

2. APPROACHES FOR MODELING HUMAN BEHAVIOR

This chapter describes a number of modeling approaches. It presents a review of existing relevant research literature to this thesis. The common theme in all the presented approaches is that of modeling human behavior and organization. It starts with a section on business process modeling. This approach is very relevant to this research, because it models work processes as sequences of tasks or flows of products in organizations. In the next section, the approach that is discussed is cognitive modeling. This approach is developed in cognitive science. It is relevant to this research, because we are interested in modeling individuals and groups of people, and their ability to act in and reason about their social and physical work context. The third section describes a distributed artificial intelligence (DAI) approach. Distributed AI focuses on multiple cognitive agents, and is thus relevant to the modeling of groups of people working together. In the fourth section, the last approach discussed is that of computational organization theory. This approach is relevant, because it also deals with modeling organizations at the agent-level.

I end this chapter with a conclusion section in which I relate all these modeling approaches to the different aspects of work practice mentioned in the first research question.

2.1 BUSINESS PROCESS MODELING

One of the most frequently used approaches to modeling human behavior in a business process is modeling the workflow through an organization. This modeling approach is often used in *business process reengineering* (BPR). In this chapter, I will describe workflow modeling, and some of its benefits, but more importantly I will discuss its weaknesses as they relate to modeling human behavior. I will use the SPARKS™ modeling and simulation tool as an example of workflow modeling. However, I argue that all of the issues related to SPARKS™ are systemic to workflow modeling in general.

Workflow modeling is a functional modeling paradigm that models the sequential tasks through which jobs⁶ flow in a business process. A workflow model typically describes the transformation of some sort of work product. In describing this transformation, workflow models focus on the time and cost parameters in each functional transformation—a task. Effectiveness is defined in terms of time and cost; that is, the number of jobs that can be processed in a specific time, and the overall cost of processing a job workflow modeling has gained significant popularity in the business world, due to the interest in business process re-engineering to gain more efficiency and cost reduction.

However, workflow models do not particularly focus on an individual's job performance, nor do they specify social and cultural behavior. Resources in workflow models are stochastic variables with specific characteristics, such as cost, work schedules, and other kind of parameters that can be measured. People are treated (modeled) as statistical resources, just like equipment and automation machines. As I will discuss later in this chapter, it is partly because of the limitations of such measurements that workflow models provide a very limited representation of how work is done in practice.

2.1.1 Modeling in business process re-engineering

The 1990s saw the development of many proprietary BPR methodologies. The idea for BPR was made popular in the business community by Hammer and Champy (Hammer and Champy 1993), and Davenport (Davenport 1993). The idea behind BPR is that work processes in many companies need to be evaluated, streamlined and automated in order to stay profitable and competitive (Scott Morton 1991). An example of a BPR methodology is BreakpointBPR™, developed by Coopers & Lybrand (Johansson 1992).

⁶ The term "job" is defined in section 2.1.2

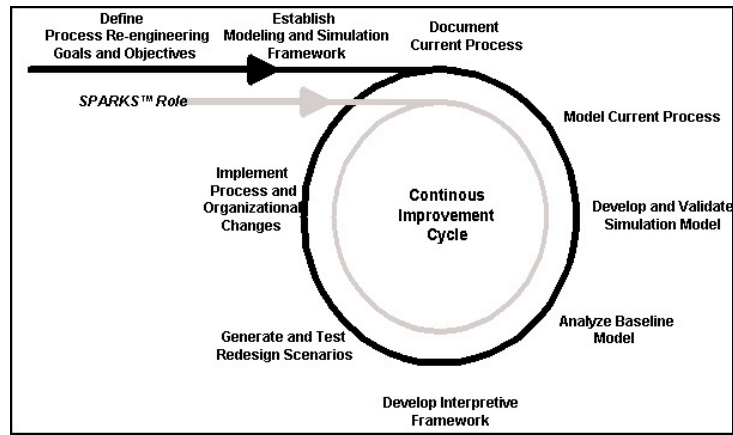


Figure 2-1. Life Cycle Model of Breakpoint BPR™ (borrowed from SPARKS™ training manual)

Figure 2-1 shows the life cycle model of the Breakpoint BPR™ methodology, and the role that a business process modeling and simulation tool plays in this life cycle. The Continuous Improvement Cycle in Figure 2-1 is primarily based on developing, analyzing, and testing a work process, using a workflow simulation tool. As changes to the business process are validated and tested through what-if scenarios in the simulation, the process moves to an implementation phase. Continuous improvement happens in the next cycle of modeling, where the current work process becomes the process that was just implemented. Detailing the methodology is outside the scope of this thesis, however the point to be made is that modeling the (current and new) work processes plays a central role in the life cycle of the methodology. Each step of the way, decisions about changing the business process are based on, and simulated in, a model. Modeling and simulation takes center stage. Design decisions are made based on statistical analysis of the efficiency (time and cost) of the simulated business process. The SPARKS™ training manual states: "SPARKS™ is [...] a computer-based tool for modeling, simulating and analyzing current business processes, and redesigning and implementing alternatives." (C&L 1994).

