

Supporting knowledge-driven processes in a multiagent process management system

John Debenham
University of Technology, Sydney
debenham@it.uts.edu.au

Abstract. A knowledge-driven process is guided by its ‘process knowledge’ and ‘performance knowledge’. The goal of a knowledge-driven process may not be fixed and may mutate. Knowledge-driven processes are a characteristic of emergent processes. Emergent processes are business processes that are not predefined and are ad hoc. Process knowledge typically contains a substantial component of general knowledge and it is generally infeasible to build a system that represents it. A multiagent process management system provides substantial support for knowledge-driven processes. This system assists the users to manage these processes. The agents negotiate with each other to distribute the responsibility for managing sub-processes. Emergent processes may have goal-directed sub-processes if this is so then these sub-processes are completely managed by this system.

1. Introduction

Emergent processes are business processes that are not predefined and are ad hoc. These processes typically take place at the higher levels of organisations [1], and are distinct from production workflows [2]. Emergent processes are opportunistic in nature whereas production workflows are routine. How an emergent process will terminate may not be known until the process is well advanced. Further, the tasks involved in an emergent process are typically not predefined and emerge as the process develops. Those tasks may be carried out by collaborative groups as well as by individuals [3]. For example, in a manufacturing organisation an emergent process could be triggered by “lets consider introducing a new product line for the US market”.

From a process management perspective, emergent processes may contain “knowledge-driven” sub-processes as well as conventional “goal-driven” sub-processes. A *knowledge-driven process* is guided by its ‘process knowledge’ and ‘performance knowledge’. The goal of a knowledge-driven process may not be fixed and may mutate. On the other hand, the management of a *goal-driven process* is guided by its goal which is fixed. A multiagent system to manage the “goal-driven” processes is described in [4]. In that system each human user is assisted by an agent which is based on a generic three-layer, BDI hybrid agent architecture. The term *individual* refers to a user/agent pair. That system is extended here to support knowledge-driven processes and so to support emergent process management. The general business of managing knowledge-

driven processes is illustrated in Fig. 1, and will be discussed in Sec. 2. The following sections are principally a description of how the system in [4] has been extended to support the management of knowledge-driven processes. Sec. 3 discusses the management of the process knowledge. Sec. 4 describes the performance knowledge which is communicated between agents in contract net bids for work. Sec. 5 compares various strategies for evaluating these bids.

Process management is an established application area for multi-agent systems [5]. One valuable feature of process management as an application area is that ‘real’ experiments may be performed with the cooperation of local administrators [4]. The system described here has been trialed on emergent process management applications within university administration.

2. Process management

Emergent processes [1] are opportunistic in nature whereas production workflows [2] are routine. Emergent processes are inherently distributed and involve asynchronous work [6]. The tasks involved in an emergent process are typically not predefined and ‘emerge’ as the process develops [7].

Following [2] a *business process* is “a set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships”. Implicit in this definition is the idea that a process may be repeatedly decomposed into linked sub-processes until those sub-processes are “activities” which are atomic pieces of work. [viz (op.cit) “An *activity* is a description of a piece of work that forms one logical step within a process.”]. A particular process is called a (process) *instance*. An instance may require that certain things should be done; such things are called *tasks*. A *trigger* is an event that leads to the creation of an instance. The *goal* of an instance is a state that the instance is trying to achieve. The *termination condition* of an instance is a condition which if satisfied during the life of an instance causes that instance to be destroyed whether its goal has been achieved or not. The *patron* of an instance is the individual who is responsible for managing the life of that instance. At any time in a process instance’s life, the *history* of that instance is the sequence of prior sub-goals and the prior sequence of knowledge inputs to the instance. The history is “knowledge of all that has happened already”.

Three classes of business process are defined in terms of their management properties (ie in terms of how they may be managed).

- A *task-driven process* has a unique decomposition into a—possibly conditional—sequence of activities. Each of these activities has a goal and is associated with a task that “always” achieves this goal. Production workflows are typically task-driven processes.

