The Case Handling Case

Hajo Reijers[1], Jaap Rigter[2] and Wil van der Aalst[1]

[1] Faculty of Technology and Management, Eindhoven University of Technology,
PO Box 513, NL-5600 MB, Eindhoven, The Netherlands.
[2] Deloitte & Touche Consultdata,
PO Box 12002, NL-1100 AA, Amsterdam, The Netherlands.
E-mail: h.a.reijers@tm.tue.nl, j.rigter@deloitte.nl w.m.p.v.d.aalst@tm.tue.nl

Abstract
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 traditional workflow management systems is proposed. This approach is being pursued by the Dutch governmental agency CJIB in a project involving the workflow management system Staffware. Based on our experiences thus far, we provide guidelines for selecting the proper technology.

1. INTRODUCTION

A workflow management system (WfMS) is a software system that supports the specification and execution of administrative business processes (Jablonski and Bussler, 1996). The promise of a WfMS is fourfold:

- *less coordination*: the WfMS liberates workers from the routine work they need for coordination,
- *higher quality*: the WfMS will offer to workers at *least* the work which is required to deliver the preferred quality of service,
- *higher efficiency*: the WfMS will offer to workers at *most* the work which is required to produce an acceptable result, and
- *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.

Despite the allure of this promise, WfMS's have not lived up to the expectations that were raised ever since their appearance in the early 90s. Although a WfMS is considered to be an important part of an enterprise systems architecture, its practical application has been limited to the support of simple and well-defined business processes.

A fundamental question is whether the manufacturing paradigm that underlies WfMS's is suitable to be applied to manage office work. Within this paradigm, a process is seen as a set of tasks that have to be carried out by resources according to a pre-defined control structure. In an office environment, resources are a typical mix of human workers and computer applications.

Case handling systems have been presented as an explicit attack on the manufacturing paradigm. A case handling system should empower the workflow participant in two ways. In the first place, a workflow participant is offered a broader view on the process within he or she performs work. *Context tunneling* may be encountered by users of traditional WfMS's, as they are only provided the work which they are required to do and lack insight into earlier and yet to be performed work. In the second place, office workers are not satisfactorily supported by WfMS in handling *exceptions*, i.e. occasions that should cause deviations from the regular, standardized flow of work.

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 researchers (e.g. Agostine and De Michelis, 2000) think that current WfMS's lack to provide to workers: *awareness* of the situation and the *flexibility* to handle changes.

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 of WfMS's, but that their is a price to be paid: traditional WfMS strengths are impaired or even annulled. As an alternative to the solution Case Handling Systems offer, we propose extensions of the traditional functionality of a WfMS's. The proposal is inspired by the need for flexibility as expressed by the Dutch governmental agency CJIB (Centraal Justitieel Incasso Bureau) at the start-up of a workflow implementation. The CJIB employs some 800 people, who are primarily concerned with collecting administrative fines for juridical offences in the Netherlands. 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 Section 2 we will reflect upon related work. In the first half of Section 3, we describe the typical characteristics of case handling systems and their effect on elevating the deficiencies of traditional WfMS's. In the second half of Section 3, we will focus on some of the undesired effects of using case handling systems. Section 4 contains our proposal for extending the capabilities of traditional WfMS's. This section builds upon the particular case of the CJIB to illustrate the proposal. We present our concluding remarks in Section 6.

2. RELATED WORK

2.1. MANUFACTURING PARADIGM

The unsuitability of the manufacturing paradigm for workflow applications has been stated by Miers and Hutton (1997), Van Tol (2000), and various vendors of case handling systems, e.g. Pallas (2000). The similarities and differences between manufacturing and administrative processes have been studied by Platier (1996) and Van der Aalst (1999). They both establish that there are striking similarities, as well as subtle differences. Their general conclusion is that many manufacturing concepts are applicable to the management and execution of administrative processes.

2.2. FLEXIBILITY

The flexibility issue of traditional WfMS's is widely studied, e.g. by Heinl et al. (1999), Van der Aalst and Basten (2002), Kammer et al. (2000), Agostini and De Michelis (2000). 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 (Casati and Pozzi, 1999).

Strong and Miller (1995) 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 (e.g. Suchman, 1983; Gasser, 1986). 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 their cause are 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. (2000) classify exceptions with respect to their impact on workflow model. They distinguish exceptions that can be tolerated by the process or safely ignored and still produce a satisfactory result as *noise*. *Idiosyncratic exceptions* influence a specific job and require a change to the process execution with relation to this job. Finally, *evolutionary exceptions* require changes in the overarching workflow model. Casati and Pozzi (1999) take a slightly different angle in distinguishing between *unexpected* and *expected* exceptions: the first category is treated on the workflow instance level, the second on the process definition level. Van der Aalst and Basten (2002) take a broader perspective and distinguish between changes on the instance and definition level as *ad hoc changes* and *evolutionary changes*.

An unexpected exception is typically handled by halting the process execution and modifying the workflow at the model or instance level, after which a consistent continuation is pursued (e.g. Reichert and Dadam, 1998; Van der Aalst and Basten, 2002; Agostini and De Michelis, 2000). Another approach for unexpected exceptions is to tolerate an inconsistency and to "excuse" it before it can proceed, see e.g. Cugola (1998) and Borgida and Murata (1999).

