How Can Cooperative Work Tools Support Dynamic Group Processes? Bridging the Specificity Frontier

Abraham Bernstein
Department of Information Systems
New York University – Stern School of Business
44 West 4th Street, Suite 9-76
New York, NY 10012 USA
avi@acm.org

ABSTRACT
In the past, most collaboration support systems have focused on either automating fixed work processes or simply supporting communication in ad-hoc processes. This results in systems that are usually inflexible and difficult to change or that provide no specific support to help users decide what to do next.

This paper describes a new kind of tool that bridges the gap between these two approaches by flexibly supporting processes at many points along the spectrum: from highly specified to highly unspecified. The development of this approach was strongly based on social science theory about collaborative work.

Keywords
Process Specificity, Process Support System, Mixed-Initiative Systems, Dynamic/Improvisational Change.

INTRODUCTION
Many researchers have commented on the increased pace of change in today's economy. Increasingly groups and organizations have to adapt their processes to rapid changes arising from new technologies, new customer demands, or new competitors. Current process-support systems (e.g., ERP, workflow management systems), however, are usually focused on supporting fixed organizational processes. Typically they are too rigid to easily support changing processes. They are mainly used for highly specified and highly routinized organizational processes. As an alternative, many organizations use communications support systems or Groupware (like email or Lotus Notes) to support their rapidly changing, non-routine processes. But these systems typically require users to do a lot of work themselves to keep track of and understand the ongoing processes: what has been done, what needs to be done next, and so forth.

CSCW'00, December 2-6, 2000, Philadelphia, PA.

This dichotomy is paralleled by an old debate in the CSCW-literature about the nature of collaborative work (see [1-5] among others). Both sides in this ongoing debate present some deeply rooted beliefs about how human actors perceive the world and decide to act.

One side follows the belief that human actors typically follow the cycle of problem analysis, solution search or synthesis, and then the execution of that plan. The goal of a process-oriented collaboration support system in this perspective is to increase the speed and efficiency of each of the steps in the cycle as well as facilitate their seamless integration. Workflow management systems (WfMS) and other process support systems like enterprise resource planning systems (ERP) are based on this research stream and have typically focused on the execution of standardized, predefined organizational process (e.g., [6-9] and others).

The other side sees plans as resources for action [3], which are used in conjunction with the environment to articulate and reason about the next action steps [10-12]. Following this perspective typical WfMSs are too restrictive as they traditionally prescribe the workflow and do not allow users to adapt the process to the local situation. Therefore, researchers following this tradition have often advocated using flexible communication support systems (like email or discussion databases) or repositories (e.g., document management/imaging systems) to support organizational processes. Those systems, however, have the disadvantage that an actor typically is on his/her own in deciding what to do next.

To date, none of the approaches has offered a conclusive answer. I concur with others (e.g., [13-15]) that organizational activities often include a mix of both procedure-like and ad-hoc type parts. The research presented in this paper, therefore, argues in favor of bridging between both perspectives by developing systems that will support the whole range of dynamic organizational activity: from well-specified and routine (reacting to exceptions as they occur) to highly unspecified and situated.

In the remainder of this paper, I will first ground this novel idea in a practical scenario and social science theory. This will help to ground and explain the approach as well as to

facilitate the presentation of the proof of concept prototype system. I will conclude with a brief survey of related work and a discussion of the major lessons learned.

A SCENARIO: HEIDI'S PROBLEM

It is Friday afternoon in Zurich, Switzerland and Heidi, a local account manager for Zing Computers (pseudonym), a worldwide producer of computers, gets a phone call from the Swiss stock exchange. They ask for a delivery of a RT2000-server within 48 hours to Zurich, since they need to replace an existing server that got damaged in a fire to recommence trading on Monday morning. Heidi now faces the problem that the traditional order-entry/fulfillment system will not be able to accommodate this request, since the truckers in the European Union (EU) are on strike and the major assembly plant for Europe is in Rotterdam. The only other tools available to her are communication support systems like email, telephone or fax, which give her all the flexibility she needs. However, that also puts the burden of contextual sensemaking (i.e., understanding the context of the task) on whomever gets her messages/faxes/phone-calls.

THE CONCEPTUAL FRAMEWORK

The conceptual framework starts with the commonalties between the situated and the procedural approaches. I believe that both approaches share some minimal assumptions about human actors. First, human actors are boundedly rational and have only limited knowledge about the future. Consequently, plans (as well as process maps or Workflow descriptions) are often imperfect, since they typically can not account for all possible circumstances. A process support system will therefore have to allow for run-time changes to the original plan and will have to provide contextual information about the running process to the actor as a basis for reasoning about the possible next steps. Process maps, a representation of plans, can serve as part of such contextual information [3, 16, 17].

Second, as Newell and Simon [13] point out, our environment includes well-structured and less well-structured problems. Consequently, we have problems with well-defined solution strategies and others, where the solutions strategy is rather unclear [15]. The transparency of the solution strategy (which can be represented as a process map) may change over time, as our understanding of the problem changes. As an elusive problem becomes, for example, better understood its solution strategy may become easier to determine. Or a seemingly simple problem may become highly complex, as new facets of the problem emerge during problem solving, rendering the original solution strategy inapplicable.

The Specificity Frontier

The first consequence of this approach in regard to the enactment of activity is that the specificity of process structure changes over time. Bernstein and Schucan [18], for example, provide a description of how the money-transfer process gained specificity over time. Before the formalities of banking were established, this process started as a vaguely specified process involving an ad-hoc letter sent by a courier. With increasing maturity of the banking industry, the specificity of the process increased significantly. Today, a money transfer is a fixed computer-based inter-bank clearing process with a fixed set of attributes.