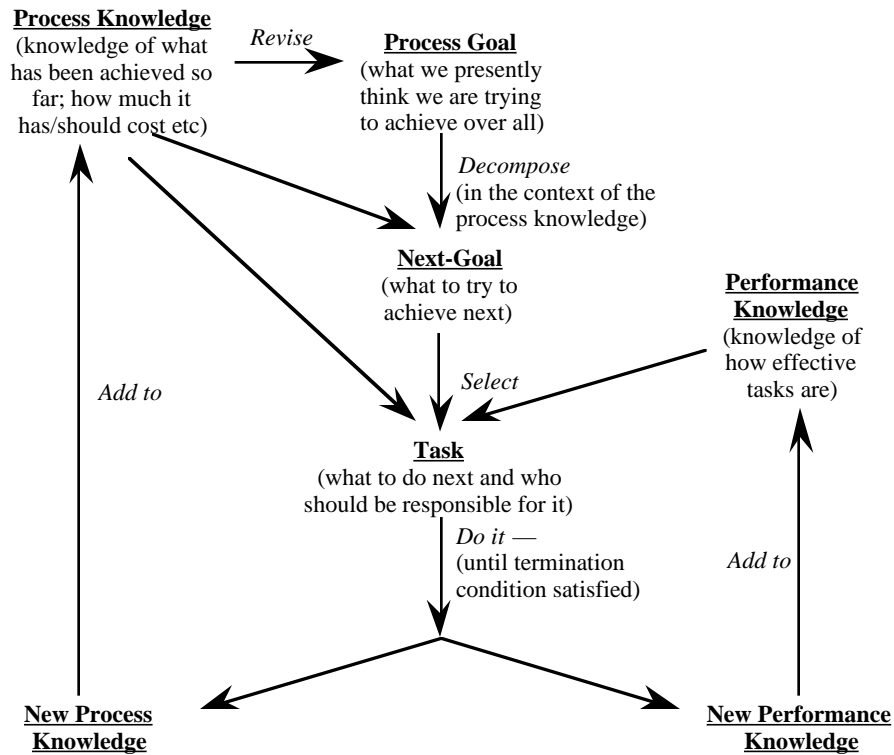


Fig. 1. Knowledge-driven process management (a simplified view)

- A *goal-driven process* has a process goal, and achievement of that goal is the termination condition for the process. The process goal may have various decompositions into sequences of sub-goals where these sub-goals are associated with (atomic) activities and so with tasks. Some of these sequences of tasks may work better than others, and there may be no way of knowing which is which [8]. A task for an activity may fail outright, or may be otherwise ineffective at achieving its goal. In other words, failure is a feature of goal-driven processes. If a task fails then another way to achieve the process goal may be sought.
- A *knowledge-driven process* may have a process goal, but the goal may be vague and may mutate [9]. Mutations are determined by the process patron, often in the light of knowledge generated during the process. At each stage in the performance of a knowledge-driven process the “next goal” is chosen by the process patron; this choice is made using general knowledge about the context of the process—called the *process knowledge*. The process patron also chooses the tasks to achieve that next goal; this choice may be made using general knowledge about the effectiveness

	Task-driven	Goal-driven	Knowledge-driven
Process goal	Determined by process patron, remains fixed	Determined by process patron, remains fixed	Determined by process patron, may mutate
Process termination condition	Process goal achieved	Process goal achieved	Determined by process patron
Next goal	Determined by instance history	Determined by instance history	Determined by process patron
Next task	Determined by instance history and next goal—should achieve next goal	Chosen (somehow) on the basis of instance history and next goal—may not achieve next goal	Chosen by process patron to generate process knowledge.
Next activity termination condition	Next goal achieved	Next goal achieved, if it fails then try another way	Determined by process patron

Fig 2. Properties of the three types of process

of tasks—called the *performance knowledge*. So in so far as the process goal gives direction to goal-driven—and task-driven—processes, the process knowledge gives direction to knowledge-driven processes. The management of knowledge-driven processes is considerably more complex than the other two classes of process. But, knowledge-driven processes are “not all bad”—they typically have goal-driven sub-processes which may be handled in conventional way. A simplified view of knowledge-driven process management is shown in Fig. 1.

Properties of the three classes of process are shown in Fig. 2.

Task-driven processes may be managed by a simple reactive agent architecture based on event-condition-action rules. Goal-driven processes may be modelled as state and activity charts [10] and managed by plans that can accommodate failure [11]. Such a planning system may provide the deliberative reasoning mechanism in a BDI agent architecture [11] and is used in a goal-driven process management system [4] where tasks are represented as plans for goal-driven processes. But the success of execution of a plan for a goal-driven process is not necessarily related to the achievement of its goal. One reason for this is that an instance may make progress outside the process management system—two players could go for lunch for example. So each plan for a goal-driven process should terminate with a check of whether its goal has been achieved.