An expected exception is typically countered by rolling back a process execution until a decision point has been reached from which forward execution can proceed along a different path than previously chosen (Alonso et al., 1996). Research mainly focuses on these types of exceptions. An exhaustive treatment of these approaches 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 (Klein et al., 2000). 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 models to handle exceptions as they occur. The other takes the approach of utilizing process models that are less prescriptive in the first place." In this special issue of CSCW, Agostini and De Michelis (2000) describe the concept of jumps. This is the basis for one of the proposed extensions described in Section 4.

It is interesting to note that few research results have made their way to commercially available WfMS's. Very few existing WfMS's provide support at all for handling exceptions (Casati and Pozzi, 1999; Van der Aalst and Basten, 2002).

2.3. Context tunneling

In contrast to the flexibility issue, the effect of *context tunneling* has not been studied so widely. In conclusion to a qualitative study, Kueng (2000) states 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 counterbalance 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 (1997) provide a very positive evaluation of case handling technology, just like Van Tol (2000) and Bayens and Tönissen (2000). Their arguments are very similar to that of case handling system vendors, e.g. Pallas (2000). Deen (2000) takes a more critical approach. He focuses on the presupposed simplified effort of modeling the control flow of a case handling system. He concludes that no larger model expressiveness or flexibility has been reached by them. Workflow users can more easily interpret the workflow model of a case handling system, although this will only be of real value when these models have a high level of detail. Our contribution 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 Hatton Blue. Some of these products have been discontinued or absorbed in other tools. There are also other case handling tools which are used on top of existing workflow management systems. For example, the Activity Manager developed by BPi can be used on top of workflow management systems such as Staffware, COSA, and IBM MQSeries. In this paper, we focus on the case handling system FLOWer. FLOWer is inspired by the ECHO (Electronic Case-Handling for Offices) system whose development started in 1986 within Philips and later moved to Digital. The project resulted in the case handling system ECHO. Although the product was quite successful, the project was discontinued and some of the people involved started to work on a successor of ECHO under the umbrella of Pallas Athena. This resulted in the case handling tool FLOWer.

The relatively small number makes generalization principally difficult, which is further complicated by differences in architecture, interfacing, etc. There is, however, some consensus that their general features are unique (Van Tol, 2000; Moore, 2000; Van der Aalst and Berens, 2001). 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* (see also Van Tol, 2000; Van der Aalst and Berens, 2001; Bayens and Tönissen, 2000),
- the *process is data-driven* (see also Van der Aalst and Berens, 2001), and
- the *parts of the process model are implicit* (see also Moore, 2000; Van der Aalst and Berens, 2001; Deen, 2000).

We will subsequently address each of these properties and compare them to those of traditional WfMS's. We will also indicate which shortcomings of WfMS's they are aimed for to solve.

System focus is on the case

To clarify relevant notions within a system that supports the management and execution of business processes, Van der Aalst and Van Hee (2002) propose the following dimensions: the process, the case and the resources. If one takes, for example, the various manifestations of a unit of work within a business process, these dimensions are helpful to clarify their distinctions. A *task* can be seen as a structural part of the process definition (process dimension), a *work item* is a task instance that has to be performed for a specific case (process + case dimension), and an *activity* is a task instance that is performed for a specific case by a specific resource (process + case + resources dimension).

Within a WfMS, the process dimension is dominant in the clarification of most notions. A WfMS consists of a *build time* and a *run time* part (Jablonski and Bussler, 1996). 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 engine, the heart of the run time part, provides to the users of the systems lists of the work items that have to be completed. Typically, the presentation of a work item to a worker does not depend on the specific case, but on the capability of the resource to execute the corresponding task. As soon as a resource completes a work item, the engine establishes which new work items arise. It does this on basis of the dependencies within the tasks of the process model – which is again strongly focused on the process.

On the other hand, within a CHS the worker at run-time is not confronted with this fragmentary, task-centered view. Instead, an *entire* case – complete with all the information that it embodies – is handed out to him. The worker can add or update information of the case, or content himself with simply browsing and viewing 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 worker with the proper rights.

This specific property of a CHS is primarily aimed at providing the user of the system a better overview of each of the cases that is handled. In other words, it should avoid *context tunneling*. The worker, for example, can inspect information that has been established earlier by co-workers. Also, it is possible to inspect which tasks still need to be executed. The risk of losing this overview is that workers become alienated from the case. When they see only a part of the larger picture, they cannot determine their contribution to the outcome. As a result, the quality of their work may drop, as well as their job satisfaction.

Aside from this primary aim, a case-centered process supports easier 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 traditional WfMS, the overall status of a case may be more difficult to establish.

Process is data-driven

A traditional WfMS manages the state of a case as the progress with respect to the ordered tasks in a process definition. 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 flow variables. In both types of systems, a strong distinction exists between *logistic information* and *content*. The logistic information is managed by the WfMS. What is more, the WfMS has no knowledge about the information which 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 state of a case within a CHS consists of the *complete* collection of information on the case. 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 a worker may be interested in. It also enables the definition of fine-grained relations between tasks. Instead of using mere precedence relations, it is possible to specify on a data level what is sufficient for an activity to count as completed and which tasks then still need to be performed. This facilitates a much less rigid execution of the process model than is the case in a traditional WfMS. In other words, it counters the *flexibility* problem of WfMS's.