This illustrates the major pillar of this conceptual framework: organizational processes lie on a continuum from highly specified and routine processes at one extreme to highly unspecified and dynamic processes at the other extreme. I call this continuum the specificity frontier (see Figure 1). A whole series of points on this frontier are possible, from a highly specific to highly unspecific.

As Figure 1 depicts the concept of a specificity frontier in some sense bridges the gap between the structured WfMS and the unstructured communication systems. It allows for the co-existence of well-specified and almost procedurally executed processes (traditionally supported by WfMSs), and emergent situated processes (typically supported by communication support systems). It also argues that those two types of processes are at the extremes of a frontier of processes. It proposes that the whole range of processes, from highly specified and routine to highly unspecified and dynamic should be supported.

Heidi's problem, for instance, starts out as having a reasonably well-specified solution strategy (process). When she, however, realizes that the truckers in the EU are on strike the process suddenly becomes much more problematic: the known description is not applicable anymore. Thus a support system that allows processes to start out as being well defined (and supported by a WfMS-type technology) and lets the structure become flexible (and supported by a groupware technology) as soon as she finds out about the strike would be ideal for her.

Consequently, a model of business processes should be able to capture a range of process specificity (from well specified to highly unspecified). A process support system should be able *to interpret process models with varying degrees of specificity*. Furthermore, it should *support users when changing the processes' specificity at run-time.* Achieving those goals it can close the specificity gap (pictured as a question mark in Figure 1) between traditional process-support systems and communication support systems, and thus bridge systems following the workflow tradition and the situated action tradition.

Emergent Activity Relies on Structure

The second consequence of those commonalties (i.e., bounded rationality and varying specificity of tasks) is illustrated in Orlikowski [19], which shows how change can be understood as a series of improvisational embellishments to existing practice. In other words: the actors attempt to solve the problem at hand following their interpretation of the structure and the current context.

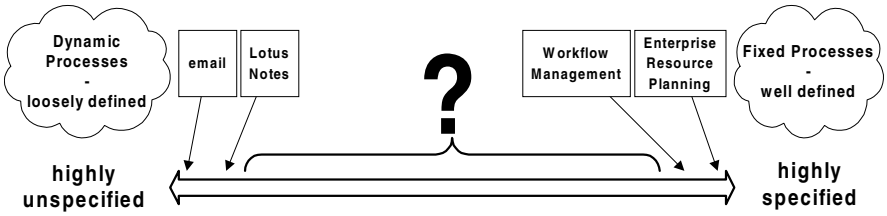

Figure 1: The Specificity Frontier

This illustrates the second pillar of the conceptual framework: that *emergent activity relies on some form of structure and thus some form of specificity*. Emergent activity surfaces "… unpredictably from complex social interactions" ([20], p. 588). However, we may be able to support it by supplying a fertile environment for new solutions to emerge, "… much as does a supersaturated solution in the moment it is disturbed" ([21], p. 267). For example, Jazz improvisation, a type of emergent activity, depends on the actors "… having absorbed a broad base of musical knowledge" ([22], p. 492). Analogously, people in an organizational context must have some foundational knowledge about the task at hand. In addition, as Weick [23] points out '… improvisation does not materialize out of thin air…' (p. 546). People need something to improvise on. This explains the limited success of communications-support systems for business process support: from an improvisational standpoint human actors using those systems incur the overhead of having to understand the context of the task at hand as a basis for improvisation. In the domain of organizational activity, a process map with a low degree of specificity and information about the enactment context could help actors in their sensemaking, provide a basis to improvise on and thus a fertile environment for emergent processes.

Consequently, any system that plans to support emergent activity (which is what all activity is to a certain degree following the situated action approach) *should provide some structure as a contextual basis for situated improvisation*. Process maps (in analogy to geographical maps) can provide such a structure.

Other Requirements

Previous research (see [24-27] among others) has shown that a process support system, also should allow for the change, composition and execution of processes at run time as well as provide some means to be integrated into an existing environment (e.g., using an open interface).

THE SPECIFICITY FRONTIER APPROACH AND PROTOTYPE SYSTEM

Now that I have explained the theoretical grounding for the prototype system I will present the major design ideas I used. I will discuss the proof-of-concept prototype system, which served to clarify, illustrate and evaluate those design ideas. Since some of the design ideas can be abstract without practical example, I will walk through Heidi's problem as a practical usage scenario that will explain the day-to-day usage of the prototype system as I introduce new concepts.

Key Ideas

The major obstacle in designing a process support system following my conceptual framework is the need for an implementation approach, which can handle process specifications at multiple points of the specificity frontier as well as transformations of the specificity of a process during execution. As Figure 2 shows, I chose to ***divide the specificity frontier into sub-spectra***, each supported by its own interpretation logic. I decided to use four sub-spectra, since existing process-support technology (email/groupware, constraint monitoring, constraint-based planning, and transaction processing) could be categorized into four groups: providing context for enactment, monitoring constraints about the task, providing/planning options to reach a goal, and guiding through a given script.

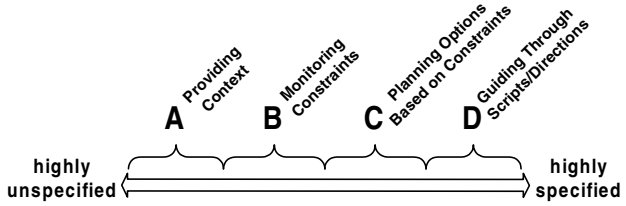

Figure 2: Different Execution Types

The second idea was to develop ***run-time transfer-mappings between the sub-spectra.*** So processes can be ***seamlessly moved to another sub-spectrum*** by increasing or decreasing the specificity of the process definition during run time.