Managing knowledge-driven processes is rather more difficult, see Fig. 1. The complete representation, never mind the maintenance, of the process knowledge would be an enormous job. But the capture of at least some of the knowledge generated during a process instance may not be difficult if the tasks chosen used virtual documents such as workspace technology, for example. The system assists with the presentation of the

new process knowledge to the patron when it can. Managing the process knowledge is discussed in Sec. 4. Some performance knowledge is not difficult to capture, represent and maintain. For example, measurements of how long an individual took to complete a sub-process can be very useful. So in the system described here, the process knowledge is left in the heads of the patron or nominated delegates, and the performance knowledge is captured by the system. The initial selection, and possible subsequent mutation, of the process goal is performed by the patron using the process knowledge, and so these actions are completely unsupported by the system, see Fig. 1. Task selection is partly supported by the system which can, for example, be given authority to form a committee, capture the expertise from that committee and forward that expertise to the patron in due course. In this way the system provides considerable assistance in the management of knowledge-driven processes. Further, if a now-goal is associated with a goal-driven, or task-driven, sub-process then the management system may be given full responsibility for the management of that sub-process as long as it contains the expertise to do so.

3. Process knowledge and the goals

This section refers to the left-hand side of Fig. 1, and to the relationship between the process knowledge, the process goal and the next-goal. This is the intractable part of knowledge-driven process management.

The process knowledge in any real application includes an enormous amount of general and common sense knowledge. For example, the process trigger “the time is right to look at the US market” may be based on a large quantity of current knowledge and a fund of experiential knowledge. So the system does not attempt to represent the process knowledge in any way; it is seen to be largely in the heads of the users. The system does assist in the maintenance of the process knowledge by ensuring that any virtual documents generated during an activity in a knowledge-driven sub-process are passed to the process patron when the activity is complete. Virtual documents are either interactive web documents or workspaces in the LiveNet workspace system [6] which is used to handle virtual meetings and discussions.

The system records, but does not attempt to understand the process goal. Any possible revisions the process goal are carried out by the patron without assistance from the system. Likewise the decomposition of the process goal to decide “what to do next”—the next-goal. It may appear that the system does not do very much at all! If the next-goal is the goal of a goal-driven process—which it may well be—then the system may be left to manage it as long as it has plans in its plan library to achieve that next-goal. If the system does not have plans to achieve such a goal then the user may be able to quickly assemble such a plan from existing components in the plan library. The organisation of the plan library is a free-form, hierarchic filing system designed completely by each user. Such a plan only specifies what has to be done at the host

agent. If a plan sends something to another agent with a sub-goal attached it is up to that other agent to design a plan to deal with that sub-goal. If the next-goal is the goal of a knowledge-driven process then the procedure illustrated in Fig. 1 commences at the level of that goal.

So for this part of the procedure, the agent provides assistance with updating the process knowledge, and if a next-goal is the goal of a goal-driven sub-process then the system will manage that sub-process, perhaps after being given a plan to do so.

4. Performance knowledge

This section refers to the right-hand side of Fig. 1. That is the representation and maintenance of the performance knowledge. The performance knowledge is used to support task selection—ie who does what—through inter-agent negotiation. So its role is a comparative one; it is not necessarily intended to have absolute currency. With this use in mind, the *performance knowledge* comprises performance statistics on the operation of the system down to a fine grain of detail. These performance statistics are proffered by an agent in bids for work. To evaluate a bid, the receiving agent defines payoff in terms of these statistics. In the case of a parameter, p , that can reasonably be assumed to be normally distributed, the estimate for the mean of p , μ_p , is revised on the basis of the i 'th observation ob_i to $\mu_{p_{new}} = (1 - \alpha) \times ob_i + \alpha \times \mu_{p_{old}}$ which, given a starting value $\mu_{p_{initial}}$, and some constant α , $0 < \alpha < 1$, approximates the

geometric mean $\frac{\sum_{i=1}^n \alpha^{i-1} \times ob_i}{\sum_{i=1}^n \alpha^{i-1}}$ where $i = 1$ is the most recent observation. In the

same way, an estimate for $\sqrt{\frac{2}{\pi}}$ times the standard deviation of p , σ_p , is revised on the basis of the i 'th observation ob_i to $\sigma_{p_{new}} = (1 - \alpha) \times |ob_i - \mu_{p_{old}}| + \alpha \times \sigma_{p_{old}}$ which, given a starting value $\sigma_{p_{initial}}$, and some constant α , $0 < \alpha < 1$, approximates

the geometric mean $\frac{\sum_{i=1}^n \alpha^{i-1} \times |ob_i - \mu_p|}{\sum_{i=1}^n \alpha^{i-1}}$. The constant α is chosen on the basis of

the stability of the observations. For example, if $\alpha = 0.85$ then “everything more than twenty trials ago” contributes less than 5% to the weighted mean; if $\alpha = 0.70$ then “everything more than ten trials ago” contributes less than 5% to the weighted mean, and if $\alpha = 0.50$ then “everything more than five trials ago” contributes less than 5% to the weighted mean.