Implicit modeling

The process definition in a WfMS is usually an enumeration of tasks. Between these tasks precedence relations exist, which specify the possible routings through the process. Additionally, in most WfMS process models there is a link with the resources dimension, specifying the authorizations for executing each of the tasks.

Different paths through a process model 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 model in a traditional WfMS is that the designer has to specify what is *permitted*. Any routing which is not specified at build time will not be supported by the WfMS at run time. Deen (2000) and Van Tol (2000) refer to this approach as *explicit modeling*.

In a CHS, the process model is modeled in a different way. Similarly to the WfMS approach, various tasks in the process definition are distinguished, but only a *preferred* or *normal* control flow between these tasks is modeled. By default, a user of the system at run-time is able to redo, bypass or rollback these tasks, possibly diverging from the normal flow. The designer of the process definition can limit these possibilities by assigning proper authorizations for the respective actions. The modeling mechanisms of the CHS's differ. ECHO uses the language CTDL (Case Type Definition Language) to specify the process definition (Miers and Hutton, 1997). FLOWer has a graphical design environment STUDIO, which supports 'execute', 'redo' and 'skip' primitives (Pallas, 2000). At a task level, the roles of workers are specified that can perform these specific primitives for each task, aside from the standard execution authorization. If somebody can 'skip' a task, the control flow turns to the task that according to the normal flow follows the skipped task. If a task is redone, the process returns to the state before its execution.

The modeling approach of CHS's is called *implicit modeling* (Van Tol, 2000; Deen, 2000). Typical for a process model in a traditional WfMS is that the designer has to specify what is *not permitted*. The principal advantage aimed for with a CHS is that it will be easier to develop a model (at build time) which in exceptional situations will conform to the wishes of an end user to select and perform the appropriate tasks (at run time). This is to counter the problem which e.g. Heinl et al. (1999) discovered: 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 worker 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. The CHS approach fits within one of the two main approaches currently being taken by adaptive workflow researchers: utilizing process models that are less prescriptive (Klein et al., 2000) (see Section 2).

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

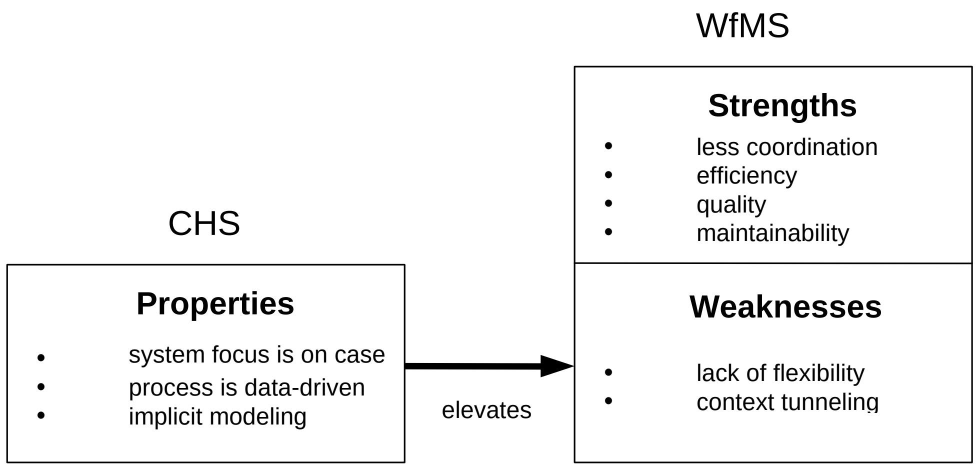

Figure 1: Conceptual effects of CHS's

The three discussed characteristic properties of CHS's are targeted to elevate the two weakness of WfMS's as visualized in Figure 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 of a WfMS. To clarify and illustrate this review, we will try to focus on the essential and commonly shared characteristics of CHS's. For the sake of illustration, we refer to the CHS FLOWer (Pallas, 2000).

System focus is on the case

As stated before (see Section 2), it is not apparent that context tunneling is indeed a problem. Having said that, we are inclined to believe that the system focus of a CHS on the case will actually give a worker a better view on the history of the case, the actual available information, and the work that has to be done still. In other words, it will diminish 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 the worker 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 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. Also, the risk of inconsistencies arises between data in the CHS and other systems, which 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 WfMS’s were out to improve on: By dividing between the logistics and the content of work, both parts would be easier to maintain.

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 a worker may want to follow in different situations, it must be similarly difficult to distinguish what is relevant state information in which a worker 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 traditional WfMS’s such as COSA and Staffware is equipped with case variables. These case variables could be used in to present with a traditional 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.

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. 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 value of the case data, it is very important to avoid inconsistent situations. Suppose that there are two workers with the same responsibilities who are working independently form each other on the same case. Because of various skipping and re-doing of tasks 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. The only 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 workers.

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 tasks, but simultaneous tasks cannot be performed. Note that in a traditional WfMS concurrent execution of tasks 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 (2002) 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 tasks that used to be sequentially ordered. This can be explained from the fact that the total throughput time of a case consists of less than 5 % out of service time (Platier, 1996). In other words, complete concurrent execution is, although not impossible, a rarity.