Specifying and Interpreting Processes Models with Varying Degrees of Specificity

Providing Context

In the least specified of the sub-spectra (A, on the left in Figure 2) the support system does not have a lot of information about the process. Therefore its major goal is to provide context for the user to be able to decide what to do next. Similar to the Task Manager presented by Kreifelts et. al. [28], the system therefore helps the users to *share to-do lists and documents (resources)*, which are specific to the task context at hand. The system also integrates with other communication techniques like email and on-line discussions, as well as on-line synchronous communication support such as chat, to allow users to communicate with their respective collaborators. The specificity of the task to the user may vary depending on the information contained in the documents. The system's support, however, will remain the same throughout this sub-spectrum, since the system cannot decode any of the information in the documents.

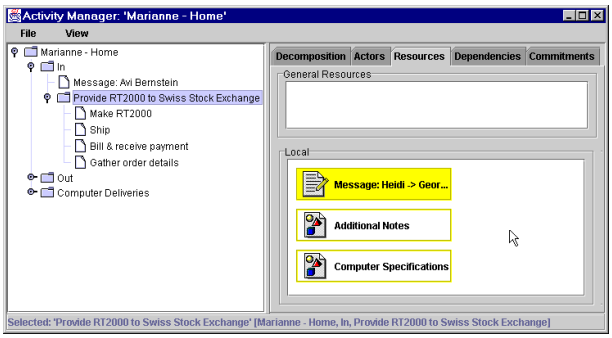

Figure 3: Activity Manager

This is exactly the type of support Heidi needs to start solving her delivery problem. Since she has to collaborate with Marianne, a European logistics manager in Rotterdam, she should be able to share information about the problem and collect information about the tasks to be done (e.g., build the new server, arrange shipment and billing, etc.). As we can see in Figure 3, the system provides a hierarchical to-do list on the left, and shows the resources associated with the task selected. Whenever Heidi writes a new document in the context of this task (like the highlighted message to George on the right in Figure 3) it gets automatically added to the resources connected to the task and complements the context.

From an implementation standpoint the system should provide a shared, distributed-accessible, hierarchical to-do list, which allows users to attach files (as resources) to each of the to-do items. I chose to implement each to-do item as a software-agent, which manages collections of other to-do items and of pointers to files in an object-oriented document repository. As we will see, the choice of active software agents, rather than a passive data structure, becomes advantageous when passing the boundary to the next sub-spectrum.

Monitoring Constraints

When the user decides to add some machine-readable constraint to a to-do item, the system provides constraint-monitoring services. For instance, adding a deadline to a to-do item could allow the system to prompt the user when the deadline is imminent (similar to a project management system). The system's support in this sub-spectrum is comparable to the support a map provides to a hiker. It shows the ravines and the mountains in the area and may therefore help the user to reach his/her goal without long detours by alerting him/her of an obstacle (i.e., a constraint). The more constraints are specified by the user the more helpful the system can be in helping the user to reach his/her goal. Summarizing, the system *helps the user by managing constraints between tasks and resources*.

As Heidi and Marianne quickly discover, there are a series of constraints that they have to keep track of: the deadline for delivery, the type of server, the facts about the strike, etc. When those constraints get specified the system can help them to observe them by reminding them whenever they are about to invalidate one of the constraints. If they were to become late at arranging the shipment, for example, the system would alert them of the impending problem of a late shipment. So Heidi and Marianne add the most relevant constraints (see Figure 4) to the "Provide RT2000"-process.

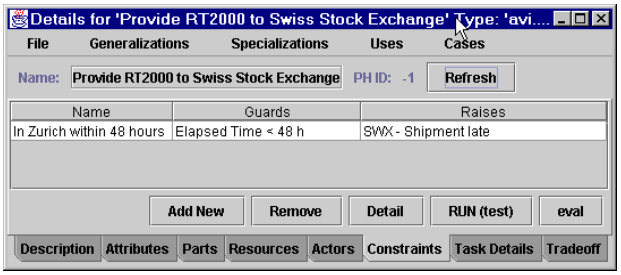

Figure 4: Adding Constraints

In this sub-spectrum the system thus offers the constraint monitoring services in addition to the context provision services. Actors therefore still have the same context information on which to decide what to do next. The boundary between the two sub-spectra is thus crossed as soon as at least one formalized constraint is defined.

Users form the constraints on attributes of the activities/to-do items or resources in the existing process models. Figure 4 for example shows a constraint defined on the attribute 'Elapsed Time' of the 'Provide RT2000'-process. I understand that users sometimes experience difficulties using formalized specification languages, such as Boolean expressions. In the long run I hope to address this problem by (1) implementing a graphical expression design tool [29] and (2) providing typical constraints to the user as templates to tailor (this was found to be useful in other situations [30]). Typical constraint types might include: time constraints (e.g., deadlines), budgetary limits (e.g., headcount, funds available), external factors (e.g., no trucking in Europe), specification of resources (e.g., types of processors/pre-fabricated servers in warehouse), etc.

Here is where we reap the benefit for using active software-agents to represent the to-do items. When the to-do agent detects the definition of a new formally defined constraint it spawns a sentinel agent, an autonomous piece of code, to monitor the constraint. The sentinel agent periodically checks for the validity of the constraints. When it detects the invalidation of the constraint it guards, the sentinel agent raises an exception. Depending on the constraint definition (by the user or template), the system will either handle the exception itself (e.g., using an exception handling routine/engine) or alert the user. Similar to personal schedulers, the users can choose how long before the actual invalidation of the constraints they want to be warned (e.g., 10 minutes before the expiration of the deadline, etc.).