Each individual agent/user pair maintains estimates for the three parameters: *time*, *cost* and *likelihood of success* for the execution of all of its plans, sub-plans and

activities. “All things being equal” these parameters are assumed to be normally distributed—the case when “all things are *not* equal” is considered below. Time is the total time taken to termination. Cost is the actual cost of the of resources allocated. For example, if a person has a virtual document in their in-tray then the time observation will be the total time that that document spent with that person, and the cost may derived from the time that the person actually spent working on that document. The likelihood of success observations are binary—ie “success” or “fail”—and so the likelihood of success parameter is binomially distributed, which is approximately normally distributed under the standard conditions. These three parameters are useful, but the *value* parameter—that is the value added to a process by a plan or individual—is at least as important.

Unfortunately, *value* is often very difficult to measure. For example in assessing the value of an appraisal for a bank loan, if the loan is granted then when it has matured its value may be measured, but if the loan is not granted then no conclusion may be drawn. The value of sub-processes are typically “less measurable” than this bank loan example. Although some progressive organisations employ experienced staff specifically to assess the value of the work of others. The existing system does not attempt to measure *value*; each individual represents the perceived *value* of each other individual’s work as a constant for that individual.

Finally, measurements of the *allocate* parameter for each individual are the amount of work w_i^j , allocated to individual j in discrete time period i . In a similar way to *time* and *cost*, the mean *allocate* estimate for individual j is made using $allocate_{new} = (1 - \alpha) \times w_i^j + \alpha \times allocate_{old}$, where w_i^j is the most recent observation for individual j . In this formula the weighting factor α is chosen on the basis of the number of individuals in the system, and the relationships between the length of the discrete time interval and the expected length of time to deal with the work. The *allocate* parameter does not represent workload. For example, if responsibility is delegated and then re-delegated, the *allocate* estimate of the first individual is not reduced. The *allocate* parameter is used by delegation strategies that address the density with which individuals are asked to do things. The *allocate* parameter is not normally distributed and the standard deviation is not estimated. The *allocate* and *value* estimates are associated with individuals. The *time*, *cost* and *likelihood of success* estimates are attached to plans.

The three parameters *time*, *cost* and *likelihood of success* are assumed to be normally distributed subject to “all things being equal”. If working conditions are reasonably stable then this assumption is acceptable, but the presence of external environmental influences may invalidate it. One virtue of the assumption of normality is that it provides a basis on which to query unexpected observations. Having made observation ob_{i+1} for parameter p , estimates for μ_p and σ_p are calculated. Then the next observation, ob_i , should lie in the confidence interval: $(\mu_p \pm \alpha \times \sigma_p)$ to some chosen degree of certainty. For example, this degree of certainty is 95% if

$\alpha = 1.645$. The set of observations $\{ob_i\}$ can progressively change without individual observations lying outside this confidence interval; for example, an individual may be gradually getting better at doing things. But if an observation lies outside this confidence interval then there is grounds, to the chosen degree of certainty, to ask why it is outside.

Inferred explanations of *why* an observation is outside expected limits may sometimes be extracted from observing the interactions with the users and other agents involved. For example, if Person X is unexpectedly slow in attending to a certain process instance then a simple interchange with X's agent may reveal that Person X will be working on the company's annual report for the next six days; this may be one reason for the unexpected observation. Inferred knowledge such as this gives *one possible cause* for the observed behaviour; so such knowledge enables us to *refine*, but *not to replace*, the historical estimates of parameters.

The measurement ob_i may lie outside the confidence interval for four types of reason:

- 1) there has been a permanent change in the environment or in the process management system—the measurement ob_i is now the expected value for μ_p —in which case the estimates $\mu_{p_{old}}$ and $\sigma_{p_{old}}$ should be re-initialised.
- 2) there has been a temporary change in the environment or in the process management system and the measurements $\{ob_i\}$ are expected to be perturbed in some way for some time—in which case the reason, Γ , for this expected perturbation should be sought. For example, a new member of staff may have been delegated the responsibility—temporarily—for this sub-process. Or, for example, a database component of the system may be behaving erratically.
- 3) there has been no change in the environment or in the process management system and the unexpected measurement ob_i is due to some feature γ that distinguishes the nature of this sub-process instance from those instances that were used to calculate $\mu_{p_{old}}$ and $\sigma_{p_{old}}$. In other words, what was thought to be a single sub-process type is really two or more different—but possibly related—process types. In which case a new process is created and the estimates $\mu_{p_{old}}$ and $\sigma_{p_{old}}$ are initialised for that process.
- 4) there has been no change in the environment or in the process management system and the nature of the most recent process instance is no different from previous instances—the unexpected measurement ob_i is due to—possibly combined—fluctuations in the performance of individuals or other systems.