There are many uses of workflow modeling and simulation in a BPR project:

- As an analysis tool for the effectiveness of a business process, and as a way to identify opportunities for improvements.
- To describe the redesign of a business process, and to test and evaluate redesign alternatives. The argument is that, if we believe "the numbers" from the simulation are representative of how things work in the real world, we can use the model to choose between various alternatives.
- To analyze the impact of new technology and automation on a business process.
- To communicate, document, and train personnel about a business process.
- To manage and control a business process in real time, by implementing the model with workflow software technology.

2.1.2 Components of a workflow simulation model

Figure 2-2 shows the components of a workflow simulation model. The taskflow⁷ model represents the *sequential* tasks in a work process. The resource model shows the resources (people and artifacts) that are *performing* the task. In the input model one specifies *what* is "flowing" through the work process. This is the *work product* that is being worked on. Objects are flowing through the model as components of jobs. A *job* (e.g. an order) represents one or more objects that are worked on during the work process. The term *job* does not refer to the work of a person, as in "what is his job." Instead, a *job* is an abstract concept that

⁷ The concepts *taskflow* and *workflow* are used synonymously. However, I use *taskflow* when I speak of a static representation of a task sequence, and use *workflow* when I speak of a dynamic model that incorporates a taskflow, resource, input and timing model.

represents the work product being worked on. As such, it has discrete entry and exit criteria, and can be seen as the product of the business process. What people in the organization understand as “a job,” depends on their role and the tasks they perform in the organization. Some individuals in the organization view a job as a physical object, whereas for others it means something conceptual. For example, for people working on the job-floor of a manufacturing plant a job is the physical object that they are working on. However, for the people in the sales department, a job is a specific customer order represented by a customer order form.

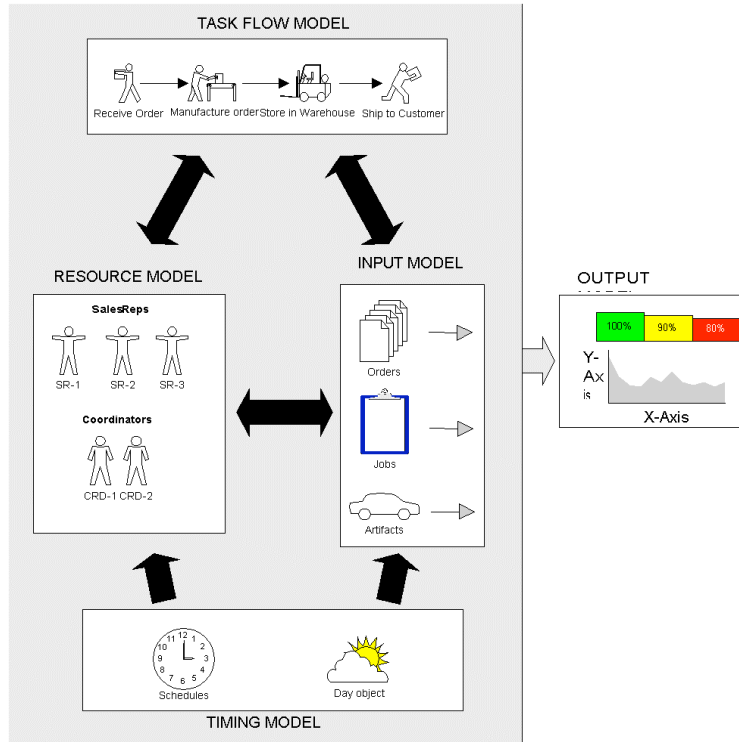


Figure 2-2. Components of a workflow model

A static model is a model that does not change with time, whereas a dynamic model changes over time through a simulation. In a static model the input, timing, resource, and taskflow models are not necessarily connected. To create a simulation, however, the models need to be interconnected. During a simulation, the input model provides the taskflow model with new jobs entering a process. The resource model provides the taskflow model with the resource units needed for each individual task. The timing model exists as part of a simulation, and provides the definition of the days and schedules used for resources, the times that jobs are entering the simulation, and the simulation clock keeping track of the time during the simulation. The output model specifies the *statistics* that are kept during the simulation and can be displayed.

2.1.2.1 Resource Model

The resource model defines the individual *resources* and groups of individual resources. Resources represent people or artifacts that perform work. Example resources include a clerk, a manager, a machinist, a machine, a drill, a computer, et cetera. The work performed by resources is not defined at the individual resource-level, or at the group-level. Instead, the work performed by individual resources is implicitly represented by the assignment of the number of resources used in a specific task in the taskflow model. Groups are only used to track statistics at the aggregate group level. When all the resources in a group are “being consumed” by tasks during a simulation run, the group has an in-box (queue) in which the work is kept until resources are available.