Planning Options Based on Constraints

When the user specifies the goal, or post-condition, of an activity in his/her to do list (via the same mechanism used to define the constraints), the system will try to propose to

the user a series of possible approaches to completing his/her work. The system achieves this by taking the constraints on the activities provided by the user in the constraint-preservation sub-spectrum as well as the goal/post-condition and using them as a problem specification for an Artificial Intelligence planner (see [31] for an introduction). The planner will search for a way to achieve the goal while guarding all the constraints using activities that reside in a repository of possible actions (see below). In the best case it may find one or multiple plans. The user can then either choose a plan to follow or can decide that none of the plans is satisfactory. This would typically indicate that there is some constraint about which the system does not know. The user can choose to ignore the proposed solutions and act on his/her own or add the additional constraint (if he/she can formulate it in a machine-readable way) and retry the planner. In some cases the planner may not return a solution. This may either be due to an incomplete repository of possible actions or due to an under-specification of the goal. In this case the user can either choose to add more actions to the repository or just rely on the more limited support functionality of the constraint monitoring sub-spectrum.

Using the hiking analogy this approach parallels giving a hiker a trail map of the area and having him/her decide what trails he/she would like to take. Since the constraints are specified, the map also contains the ravines and mountains, such that the user might be able to decide that none of the proposed trails are feasible, and choose to take his/her own route. Consequently, in this sub-spectrum the system *plans tasks and resources to achieve goals* and lets the user decide which of the possible paths to take.

Marianne realizes that the system might help her to solve the problem of how to ship the server to Zurich in time. She therefore initiates the planner, which uses the constraints defined for monitoring (in the last sub-spectrum) and the goal specification (i.e. RT2000 delivered to Swiss Stock Exchange) as a problem specification. It proposes three shipping options (see Figure 5). First, it proposes to airfreight the server from Rotterdam. Second, it proposes to ship the server from the facility in Rotterdam using a train. And last, it suggests airfreighting the server from an American facility in Boston. Marianne did not consider this last option before, since deliveries to Europe typically come from Rotterdam. Given the looming deadline and the EU-trucker strike she decides to explore all three possibilities. She quickly discovers that given the strike she can't even find a truck to bring the server from the Rotterdam production facility out to the airport. Therefore the first option, shipping the server by plane from Rotterdam becomes implausible. To investigate the second possibility, using the train, she goes into the repository and looks at the train-shipment process. She realizes that since Switzerland is not part of the EU the train will have to clear customs at the Swiss border. During a phone call to the Swiss customs authority she learns that Swiss customs at the port of entry (for the train) is closed all day Sunday, which would delay the shipment by an additional 24 hours. Consequently, she chooses the only remaining option: shipping the server from the Boston.

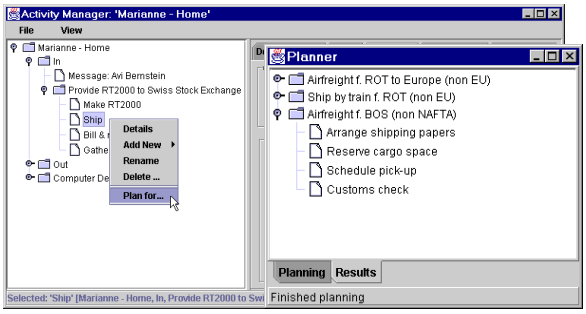

Figure 5: Planner

This part of the interpreter was implemented as a simple translator to an existing AI-planner [32]. The interpreting agent gathers all the constraints relevant for a to-do item, information about the current state of the process (as defined by the state of all the involved agents and data-structures) as well as a goal description (defined as a logical expression derived from the post-condition/goal of the process) and then passes it as a problem definition to the planner. In the scenario, for example, the agents gather the constraints like 'elapsed time < 48 hours' and 'no trucks', the goal description 'having an RT2000-server in Zurich', as well as the definition current state, including the knowledge that the EU-truckers are on strike, knowledge about Zing Computer's production facility and information about the current time.

The planner attempts to find a set of actions in the repository that will lead from the current state to the goal and pass the possible results to the interpreter, which translates them back to the process representation used within the system and presents them to the user. The repository contains a collection of possible actions, which are defined by their pre-/post-condition and a description of how the transformation from precondition to post-condition happens in detail.

In Marianne's situation, for example, the repository had to contain descriptions of all kinds of transport mechanisms and their properties. It thus had to have a description of trucking a good, including the property that it typically requires a truck (which are unavailable in our scenario), airplane-shipping (which was incomplete, since it didn't take into account the need for getting the good to the airport), as well as shipping by train.

Obviously, the quality of the planner's results is limited by two factors. First, the quality of the constraints entered (including the precision of the goal specification) has a major influence on the ability of the planner to prune its search-space. Since the users have entered them, the quality of the specification of those constraints is highly dependent on the abilities of the users. As mentioned above, though, I hope that the usage of expression design tools as well as the provision of tailorable typical expressions and expression tem-

plates provided by process specialists (e.g., residing in the repository) may alleviate this problem.

Second, the quality of the plans generated by the planner is dependent on the contents of the repository searched. As with any knowledge-based approach there is a bootstrapping problem in filling the repository with an adequate initial number of possible actions/processes. In most environments, however, a good part of those actions have already been formalized and defined in some system (e.g., WfMS, ERP). Furthermore, the repository records past cases as templates for future action. This 'case-based'-alike [33] approach can simplify some of the initial growing phase of the repository by limiting the enormous set-up costs.