In option 2) above the reason Γ is sometimes inferred by the system itself. This has been achieved in cases when a user appears to be preoccupied working on another task. If the reason Γ is to be taken into account then some forecast of the future effect of Γ is required. If such a forecast effect can be quantified—perhaps by simply asking a user—then the perturbed values of $\{ob_i\}$ are corrected to $\{ob_i | \Gamma\}$ otherwise the perturbed values are ignored.

5. Task Selection

This section concerns the selection of a task for a given now-goal as shown in the middle of Fig. 1. The selection of a plan to achieve a next goal typically involves deciding *what* to do and selecting *who* to ask to assist in doing it. The selection of what to do and who to do it can not be subdivided because one person may be good and one form of task and bad at others. So the “what” and the “who” are considered together. The system provides assistance in making this decision. Sec. 5 describes how performance knowledge is attached to each plan and sub-plan. For plans that involve one individual only this is done for instantiated plans. That is there are estimates for each individual and plan pair. In this way the system offers advice on choosing between individual A doing X and individual B doing Y. For plans that involve more than one individual this is done for abstract, uninstantiated plans only. This is something of a compromise but avoids the system attempting to do the impossible—for example, maintaining estimates on performance of every possible composition of committee. This does not weaken the system if a plan to form a committee is embedded in a plan that gives an individual the responsibility for forming that committee, because estimates are gathered for the performance of the second of these.

There are two basic modes in which the selection of “who” to ask is done. First the *authoritarian* mode in which an individual is told to do something. Second the *negotiation* mode in which individuals are asked to express an interest in doing something. This second mode is implemented using contract nets with focussed addressing [12] with inter-agent communication being performed in KQML [13]. When contact net bids are received the successful bidder has to be identified. So no matter which mode is used, a decision has to be made as to who to select. The use of a multi-agent system to manage processes expands the range of feasible strategies for delegation from the authoritarian strategies described above to strategies based on negotiation between individuals. Negotiation-based strategies that involves negotiation for each process instance are not feasible in manual systems for every day tasks due to the cost of negotiation. If the agents in an agent-based system are responsible for this negotiation then the cost of negotiation is may be negligible. A mechanism is described here to automate this negotiation.

If the agent making a bid to perform a task has a plan for achieving that task then the user may permit the agent to construct the bid automatically. As the bids consist of six meaningful quantities, the user may opt to construct the bid manually. A bid consists of the five pairs of real numbers (Constraint, Allocate, Success, Cost, Time). The pair *constraint* is an estimate of the earliest time that the individual could address the task—ie ignoring other non-urgent things to be done, and an estimate of the time that the individual would normally address the task if it “took its place in the in-tray”. The pairs Allocate, Success, Cost and Time are estimates of the mean and standard deviation of the corresponding parameters as described above. The receiving agent then:

