handling and coordinated execution of concurrent workflows, *Proc. Fourteenth Int. Conf. Data Engineering* (IEEE Computer Society, Brussels, 1998) 334–341.
32. P. J. Kammer, G. A. Bolcer, R. N. Taylor, A. S. Hitomi and M. Bergman, Techniques for supporting dynamic and adaptive workflow, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 269–292.
33. M. Klein and C. Dellarocas, A knowledge-based approach to handling exceptions in workflow systems, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 399–412.
34. M. Klein, C. Dellarocas and A. Bernstein, Introduction to the special issue on adaptive workflow systems, *Computer Supported Cooperative Work* **9**, 3–4 (2000) 265–267.

35. T. M. Koulopoulos, *The Workflow Imperative* (Van Nostrand Reinhold, New York, 1995).
36. M. Kradolfer and A. Geppert, Dynamic Workflow schema evolution based on workflow type versioning and workflow migration, *Proc. Fourth IFCIS Int. Conf. Cooperative Information Systems* (IEEE Computer Society, Brussels, 1999) 104–114.
37. P. Kueng, The effects of workflow systems on organizations: A qualitative study, *Business Process Management*, Lecture Notes in Computer Science 1806 (Springer Verlag, Berlin, 2000) 301–316.
38. F. Leymann and D. Roller, Workflow-based applications, *IBM Systems Journal* **36**, 1 (1997) 102–123.
39. Z. Luo, A. Sheth, K. Kochut and J. Miller, Exception handling in workflow systems *Applied Intelligence* **13**, 2 (2000) 125–146.
40. G. MacSweeney, Traveling the long road to end-to-end processing, *Insurance and Technology*, **26**, 10 (2001) 30–34.
41. D. Miers and G. Hutton, The business case for case handling, White Paper, Enix Consulting Ltd., http://www.enix.co.uk/caseman.htm, 1997.
42. J. A. Miller, A. P. Sheth, K. J. Kochut and Z. Luo, Recovery issues in web-based workflow, *Proc. 12th Int. Conf. Computer Applications in Industry and Engineering* (1999) 101–105.
43. C. Moore, Evaluation framework: Workflow or workware, GIGA Information Group Report, http://www.enix.co.uk/workware.htm, 2000.
44. W. J. Orlikowski, Learning from notes: Organizational issues in groupware implementation, *Proc. Conf. Computer Supported Cooperative Work* (ACM Press, New York, 1992) 362–369.
45. P. Athena, Case handling with FLOWer: Beyond workflow, Positioning Paper, http://www.pallas-athena.com/downloads/eng_flower/flowerwp.pdf, 2000.
46. E. A. H. Platier, A logistical view on business processes: BPR and WFM Concepts, PhD Thesis, Eindhoven University of Technology, Eindhoven, 1996.
47. W. Prinz and S. Kolvenbach, Support for workflows in a ministerial environment, *Proc. Conf. Computer Supported Cooperative Work* (ACM Press, 1996) 16–20.
48. M. Reichert and P. Dadam, $\text{ADEPT}_{\text{flex}}$ — Supporting dynamic changes of workflows without losing control, *J. Intelligent Information Systems* **10**, 2 (1998) 93–129.
49. W. Reisig and G. Rozenberg, *Lectures on Petri Nets I: Basic Models*, *Lecture Notes in Computer Science* 1491 (Springer-Verlag, Berlin, 1998).
50. D. M. Strong and S. M. Miller, Exceptions and exception handling in computerized information processes, *ACM Trans. Information Systems* **13**, 2 (1995) 206–233.
51. L. A. Suchman, Office procedure as practical action: Models of work and system design, *ACM Trans. Office Information Systems* **1**, 4 (1983) 320–328.
52. L. A. Suchman, Do categories have politics? The language/action perspective reconsidered, *Computer Supported Cooperative Work* **2**, 3 (1994) 177–190.
53. R. van Tol, Workflow systems are not flexible enough, *Automatisering Gids* **11**, 2000 (in Dutch).
54. T. Winograd, Categories, disciplines, and social coordination, *Computer Supported Cooperative Work* **2**, 3 (1994) 191–197.
55. T. E. White and L. Fischer, *The Workflow Paradigm* (Future Strategies, Alameda, 1994).
56. S. Whittaker, Talking to strangers: An evaluation of the factors affecting electronic collaboration, *Proc. Conf. Computer Supported Cooperative Work* (ACM Press, 1996) 409–418.