A straightforward patch for the concurrency problem in a CHS is that disjunct sets of data may be distinguished which can be concurrently processed. However, this undermines the unique system focus and state case of a CHS. In fact, this patch turns the CHS into an ordinary WfMS.

Implicit modeling

Deen (2000) already established that the expressiveness of implicit modeling does not surpass 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 can be expressed in modeling constructions that other WfMS's support. For example, the COSA WfMS uses Petri nets (see e.g. Desel and Esparza, 1995; Reisig and Rozenberg, 1998) as modeling technique. In Figure 2, four Petri net constructions are visualized respectively modeling a task that can and should be executed (A), a task which can be executed or skipped (B), a task that can be executed and possibly redone (C), and a task that can be executed, skipped or redone (D).

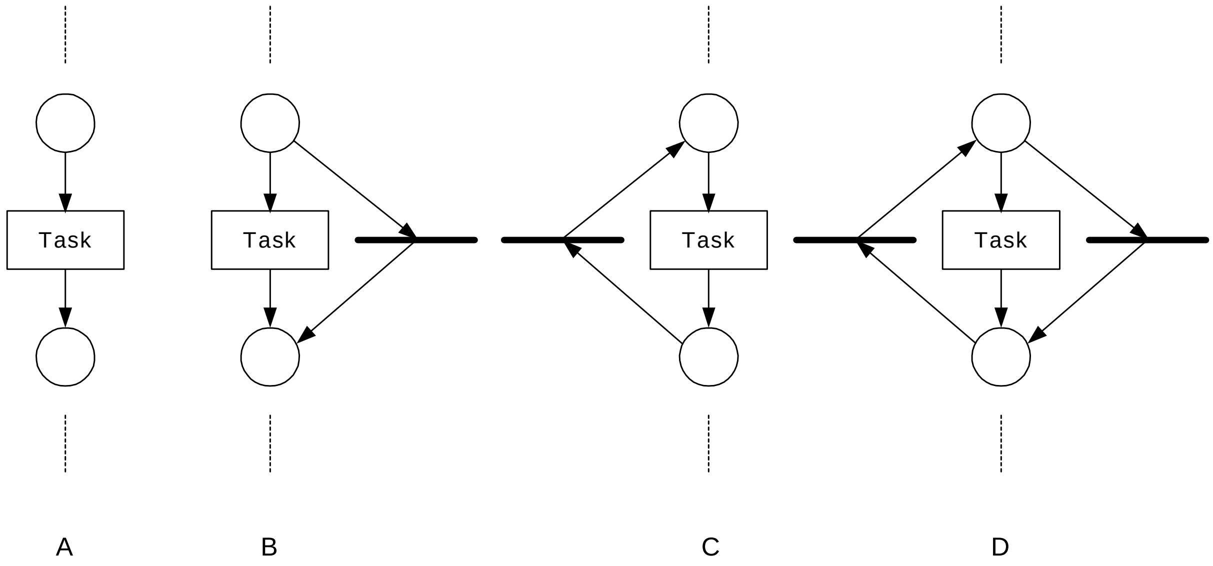

Figure 2: CHS primitives expressed in Petri nets

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 provides this flexibility *at a lower price*. More in specific, we are interested to know:

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

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 (2000) 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 tasks are modeled. Miers and Hutton (1997) observe that the development of a process definition – indeed of the overall application – 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 a implicitly modeled process definition is easier to make, to maintain and to interpret seems to be 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 arrows that obstruct the view.

Let us now consider the example in Figure 3 to examine this supposed ease. In this figure, the left-hand model represents a CHS process definition. Alongside each task, its task 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 task A when task C has been completed. To determine this for a FLOWer model, we have to know the redo statuses of tasks A, B, and C. After all, the semantics of the FLOWer engine is such that all tasks between 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.