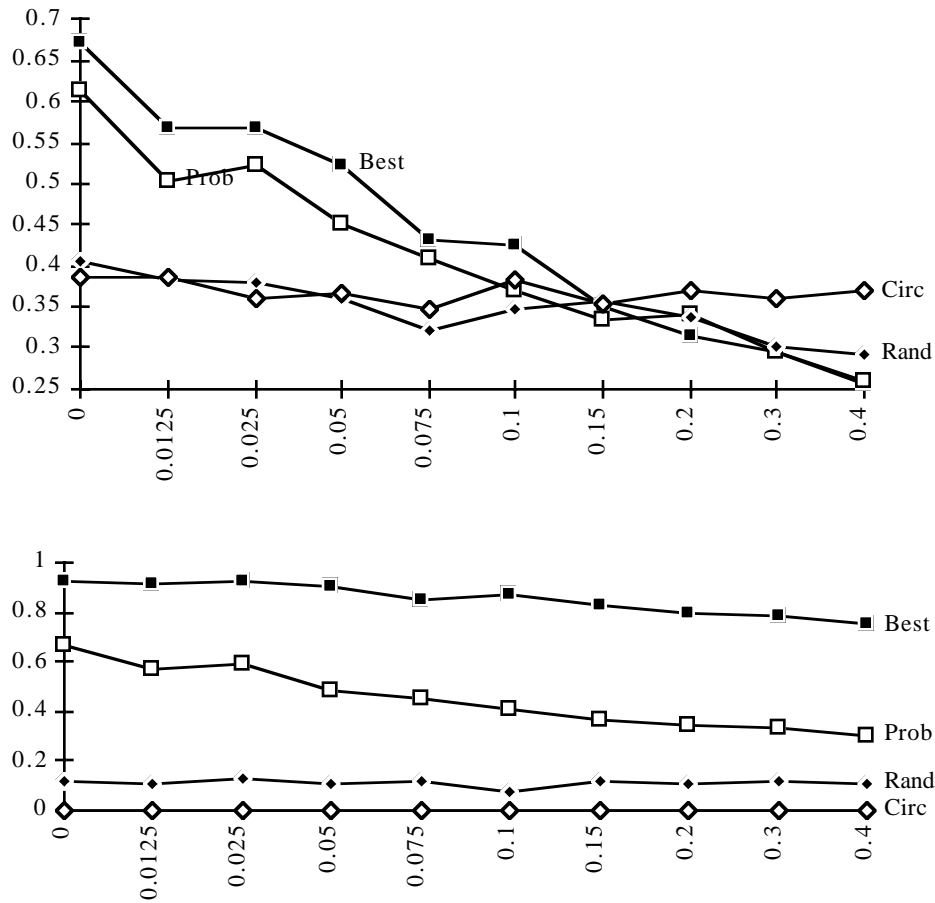


Fig 3. Payoff (top figure) and triple duplications (bottom figure) against the rebel factor for a learning rate = 0.1, death factor = 0.05, and $\alpha = 0.6$.

- attaches a subjective view of the *value* of the bidding individual;
 - assesses the extent to which a bid should be downgraded—or not considered at all—because it violates process constraints, and
 - selects an acceptable bid, if any, possibly by applying its ‘delegation strategy’.
- If there are no acceptable bids then the receiving agent “thinks again”.

5.1 The delegation strategy

A *delegation strategy* is a strategy for deciding who to give responsibility to for doing what. A user specifies the delegation strategy that is used by the user's agent to evaluate bids. In doing this the user has considerable flexibility first in defining payoff and second in specifying the strategy itself. Practical strategies in manual systems can be quite elementary; delegation is a job which some humans are not very good at. A delegation strategy may attempt to balance some of the three conflicting principles: maximising payoff, maximising opportunities for poor performers to improve and balancing workload. Payoff is defined by the user and could be some combination of the expected value added to the process, the expected time and/or cost to deal with the process, and the expected likelihood of the process leading to a satisfactory conclusion [14].

The system provides assistance to the user by suggesting how delegation could be performed using a method that the user has specified in terms of the tools described below. The user can opt to let the system delegate automatically, or can opt to delegate manually.

Given a sub-process, suppose that we have some expectation of the payoff D_i as a result of choosing the i 'th individual (ie agent and user pair) from the set of candidates $\{X_1, \dots, X_i, \dots, X_n\}$ to take responsibility for it. A *delegation strategy* at time τ is specified as $S = \{P_1, \dots, P_i, \dots, P_n\}$ where P_i is the probability of delegating responsibility at time τ for a given task to individual X_i chosen from $\{X_1, \dots, X_i, \dots, X_n\}$. The system suggests an individual/task pair stochastically using the delegation strategy.

Corporate culture may determine the delegation strategy. Four delegation strategies are described. If corporate culture is to choose the individual whose expected payoff is maximal then the delegation strategy *best* is:

$$P_i = \begin{cases} \frac{1}{m} & \text{if } X_i \text{ is such that } \Pr(X_i \gg) \text{ is maximal} \\ 0 & \text{otherwise} \end{cases}$$

where $\Pr(X_i \gg)$ means "the probability that X_i will have the highest payoff" and m is such that there are m individuals for whom $\Pr(X_i \gg)$ is maximal. In the absence of any other complications, the strategy *best* attempts to maximise expected payoff. Using this strategy, an individual who performs poorly may never get work. Another strategy *prob* also favours high payoff but gives all individuals a chance, sooner or later, and is defined by $P_i = \Pr(X_i \gg)$. The strategies *best* and *prob* have the feature of 'rewarding' quality work (ie. high payoff) with more work. If corporate culture dictates that individuals should be treated equally but at random then the delegation strategy *random* is $P_i = \frac{1}{n}$. If the corporate culture dictates that each task should be allocated to m individuals in strict rotation then the delegation strategy *circulate* is:

$$P_i = \begin{cases} 1 & \text{if this is the } i\text{'th trial and } i \equiv 0 \text{ (modulo } n) \\ 0 & \text{otherwise} \end{cases}$$

The strategies *random* and *circulate* attempt to balance workload and ignore expected payoff. The strategy *circulate* only has meaning in a fixed population, and so has limited use.