Providing "Imperative" Scripts/Directions

System support in this last sub-spectrum can be likened to a traditional WfMS (see [6, 7, 9]). Since the process details are algorithmically well defined the to-do item software agent will direct each step leading to the result. Rather than guarding some constraints, the imperative plan avoids them through direction. Thus the *system directs the execution of tasks using resources to achieve goals*.

The boundary between the constraint-based planning sub-spectrum and the imperative sub-spectrum is crossed as soon as one of the results returned by the planner is chosen for enactment that is in an imperative form. The user can delegate the choice between the options to the system by defining a utility function. As an alternative to using the results of the planner the user can also directly browse the repository and compose a process manually [34], which can also result in an imperative script. The reverse transformation happens when the interpreter executing a task in the imperative sub-spectrum encounters an exception (which might be raised by a user!), stops its execution and runs the planner to find a number of alternatives to solve the current problem.

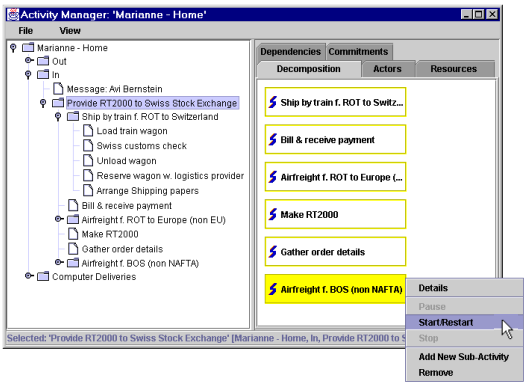

Figure 6: Starting a WfMS-like Script

Using the hiker's analogy again this sub-spectrum can be best compared to giving a hiker a specific set of directions. The directions are useful as long as he/she does not encounter a problem (e.g., an avalanche has cut off an existing path). As soon as a problem is encountered the hiker has to use a more situated method to finding his/her way to the goal (i.e., he/she has to drop the specificity of the process specification and use the support provided by the system in the other sub-spectra).

When Marianne chooses to airfreight the server from Boston by choosing to start that sub-process in her Activity Manager (Figure 6) the system starts the underlying WfMS-like shipment process of a new server from Boston to Zurich using airfreight in the last of the four sub-spectra. Assuming no new exceptions the system will direct the shipment just like a traditional order fulfillment system.

Division of Labor and Transfer Mappings in the Frontier

It is important to note that this system view relies on a co-operative understanding of the user system collaboration, where the system attempts to provide as much help as it can. *The more specific a task description is, the more the system can support the user* and relieve him/her of some part of the task. The less specific the task is, the more the user will have to do. Consequently the specificity of a process description guides the resulting type of division of labor between the human actor and the system.

Another important point is the system's capability to seamlessly integrate between the different spectra. The boundary between the context-provision and the monitoring sub-spectra is automatically crossed when some constraints are formalized in a machine-readable form. The next boundary is traversed when the system can find a series of paths from the current state to the goal (i.e., the planner can find an acceptable plan). Finally, the provision of some type of utility function by the user (either implicitly by choosing one of the options or explicitly by defining some sort of sort criteria between the options) helps the system to cross to the scripts sub-spectrum.

From the user's point of view, the transfers between the sub-spectra happen automatically as soon as the system can find the appropriate information. The user does not need to explicitly tell the system to cross the boundaries between the spectra. He/She does, however, need to enter the information (e.g., the constraint specification) that will prompt the system to cross the boundary.

Providing Structure for Situated Improvisation

The second requirement that the conceptual framework puts on process support system is the provision of a context for sensemaking and the articulation of next steps. As we have seen, the prototype system provides the user with ample contextual information (past activities in process context, documents related to this process, other actors involved, etc.) in order to understand the current state of the process. In some stages, however, he/she might not exactly know what to do next. The possible options of next actions can, for example, lie beyond his/her experience or an alternative, novel course of action is needed. Malone et. al. [35] have described how a repository of re-usable process components as well as past cases can be applied to organizational

processes and can be useful in a process-design and innovation setting. I therefore believe that a process repository containing process fragments and past cases can help users to articulate next steps and have included a repository, similar to the one presented by Malone et. al. [35], in my prototype.

Implementation Details

Process Models

The prototype system uses a process description, which is comparable to the one used by the MIT-process handbook (see [35] or Figure 7 for a meta-model). Activities are the central element of the model. Each activity can have an arbitrary number of resources in its context (e.g., for providing the links for the documents, which are related to a task). An activity can also have sub-activities (for functional decomposition) and sub-dependencies. Dependencies represent constraints between activities. In order to ensure the constraint represented by the dependency it needs to be coordinated by an activity. When two activities share a resource (i.e. a sharing dependency), for example, they can be coordinated using a first-come first-serve activity. Finally, activities, resources and dependencies can all have an arbitrary number of constraints defined on their attributes and parts. Furthermore, all elements can participate in a type of specialization hierarchy. This allows for a construction of an object-oriented-like hierarchy in the process- and case-repository. The main difference from traditional object-oriented inheritance is the possibility to 'disinherit' a feature from its parent. So when a person changes an inherited part of an activity it does not have loose its inheritance relations (see also [35], p. 427).

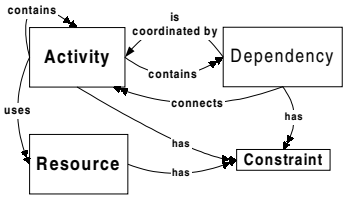

Figure 7: Process Model Parts

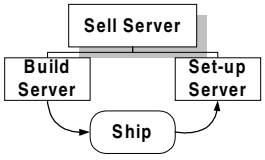

Figure 8: Example Process

System Architecture and Implementation

The overall system consists of five major logical components: a repository, a process-model interpreter, a planner, a user-interface, and an application-programming interface (API), which is used by other programs to interact with the system.