Individual resources in a workflow model do not exhibit individual behavior, neither task nor cognitive. Resources are statistical units assigned to tasks, which simply means that performing the task consumes the amount of resources specified at the task level.

2.1.2.2 Representing work as a taskflow

In a taskflow model work is represented in functional units, called *tasks*, through which jobs flow. Tasks represent the atomic steps of a process. Tasks are functions taking resources and jobs as input and calculating cost output, based on the time it takes to do the task and the cost of the resources for one job. Tasks are chained in a sequential flow, represented from left to right in task sequences. A number of special task-types enable representing work as a complex directed graph. In this section, I will describe some of the representational issues using workflow models.

2.1.2.2.1 Branches: Modeling different possible flows

A *branch* task is used to represent a decision point in a workflow. One specifies, usually in percentages, how many jobs will flow up one branch and how many flow down the other branch(es). *Joins* bring several branches of a flow back together into a single flow. Branch tasks and joins allow a modeler to represent the different ways a job can flow based on some attribute of the job. For example, the jobs (i.e. orders) flowing through the workflow represented in Figure 2-3 have an associated attribute representing the type of order form used for the job. In the branch task, this attribute is examined. Figure 2-3 shows the use of a branch and a join to represent an error condition.⁸

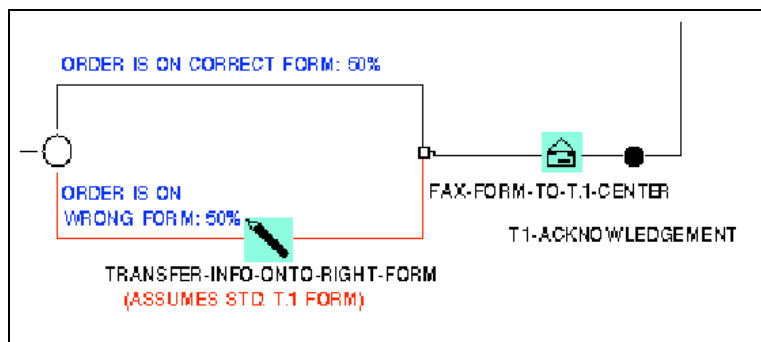


Figure 2-3. Modeling error conditions with a branch and a join task

The example shows an error condition in which the wrong form for an order is used; this happens in 50 percent of the cases. Thus, 50 percent of the jobs in the simulation will flow down through the task TRANSFER-INFO-ONTO-RIGHT-FORM. The two branches come together before the task FAX-FORM-TO-T.1-CENTER. From here on the jobs will again follow the same flow. Branches are “either-or” paths, and a job can only flow down one of the branches.

2.1.2.2.2 Spawns & Merges: Modeling simultaneous work performed by multiple people

Modelers often need to represent a situation where more than one person is working on the same job at the same time. Sometimes, this is done in isolation; for example, two people in different organizations have to perform tasks on the same job independent from each other. When people are working together at the same time modeling becomes more complicated, because their work needs to be synchronized. In workflow models, these work situations are modeled with a special kind of task, called a *spawn task*. Figure 2-4 shows the collaboration between the CO (central office) and the Field organization in troubleshooting and fixing a problem, by branching the workflow using the spawn task “co-helps-field.” The example shows the two separate tasks done by the two separate organizations at the same time. After the collaboration has been completed, the merge task “merge-shoot-trouble” recombines the spawned job.

⁸ This example, and the following examples, are from the T1 Radical Redesign model in SPARKS™, developed by Dave Torok at NYNEX Science & Technology, as part of the T1 Redesign project in 1992.

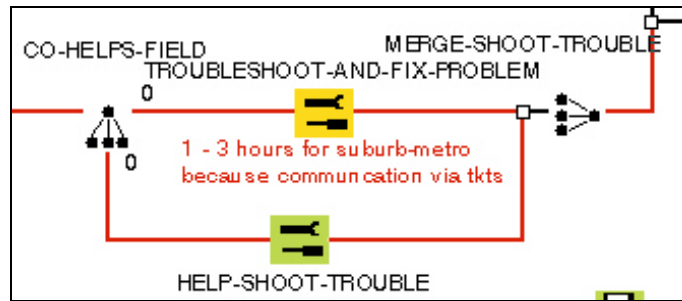


Figure 2-4. Modeling simultaneous work

A spawn task looks like a branch task, but is semantically different, because there is no decision whether the job should go one way or the other. Instead, the job is split—*spawned*—into two jobs, as if the job is flowing through both branches at the same time. Each branch of the spawn is used for representing the tasks of a different resource. At the end of the spawn there is a *merge task*. A merge task *merges* the split jobs together into the job from before the spawn. In this way, the simulation can keep track of the time spent by all resources for both workflows.