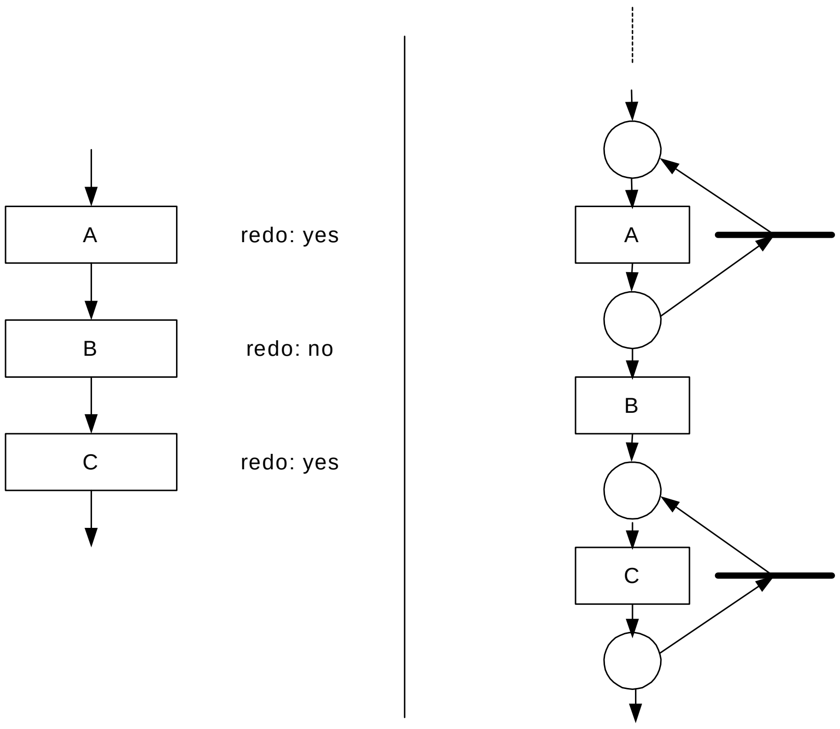

Figure 3: Examples of a CHS and WfMS process definition

Although we used a specific example, we believe that the simplicity of the CHS model is reached by omitting information which is relevant to understand what is really happening. We conjecture therefore that possible flow paths cannot be identified more easily in an implicit process model. 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 this first glance. Even an experienced modeler of implicit process definitions may find it hard to decide whether an *unwanted* sequence of tasks may occur on basis of an arbitrary model. This is a quality aspect, which traditional WfMS aim to improve upon (see Section 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 tasks 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 workers 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 a worker 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 worker for completion of the process.

We have named the quality and efficiency aspects among the primary promises of WfMS's in Section 1: precisely those tasks 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 WfMS's.

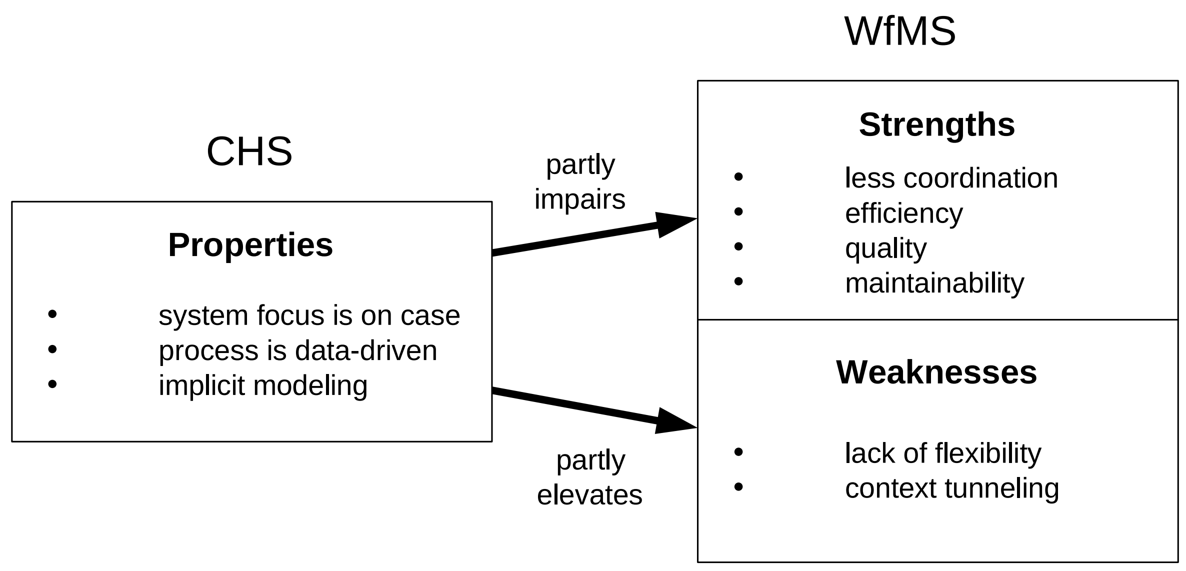

Figure 4: Actual effects of CHS's

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 WfMS's with respect to the decreased coordination effort, maintainability, efficiency, and quality. This is visualized in Figure 4. A general superiority of CHS's 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 WfMS's, we present in this section yet another. However, our proposal combines existing views on exceptions, existing technology, and existing research concepts. The innovative aspect is that this combination is unique and will be actually implemented in one of the world's leading WfMS's. Our presentation in this section is illustrative rather than formal.

To start with, we clearly want to distinguish the type of exceptions that a WfMS can handle. Because a WfMS is to support the execution of a business process, it may seem logical to expect from a WfMS to handle each disruption of normal business flow. However, it is unreasonable to expect a WfMS to coop with *unexpected exceptions* (see Section 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 workflow specification 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 (1995) 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 want to stress that it is only meaningful to consider expected exceptions that are in principle technologically solvable. We will refer to these exceptions as *addressable*. 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 workflow specifications. We will not discuss this organizational side in this paper, but focus on the second part of the proposal which is technical in nature.

We propose to extend the technical facilities of a WfMS to handle addressable exceptions *without* jeopardizing its main strengths. In other words, we will propagandize the use of existing technology, which is embedded in organizational procedures as we mentioned. We will illustrate the use of these facilities with the case of the Dutch governmental agency CJIB (Centraal Justitieel Incasso Bureau). Among other responsibilities, the CJIB takes care of collecting fines levied because of traffic offenses. ** aantallen fines per jaar, andere kengetallen **

The main idea of our proposal is to distinguish each addressable exception explicitly. Instead of defining workflow specifications 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 workflow 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 workflow will be executed in *a predefined, specific way*,
2. a *predefined, separate workflow* will be executed.