The *repository* stores all the process models, process fragments, and past cases. It furthermore contains references to all the resources (e.g., files) that are referenced by processes in the system and has some information about all the actors/users of the system. Distribution of the process data is accomplished through the services of an Object Request Broker (ORB). Each object in the repository is currently stored in a file, which transparently gets loaded when needed. Figure 8 shows the graphical representation for the "Sell Server"-process, as it is stored in the repository. It consists of three parts, the "Build Server" and "Set-up Server" processes as well as the "Ship" dependency.

The *interpreter* is implemented using a software-agent-oriented approach. Each active element of a process model is assigned to an agent. Collaborating with the other agents in the process model the software agent attempts to provide as much support as possible given the process specification. Thus for the "Sell Server"-process, a software-agent is going to be started for "Sell Server", "Build Server", "Set-Up Server", and "Ship". All those agents are going to interact using a speech-act-based protocol [36] to achieve the goal of the task. If the process specification falls within the context-provision spectrum, the agents ensure that all the resources referenced are accessible. When constraints get defined (i.e. in the constraint-preservation sub-spectrum) the agents start special sentinel-agents, which regularly check the consistency of the constraint. When a post-condition is specified the agents pass the process definition to the planner (see below). Finally, if the process model contains imperative features, they execute them analogous to a traditional WfMS while still checking on the constraints (to find exceptions). In all cases the agents maintain the relationships to other agents to which they have dependencies. This *integration of previously unconnected techniques provides the system its ability to support the enactment of processes that move along the specificity-frontier at run-time*. Consequently, it is the heart of this system's support for dynamic, rapidly changing organizational processes. Using the agent-based approach allowed me to build a dynamic interpreter, where local variation in process specificity and composition is handled by single agents and global changes are handled by the interplay between agents. This greatly reduced the complexity of the interpreter.

As a *planner* I used sensory graph-plan (SGP), a LISP-based research prototype presented by Weld et. al. [32]. The interpreter-agents translate the process model and the repository-content to a problem definition in the format understood by SGP. If the planner returns a result, then the interpreter-agents translate it back to the internal process-specification format.

In our scenario all parts of the process other then the "Ship"-dependency (Figure 8) were relatively well defined. So when Marianne initiated the planner (Figure 5) the interpreter collected all the constraints relevant to the problem (i.e. the constraints on "ship" directly, including the fact that it is in relation with both "build server" and "set-up-server").

The *user-interface* (see Figures 3-6) contains is a mixture of a traditional workflow-management work-list and a task-management user-interface (like the one presented in Kreifelts et. al. [28]) as well as a process model editor. It provides a direct-manipulation interface to all the major functions in the system like a browser for the process fragment/case repository, an activity-manager that provides a look at the activities a user is presumed to complete, a process-editor to change the tasks, and some additional maintenance editors.

The *API* provides a bi-directional interface between the prototype-system and external tools such as email, discussion databases, and on-line chat-programs.

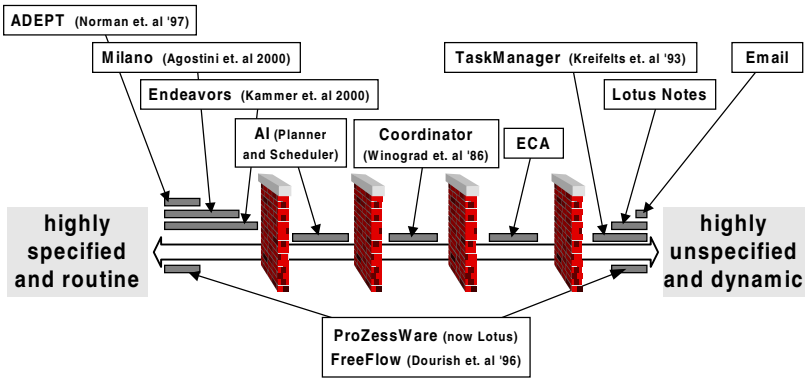

Figure 9: Related Work

EVALUATION AND LESSONS LEARNED

I have chosen three routes to evaluate the validity of the work presented. First, I chose to thoroughly ground my work in existing theory, previous work on requirements for supporting dynamic organizational processes, and some direct exchange with potential users, which provides me some assurance that the approach would be helpful in a practical setting. Second, I have implemented a proof-of-concept system and used it myself.

Finally, I am developing a number of detailed usage scenarios based on real-world occurrences and am evaluating how those scenarios would play out in different types of support systems: an email/Lotus-Notes-type system, a WfMS, and the prototype system presented. At the time of writing this analysis is underway; preliminary results support my assumptions about the advantages of a system basing on the specificity frontier, given its guidance in more routine tasks as well as flexibility where needed.

One interesting lesson learned was that the combination of previously unconnected approaches could lead to extremely useful solutions, just as the combination of messaging, database technology, security, and networking approaches led to a versatile tool like Lotus Notes. In my case it led to a system with the capability to support rapidly changing processes. However, I believe that this type of judicious integration could be extremely useful for many problems.

Another insight was that the usage of agent-oriented techniques allowed me to simplify the implementation of my multi-faceted prototype system (given its multiple sub-spectra) by avoiding code tangling, which complicates implementations. As Lopes [37] points out, code tangling typically happens when different concerns (or implementation issues like synchronization and information exchange) have to be addressed within the same piece of code. Using the agent-based approach I was able keep the complex parts of the implementation (for example, the code handling the change in specificity for different types of objects in my system) local to its effects and successfully avoid code tangling. This insight becomes increasingly important for CSCW-researchers, as the experimental systems we implement become more complicated and have to integrate more technologies (see previous insight).