A practical strategy that attempts to balance maximising “expected payoff for the next delegation” with “improving available skills in the long term” could be constructed if there was a model for the expected improvement in skills—ie a model for the rate at which individuals learn. This is not considered here.

An *admissible* delegation strategy has the properties:

- if $\Pr(X_i \gg) > \Pr(X_j \gg)$ then $P_i > P_j$
- if $\Pr(X_i \gg) = \Pr(X_j \gg)$ then $P_i = P_j$
- $P_i > 0$ ($\forall i$)

So the three strategies *best*, *random* and *circulate* are *not* admissible. An admissible strategy will delegate more responsibility to individuals with a high probability of having the highest payoff than to individuals with a low probability. Also with an admissible strategy each individual considered has some chance of being given responsibility. The strategy *prob* is admissible and is used in the system described in [4]. It provides a balance between favouring individuals who perform well with giving occasional opportunities to poor performers to improve their performance. The strategy *prob* is *not* based on any model of process improvement and so it can *not* be claimed to be optimal in that sense. The user selects a strategy from the infinite variety of admissible strategies: $S = \delta \times \textit{best} + \epsilon \times \textit{prob} + \phi \times \textit{random} + \gamma \times \textit{circulate}$ will be admissible if $\delta, \epsilon, \phi, \gamma \in [0, 1]$, $\delta + \epsilon + \phi + \gamma = 1$ and if $\epsilon > 0$. This leads to the question of how to select a strategy. As *circulate* is only meaningful in stable populations it is not considered here.

There are three ways that an ‘optimal’ strategy could be identified. First, theoretically given a specification of what strategy should achieve. Second, by trial and error in a real experiment. Third, by a laboratory simulation experiments. The value of a theoretical derivation depends on the validity of the model on which the derivation is based. In simple cases this can be done, for example to achieve uniform allocation of responsibility. Real experiments to evaluate delegation strategies are just not viable. Laboratory simulation experiments are cheap and indicate of how the strategies perform.

A world is designed in which the relative performance of the four strategies *best*, *prob*, *random* and *circulate* are simulated. There are always three individuals in this world. If individuals die (ie they become unavailable) then they are replaced with new individuals. At each *cycle*—ie a discrete time unit—one delegation is made. There is a natural death rate of 5% for each individual for each cycle. The payoff of each individual commences at 0 and improves by 10% of “what there is still to learn” on each occasion that an individual is delegated responsibility. So an individual’s recorded payoff is progressively: 0, 0.1, 0.19, 0.271, 0.3439, and so on, tending to 1.0 in the

long term. The mean and standard deviation estimates of expected payoff are calculated as described above in Sec. 4 using a value of $\alpha = 0.6$. In addition the individuals have a strength of belief of the extent to which they are being given more work than the other two individuals in the experiment. This strength of belief is multiplied by a “rebel” factor and is added to the base death rate of 5%. So if work is repeatedly delegated to one individual then the probability

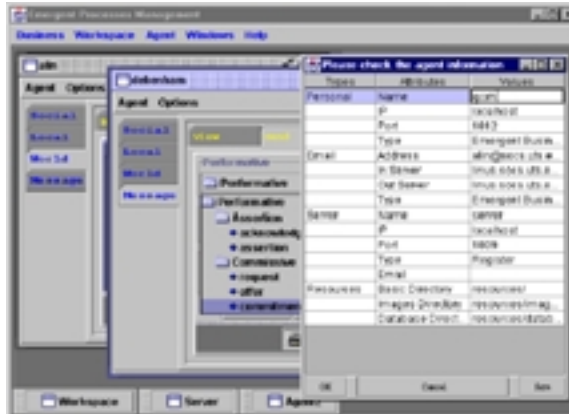


Fig 4. Setting up a task in the system

of that individual dying increases up to a limit of the rebel factor plus 5%. A *triple duplication* occurs when work is delegated to the same individual three cycles running. The proportion of triple duplications is used as a measure of the lack of perceived recent equity in the allocation of responsibility. The payoff and proportion of triple duplications for the four strategies are shown against the rebel factor on the top and bottom graphs respectively in Fig. 3. The simulation run for each value is 2000 cycles. The lack of smoothness of the graphs is partially due to the pseudo-random number generator used. When the rebel factor is 0.15—ie three times the natural death rate—all four strategies deliver approximately the same payoff. The two graphs indicate that the *prob* strategy does a reasonable job at maximising payoff while keeping triple duplications reasonably low for a rebel factor of < 0.15 . However, *prob* may only be used when the chosen definition of payoff is normally distributed. The strategy *best* also assumes normality; its definition may be changed to “such that the expected payoff is greatest” when payoff is not normal.