Note that for each newly distinguished addressable exception at workflow 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 functions of the WfMS.

To illustrate both types of actions we consider the procedure the CJIB uses to collect traffic fines. In our view, the main workflow consists of the notification of the fine to the wrongdoer and subsequently of increasingly ponderous measures to collect the fine, e.g. serving summons, send in the bailiffs, imprisonment for debt, etc. So, the normal flow is considered the flow of work that should be carried out if payment is not received after (repeated) summons. The payment is here (rather cynically) the expected, addressable exception that may take place. A *separate workflow* is concerned with handling these payments. When a payment is received, this latter workflow is executed. The main workflow is then completed without superfluous extra summoning, which is *a specific way* of executing the normal flow.

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.

Case variables and preconditions

Case variables are predefined data attributes. They carry information on the state of the case in question. This information may be used to select an alternative predefined path through the main workflow process. We recall the example of the CJIB to illustrate its use: when a payment is received a separate workflow will handle this payment. This separate workflow sets a general case variable to 'payment received'. Each task summoning the offender in the main workflow has as a precondition that the case variable should evaluate to 'payment not yet received'. As a consequence, each of these tasks will then be skipped in further executing the main workflow. Other tasks may still be executed, e.g. the registration of the last summoning date.

Case variables and preconditions are already known constructs in some WfMS's, such as COSA. In general, its addition does not expand the expressiveness of a modeling language. Its addition is comparable to the color that is added to the classical Petri net to make it more manageable, see e.g. Jensen (1992).

Workflow execution status

Each execution of the main workflow will be assigned a special status, which is observable for each worker on the case. Workers 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 activity of the workflow execution. We will consider three basic values of this status. When a workflow is *active,* normal support of the WfMS holds, i.e., work items are allocated to workers in conformance with the predefined specification of the main workflow. An active workflow may be turned into a *suspended* mode. As soon as this status is set, all current activities may be finished but no new work items are assigned to workers anymore. As soon as there are no activities in progress, i.e., all assigned workflow items are completed, the status of a suspended workflow turns to *stopped*. A suspended or stopped workflow can be turned into the active mode again.

The use of the execution status may be illustrated by another example of an addressable exception in the case of the CJIB. If a notice is received during the execution of the main workflow that an offender has passed away, this exception is in our view to be handled by a separate workflow. A worker that starts this latter workflow immediately suspends the main workflow and checks whether the offender is indeed deceased. There is still the option to resume the execution of the main workflow by turning it into the active status when the notification turns out to be a creative way of dodging a fine. Note the superiority of this solution with respect to aborting the main workflow.

Not many existing WfMS's distinguish a separate workflow execution status. It has already been defined as a useful construct in the Mobile framework underlying a research WfMS by Jablonski and Bussler (1996).

Jump facilities

When an addressable execution takes place, it may be desired to continue the main workflow in an ad-hoc way. The extension of a traditional WfMS with a jump facility enables a worker 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.

We recall the addressable exception of a death announcement of an offender. If we suppose that the main workflow incorporates paths for collecting fines from normal and criminal citizens it may be desirable that issuers of false death announcements are treated as criminals – although the offense takes place only after the normal path was already selected. A worker that has suspended the main workflow may now also move the case state in the main workflow to another, so that the tasks will be performed that are devised for issuing fines from criminals.

The jump facilities we propose explicitly use the *reachability graph* that can be derived from a workflow (Reisig and Rozenberg, 1998). 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 by Agostini and De Michelis (2000) in their design of the Milano research WfMS. Jumps correspond to additional transitions not possible in standard workflow and should not introduce new states. Moreover, forward and backward jumps in a workflow are only permitted by authorized personnel and must be selected from a list of *predefined tolerated* transitions from one state in the reachability graph to another.

4.2. Review

We claim that the extension of a traditional WfMS in the way presented in the previous section:
(1) will provide a WfMS's with clear and easy-to-use facilities to better handle addressable exceptions, while
(2) the important strengths of WfMS's are not seriously impaired.
We will shortly review these claims in this section.

Our first claim is based on an examination of the typical exceptions that take place at the CJIB. A traditional WfMS extended with case variables, preconditions, execution statuses, and jumps seems to be an adequate answer to the company's required ability to deal with exceptions. Dividing the main workflow from the separate workflows for the addressable exceptions helps to make the models more manageable and understandable. Their actual use will facilitate the desired handling of cases. 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.

The second claim we make deserves some more attention. If we consider one main workflow with its supporting separate workflows and the interaction mechanisms as discussed, it can be imagined that one large workflow specification 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 traditional 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 CHS's 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 Section 3.2, these are precisely the factors that cause a possible impairment of the WfMS strengths by a CHS. Note that the extension we propose will not elevate the context tunneling effect in the way a CHS can be expected to do. Still, a smartly chosen set of case variables may improve upon the context support that traditional WfMS's offer.