RELATED WORK

As explained in the introduction the approach presented here is closely related to systems in the WfMS-tradition as well as the Groupware tradition. In the WfMS-domain a number of projects have tried to address the issues of adaptiveness and flexibility [27, 38-40]. However, all of those approaches aim at completely specifying the process before it is started using some formal method (e.g., Petri-nets) and adapting them when exceptions occur. They typically do not allow the execution of partially specified or abstractly specified process descriptions. On the other end of the frontier a number of CSCW-projects and Groupware tools have addressed the support for highly flexible processes.

The biggest problem of all those related projects, however, is the impermeability of processes across the specificity frontier. As can be seen in Figure 9, processes that get started in one category of support system are stuck in that type of support. Thus the support for an emergent process, for example, stays trapped in an ad-hoc system, even though its process structure may have emerged during a first part of its execution. Even systems basing on event-condition-action rules (ECA), which are typically used for constraint preservation or AI-planning systems, do not allow for mobility across the specificity frontier.

I know of three exceptions: ProZessware [41], Bramble [42] and FreeFlow [43]. ProZessware allows embedding Lotus-Notes Discussions into well-specified Workflows. However, these embedded discussions have to be pre-specified and the actual process structure is fixed. Bramble divides activities into well specified and unstructured. Similar to ProZessware it allows composing semi-structured activities from both well-specified and unstructured activities. In addition, it provides a rich mechanism for providing process context. Unlike the system presented here, though, it doesn't seem to allow for run-time transformations of activities from well specified to

unstructured and vice-versa. FreeFlow provides a mechanism to break the predefined constraints, which specify the Workflow. Once a constraint is broken, however, its guidance is lost for the process. Thus the system only allows a one-time reduction of specificity of a process description during run-time.

The work presented here is set apart from other projects by proposing a novel well-grounded approach to enabling the mobility of a process instance across the specificity frontier *during run time*.

CONTRIBUTIONS AND CONCULSIONS

The primary contribution of this research project is twofold. First, it suggests a novel approach to addressing the problem of support for dynamic organizational processes. The proposition of using varying specificity as an approach to solving the problem of supporting dynamic organizational processes is novel, non-obvious, promising, and supported by social science theory (see above). Second, the project showed the technical feasibility of this approach. Combining previously separate process-support technologies from well-specified and routine, to highly unspecified and dynamic, into a seamlessly integrated system that facilitates the mobility of processes across the specificity-frontier during run-time using a common process model is a non-trivial technical achievement.

Even though the primary focus of this project was not to empirically test the usefulness of the system, it provides some evidence to its plausibility. By developing detailed usage scenarios, based on empirical data, I have shown that it is at least plausible that a system like the one I have developed could be usable and useful. The preliminary results of the scenario analysis indicate that the variation of process specificity provides a useful approach to support dynamic organizational activity. It seems to reduce the overhead incurred by actors when attempting to adapt existing (running) processes to changing circumstances compared to traditional approaches.

For final proof, however, we will have to wait for a detailed empirical test of the usability and usefulness of a system like mine in a real world environment – a substantial research project in its own right.

AKNOWLEDGEMENTS

I would like to thank Tom Malone, Mark Klein and the other members of the MIT-CCS for their invaluable help, support and contribution to the ideas underlying this paper.

REFERENCES

1. R. C. Schank and R. P. Abelson, Scripts, Plans, Goals And Understanding: Lawrence Erlbaum Associates, 1977.
2. T. Winograd and F. Flores, Understanding Computers and Cognition: a New Foundation for Design. Norwood, N.J.: Ablex Pub. Corp., 1986.
3. L. A. Suchman, Plans and Situated Actions: The Problem of human-machine communication. Cambridge, UK: Cambridge University Press, 1987.
4. T. Winograd, "Categories, Disciplines, and Social Coordination," Computer Supported Cooperative Work, vol. 2, pp. pp. 191-197, 1994.
5. L. Suchman, "Do Categories Have Politics?," Computer Supported Cooperative Work, vol. 2, pp. pp. 177-190, 1994.
6. M. Hammer, W. G. Howe, V. J. Kruskal, and I. Wladawsky, "A Very High Level Language for Data Processing Applications," Communications of the ACM, vol. 20, pp. 832-840, 1977.
7. M. D. Zisman, "Office Automation: Revolution or Evolution?," Sloan Management Review, vol. 19, pp. 1-16, 1978.
8. C. Mohan, G. Alonso, R. Günthör, and M. Kamath, "Exotica: A Research Perpespective on Workflow management Systems," IEEE-Data Engineering, vol. 18, pp. pp. 19-26, 1995.
9. S. Jablonski and C. Bussler, Workflow management: modeling, concepts, architecture and implementation. London & Boston: International Thomson Computer Press, 1996.
10. L. Gasser, "The Integration of Computing and Routine Work," ACM Transactions on Information Systems, vol. 4, pp. 205-25, 1986.
11. E. M. Gerson and S. L. Star, "Analyzing due process in the workplace," ACM Transactions on Information Systems, vol. 4, pp. 257-70, 1986.
12. L. Suchman, "Supporting Articulation Work," in Computerization and controversy: value conflicts and social choices, R. Kling, Ed., 2nd ed. San Diego: Academic Press, 1996, pp. 407-423.
13. A. Newell and H. A. Simon, Human problem solving. Englewood Cliffs, N.J.,: Prentice-Hall, 1972.
14. P. G. W. Keen and M. S. Scott Morton, Decision support systems: an organizational perspective. Reading, Mass.: Addison-Wesley Pub. Co., 1978.
15. R. Rock, P. Ulrich, and F. H. Witt, Dienstleistungsrationalisierung im Umbruch - Wege in die Kommunikationswirtschaft, vol. 11, 1st ed. Opladen: Westdeutscher Verlag, 1990.
16. K. E. Weick, The social psychology of organizing, 2nd ed. Reading, Mass.: Addison-Wesley Pub. Co., 1979.
17. J. E. Bardram, "Plans as Situated Action: An Activity Theory Approach to Workflow Systems," presented at Fifth European Conference on Computer Supported Cooperative Work. Kluwer Academic Publishers, 1997.
18. A. Bernstein and C. P. Schucan, "Document and Process Transformation During the Product Life-Cycle," in