6. Conclusion

High-level business processes are analysed as being of three distinct types [15]. The management of knowledge-driven processes has been described. An existing multi-agent system for goal-driven process management [4] has been extended to support the management of knowledge-driven processes. The conceptual agent architecture is a three-layer BDI, hybrid architecture [16]. During a process instance the responsibility for sub-processes may be delegated. The system forms a view on who should be asked to do what at each step in a process. Each user defines payoff in some acceptable way. Payoff may be defined in terms of estimates of various parameters. These estimates are based on historic information; they are revised if they are not statistically stable. Using

three basic built-in strategies, the user then specifies a delegation strategy for the chosen definition of payoff. In this way the system may be permitted to handle sub-process delegation automatically. The system has been trialed on an application in a university administrative context. Three delegation strategies $[\delta = 0.5, \epsilon = 0.5, \phi = 0]$, *prob* and $[\delta = 0, \epsilon = 0.5, \phi = 0.5]$ represent varying degrees of the “aggressive pursuit of payoff” and have been declared “reasonable” in very limited trials.

References

1. Dourish, P. “Using Metalevel Techniques in a Flexible Toolkit for CSCW Applications.” ACM Transactions on Computer-Human Interaction, Vol. 5, No. 2, June, 1998, pp. 109—155.
2. Lawrence, P. “Workflow Handbook.” Workflow Management Coalition. John Wiley & Son Ltd, 1997.
3. A. P. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. C. Wileden, and A. L. Wolf. “Report from the NSF workshop on workflow and process automation in information systems.” SIGMOD Record, 25(4):55—67, December 1996.
4. Debenham, J.K. “A Multi-Agent System for Emergent Process Management”, in proceedings Nineteenth International Conference on Knowledge Based Systems and Applied Artificial Intelligence, ES’99: Applications and Innovations in Expert Systems VII, Cambridge UK, December 1999, pp51-62.
5. Jain, A.K., Aparicio, M. and Singh, M.P. “Agents for Process Coherence in Virtual Enterprises” in Communications of the ACM, Volume 42, No 3, March 1999, pp62—69.
6. Hawryszkiewicz, I.T. “Supporting Teams in Virtual Organisations.” In Proceedings Tenth International Conference, DEXA’99, Florence, September 1999.
7. Debenham, J.K. “Supporting Strategic Process”, in proceedings Fifth International Conference on The Practical Application of Intelligent Agents and Multi-Agents PAAM2000, Manchester UK, April 2000.
8. C. Bussler, S. Jablonski, and H. Schuster. “A new generation of workflow management systems: Beyond Taylorism with MOBILE.” SIGOIS Bulletin, 17(1):17—20, April 1996.
9. Debenham, J.K. “Knowledge Engineering: Unifying Knowledge Base and Database Design”, Springer-Verlag, 1998
10. Muth, P., Wodtke, D., Weissenfels, J., Kotz D.A. and Weikum, G. “From Centralized Workflow Specification to Distributed Workflow Execution.” In Journal of Intelligent Information Systems (JIIS), Kluwer Academic Publishers, Vol. 10, No. 2, 1998.
11. Rao, A.S. and Georgeff, M.P. “BDI Agents: From Theory to Practice”, in proceedings First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, pp 312—319.
12. Durfee, E.H.. “Distributed Problem Solving and Planning” in Weiss, G. (ed). Multi-Agent Systems. The MIT Press: Cambridge, MA.
13. Finin, F. Labrou, Y., and Mayfield, J. “KQML as an agent communication language.” In Jeff Bradshaw (Ed.) Software Agents. MIT Press (1997).
14. Koriche, F. “Approximate Reasoning about Combined Knowledge” in Intelligent Agents IV, Singh M.P, Rao, A. and Wooldridge, M.J. (Eds), Springer Verlag, 1998
15. Debenham, J.K. “Three Intelligent Architectures for Business Process Management”, in proceedings 12th International Conference on Software Engineering and Knowledge Engineering SEKE2000, Chicago, 6-8 July 2000.
16. Müller, J.P. “The Design of Intelligent Agents” Springer-Verlag.