There is, however, a price to be paid at the build time part of the workflow model. The workflow specification of the main workflow and its separate supporting workflow will not give full information on the possible workflow 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 to understand the overall process. This drawback is similar to that of interpreting a CHS specification as discussed in Section 3.2. However, while the default strategy of modeling a workflow 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 explicit requirement to model all possible transitions through the reachability graph is probably the most time-consuming 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 Van der Aalst and Basten (2002) and Agostini and De Michelis (2000). Therefore, we expect the real effort of modeling reachability graph transitions to be acceptable. Further work at the CJIB will be needed to support our conviction.

5. SELECTING THE RIGHT TECHNOLOGY

Thus far we have only considered traditional production WfMS's such as Staffware, COSA and MQSeries, case handling systems such as FLOWer and ECHO, and WfMS's 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). In this section, we first discuss the latter two categories and then characterize all types of "workware" to aid the 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 workers. 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. 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.

Ad-hoc workflow systems such as InConcert, Ensemble and TeamWARE Flow allow for the on-the-fly creation and modification of workflow processes. Each case has a private process model and therefore the traditional problems encountered when changing a workflow specification can be avoided. Ad-hoc workflow management systems allow for a lot of flexibility. The WfMS InConcert even allows the user to initiate a case having an empty process model. When the case is handled, the workflow model is extended to reflect the work conducted. Another possibility is to start using a template. The moment a case is initiated, the corresponding process model 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 workflow instance can be used to create a new template. This way, actual workflow executions can be used to create workflow process definitions. Ad-hoc WfMS's are attractive from a flexibility point-of-view. However, there are two important requirements for the successful application of ad-hoc workflow management systems. The first requirement is that workers are aware of the processes they are dealing with. This means that the processes should only be defined or modified by workers having a good overview of the *whole* process. The second requirement is that workers have the ability to use advanced modeling tools and have a good understanding of process modeling 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.

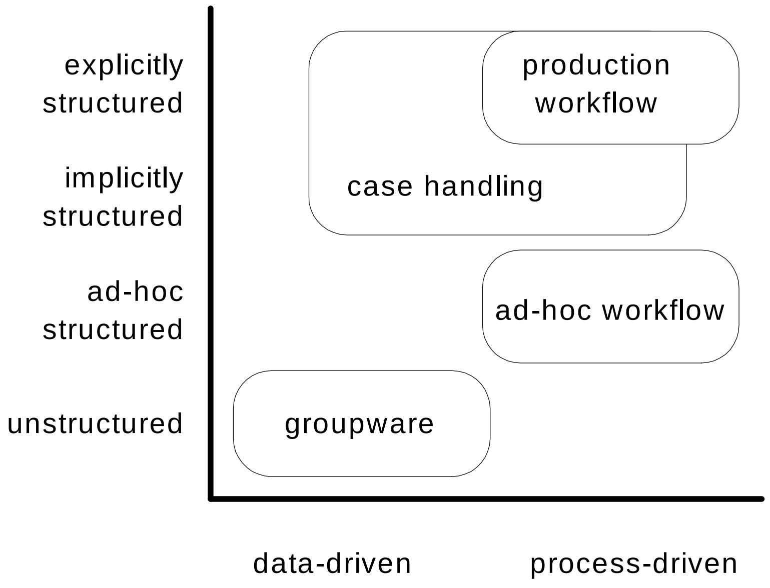

Figure 5: Positioning four categories of workware.

Figure 5 shows four categories of workware systems: groupware, ad-hoc WfMS's, CHS's, and production WfMS's. 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 WfMS's 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 *design time*. In Figure 5, this is referred to as explicitly structured. Ad-hoc WfMS's allow for on-the-fly changes of a single workflow instance. At any time the process model for a case is structured, but the model can be changed while the case is being handling. In Figure 5, this is referred to as ad-hoc structured. CHS's support both explicitly and implicitly structured processes. Using the skip and redo roles (cf. Section 3) it is possible to allow for routes not directly visible in the graphical representation of the model. Moreover, the case automatically moves to a subsequent state if all mandatory data elements are present.

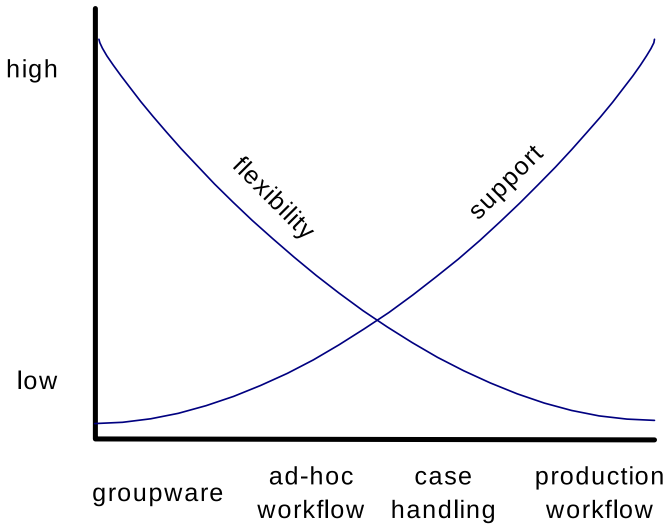

Figure 6: The trade-off between flexibility and support.