Information and Process Integration in Enterprises - Rethinking Documents, T. Wakayame, S. Kannapan, C. M. Khoong, S. Navathe, and J. Yates, Eds. Norwell, MA: Kluwer Academic Publishers, 1998.

19. W. J. Orlikowski, "Improvising Organizational Change Over Time: A Situated Change Perspective," Information Systems Research, vol. 7, pp. 63-92, 1996.
20. M. L. Markus and D. Robey, "Information Technology and Organizational Change: Causal Structure in Theory and Research," Management Science, vol. 34, pp. pp. 583-598, 1988.
21. H. Mintzberg and J. A. Waters, "Of Strategies, Deliberate and Emergent," Strategic Management Journal, vol. 6, pp. 257-272, 1985.
22. P. F. Berliner, Thinking in Jazz: The Infinite Art of Improvisation. Chicago, IL: University of Chicago, 1994.
23. K. Weick, "Improvisation as a Mindset for Organizational Analysis," Organization Science, vol. 9, pp. 543-555, 1998.
24. K. D. Swenson, "Visual support for reengineering work processes," presented at Conference on Organizational Computing Systems (COCS), 1993.
25. K. R. Abbott and S. K. Sarin, "Experiences with workflow management: issues for the next generation," presented at Conference on Computer Supported Cooperative Work, Chapel Hill United States, 1994.
26. C. A. Ellis and G. J. Nutt, "Workflow: The Process Spectrum," presented at NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions, University of Georgia, Athens, 1996.
27. P. J. Kammer, G. A. Bolcer, R. N. Taylor, and M. Bergman, "Techniques for Supporting Dynamic and Adaptive Workflow," Computer Supported Cooperative Work: The Journal of Collaborative Computing, vol. 9, pp. 269-292, 2000.
28. T. Kreifelts, E. Hinrichs, and G. Woetzel, "Sharing to-do lists with a distributed task manager," presented at Proceedings of the Third European Conference on Computer Supported Cooperative Work, (ECSCW), Milan, Italy, 1993.
29. A. Spoerri, "InfoCrystal: a visual tool for information retrieval management," presented at Second International Conference on Information and Knowledge Management, Washington, D.C., 1993.
30. A. MacLean, K. Carter, L. Lövstrand, and T. Moran, "User-tailorable Systems: Pressing the Issues with Buttons," presented at Human Factors in Computing Systems, Seattle, Washington, 1990.
31. D. S. Weld, "An introduction to least commitment planning," AI Magazine, vol. 15, pp. 27-61, 1994.
32. D. S. Weld, C. R. Anderson, and D. E. Smith, "Extending Graphplan to Handle Uncertainty & Sensing Actions," presented at National Conference on Artificial Intelligence, 1998.
33. J. L. Kolodner, R. L. Simpson, and K. Sycara-Cyranski, "A process Model of Case-Based Reasoning in Problem Solving," presented at International Joint Conference on Artificial Intelligence (IJCAI), 1985.
34. A. Bernstein, "The Product Workbench: An Environment for the Mass-Customization of Production-Processes," presented at Workshop on Information Technology and Systems (WITS), Helsinki, Finland, 1998.
35. T. W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell, "Tools for inventing organizations: Toward a handbook of organizational processes," Management Science, vol. 45, pp. 425-443, 1999.
36. J. R. Searle, Speech acts: an essay in the philosophy of language. London,: Cambridge U.P., 1969.
37. C. V. Lopes and G. Kiczales, "D: A Language Framework for Distributed Programming," Xerox Palo Alto Research Center, Palo Alo, CA, Technical Report SPL97-010, P9710047, February 1997 1997.
38. T. J. Norman, N. R. Jennings, P. Faratin, and E. H. Madami, "Designing and Implementing a Multi-Agent Architecture for Business Process Management," presented at Intelligent agents III: Agent Theories, Architectures, and Languages: IJCAI'96 Workshop (ATAL), Budapest, Hungary, 1996.
39. A. Agostini and G. De Michelis, "A Light Workflow Management System Using Simple Process Models," Computer Supported Cooperative Work: The Journal of Collaborative Computing, vol. 9, pp. 335-363, 2000.
40. C. Ellis and K. Keddara, "ML-DEWS: A Workflow Change Specification Model and Language," Computer Supported Cooperative Work: The Journal of Collaborative Computing, vol. 9, pp. 293-333, 2000.
41. ONEstone, "ProZessware for Lotus Domino/Notes," ONEstone Information Technologies GmbH, Paderborn, Germany, White Paper 1998 1998.
42. R. L. Blumenthal, "Supporting Unstructured Activities with a Meta-Contextual Protocol in Situation-Based Workflow," in Department of Computer Science. Boulder, CO: University of Clorado, 1998, pp. 173.
43. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw, "Freeflow: Mediating Between Representation and Action in Workflow Systems," presented at Computer Supported Cooperative Work, Boston, MA, 1996.