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 workflow processes being executed, one should not expect any process support. Note that groupware systems will support the execution of individual tasks, but not the management and enactment of processes. Production WfMS’s 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 WfMS’s offer more flexibility than CHS’s because it is possible to allow for routings that could not be anticipated at design time. However, ad-hoc WfMS’s offer less support than CHS’s because the worker is expected to redesign the workflow to enable an alternative route.

How does the discussion in Section 4 relate to figures 5 and 6? The extended WfMS has not been depicted in Figure 5. A WfMS extended to handle addressable exceptions will move the WFMS into the direction of CHS’s. This means that in Figure 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 Section 4 is to find the right balance between these two criteria.

6. CONCLUSION

We have treated the advantages and drawbacks of CHS's and concluded that they are not universally superior to traditional WfMS's. We presented a proposal to extend the facilities of traditional WfMS's 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 traditional WfMS strengths.

At the time of writing this paper, the proposal has been accepted by the Staffware corporation as a functional specification of the extension of their WfMS. During the year 2002, Staffware will be extended in this way and pilot projects at the Dutch CJIB will be initiated to test the concept. It is our aim to follow these development carefully to assess the validity of our claims. Further work on the side of Deloitte & Touche will focus on developing organizational procedures to imbed the concept within an organizational context and systems integration between the WfMS and transaction systems. From a research perspective, we will continue to investigate the different aspects of workflow modeling on quality, efficiency, and maintainability issues.

REFERENCES

W.M.P. van der Aalst. On the Automatic Generation of Workflow Processes Based on Product Structures. *Computers in Industry,* 39:97-111, 1999.

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):125-203, 2002.

W.M.P. van der Aalst and P.J.S. Berens. Beyond Workflow Management: Product-Driven Case Handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001), 42-51. ACM Press, New York, 2001.

W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, 2002.

A. Agostini and G. De Michelis. A Light Workflow Management System Using Simple Process Models. *Computer Supported Cooperative Work,* 9(3/4): 335-363, 2000.

G. Alonso, M. Kamath, D. Agrawal, E. Abbadi, R. Gunter, and C. Mohan. Failure Handling in Large Scale Workflow Management Systems. Technical Report RJ9913, IBM Almaden Research Center, 1994.

G.I.H.M. Bayens and J.B.M. Tönissen. Provision of Service Automation Does Not Go Smoothly. *Automatisering Gids*, 49, 2000 (In Dutch).

A. Borgida and T. Murata. Tolerating Exceptions in Workflows: A Unified Framework for Data and Process. In *Proceedings of International Conference on Work Activities Coordination and Collaboration*, 1999.

F. Casati and G. Pozzi. Modeling Exceptional Behaviors in Commercial Workflow Management Systems. In *Proceedings Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS 99)*, IEEE Computer Society, Brussels, 1999.

G. Cugola. *Inconsistencies and Deviations in Process Support Systems*. PhD thesis, Politecnico di Milano, Milan, 1998.

R. Deen. Flexible Handling: The Use of Case Handling. *Workflow Magazine*, 6(4):10-12, 2000.

J. Desel and J. Esparza. *Free Choice Petri Nets.* Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press, Cambridge, 1995.

L. Gasser. The Integration of Computing and Routine Work. *ACM Transactions on Office Information Systems,* 4(3): 205-225, 1986.

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): 79-88, 1999.

S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.

K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use* 2. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992.

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): 269-292, 2000.

M. Klein, C. Dellarocas, and A. Bernstein. Introduction to the Special Issue on Adaptive Workflow Systems. *Computer Supported Cooperative Work,* 9(3/4): 265-267, 2000.

P. Kueng. The Effects of Workflow Systems on Organizations: A Qualitative Study. In *Business Process Management*, Lecture Notes in Computer Science nr 1806, 301-316, Springer Verlag, Berlin, 2000.

G.D. Michelis and M.A. Grasso. Situating Conversations Within the Language/action Perspective: The Milan Conversation Model. In *Proceedings of Computer Supported Cooperative Work*, 89-100, 1994.

D. Miers and G. Hutton. The Business Case for Case Handling. White paper, Enix Consulting Ltd., http://www.enix.co.uk/caseman.htm, 1997.

C. Moore. Evaluation Framework: Workflow or Workware. GIGA Information Group report, http://www.enix.co.uk/workware.htm, 2000.

Pallas Athena. Case Handling with FLOWer: Beyond Workflow. Positioning paper, http://www.pallas-athena.com/downloads/eng_flower/flowerwp.pdf, 2000.

E.A.H. Platier. A Logistical View on Business Processes: BPR and WFM Concepts. PhD Thesis, Eindhoven University of Technology, Eindhoven, 1996.

M. Reichert and P. Dadam. ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows without Losing Control. *Journal of Intelligent Information Systems*, 10(2): 93-129, 1998.

W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

D.M. Strong and S.M. Miller. Exceptions and Exception Handling in Computerized Information Processes. *ACM Transactions on Information Systems*, 13(2): 206-233, 1995.

L.A. Suchman. Office Procedure as Practical Action: Models of Work and System Design. *ACM Transactions on Office Information Systems*, 1(4): 320-328, 1983.

R. van Tol. Workflow Systems Are Not Flexible Enough. *Automatisering Gids*, 11, 2000 (In Dutch).