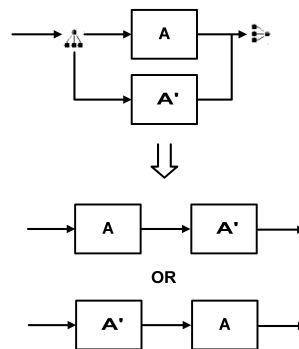


Figure 2-5. Spawn semantics

Note that during a simulation the tasks in the spawned flow are not necessarily performed at the same time. For example, in Figure 2-4 we might think that the tasks “troubleshoot-and-fix-problem” and “help-shoot-trouble” would be performed simultaneously. However, this depends on the availability of resources. Let us assume there are only two resources in the model. The first resource is assigned to the “troubleshoot-and-fix-problem.” The second resource is assigned to the “help-shoot-trouble” task. Let us also assume that at the moment a job is entering the spawn, the second resource is assigned to a task not shown in Figure 2-4. The second part of the spawn-job waits at the “help-shoot-trouble” task until the second resource is available. The second resource becomes available after its current task is completed, and the resource can start working on the “help-shoot-trouble” task. Consequently, the two tasks are performed in sequence (see Figure 2-5). This is not what is conceptually meant with the spawned flow in Figure 2-4. Of course, statistically the amount of time that was worked on the two tasks is correct, because the same amount of resources work on the tasks. However, the overall duration time for the job will now be longer than it should be, because the two tasks are done in sequence instead of in parallel. Remember that conceptually, the model is supposed to represent collaboration between two individuals. It would be better if the simulation showed that if one of the two resources, needed to accomplish the troubleshooting task, is not available, the task does not get accomplished. However, workflow simulations in Sparks™ do not give the modeler control over the assignment of resources to tasks, and availability of resources during a simulation. In short, in a Sparks™ workflow model there it is not guaranteed that two tasks are actually performed in parallel. There is no control over the execution of parallel tasks. This makes it hard to represent collaborative tasks between resources.

2.1.2.2.3 Modeling behavior with delays in the flow

Sometimes work cannot proceed because of some condition that needs to be met, but cannot be met at that moment. For example, when a sales-representative needs to call back a customer because he or she needs more information, and the customer does not answer, the job has to wait until the sales representative can talk to the customer. In workflow models, such a situation is modeled with a *delay task*.

While in a delay task a job is held and no resources are working on it—the resources are released to work on other tasks—while time is continuing. Figure 2-6 shows the example of a delay task, representing a sales rep waiting to call back a customer.

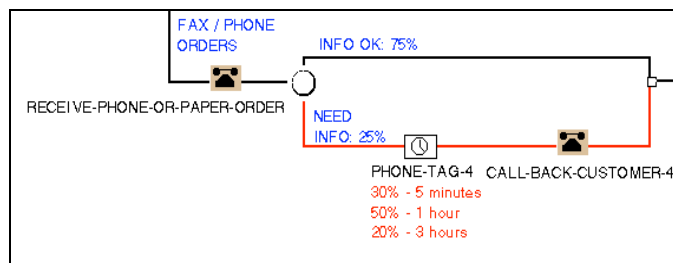


Figure 2-6. Modeling job delay

In order to model the task more accurately, the delay "phone-tag-4" in Figure 2-6 has several delay times associated with it: in 30 percent of the cases the delay takes 5 minutes, in 50 percent, 1 hour, and in 20 percent, 3 hours. Using this distribution of delay times, the model-builder intended to represent the different durations before the telephone call actually takes place. This representation of a phone call using delay tasks does not represent how phone calls happen in the real world. Phone calls are not planned tasks, but are situation dependent. Whether the sales-representative gets a hold of the customer depends on where the customer is and what the customer is doing. It will be very difficult to define a realistic distribution function for the time it takes for the sales-representative to get a hold of the customer, because it depends on the customer's situation. Also, the task of the resource being called (i.e. the customer) is not represented at all. It is not possible to associate the customer resource with the "call-back-customer-4" task, because resources are not tracked with the order, but with the task. During the simulation it is not known which customer resource should be associated with the callback task. This relates to the inherent problem of modeling collaborative tasks in a workflow model. Therefore, the model leaves out which customer is being called, and the time it takes for the customer to answer the phone call and talk to the representative is not taken into account.

Similar representational issues arise in a number of other cases; "off-task" behaviors, such as having coffee, or discussing the win in last night's basketball game (Sachs 1995). How do we represent the delay of a job, because the manager suddenly asks the sales-rep for a report on the monthly order figures? In tasks that are unrelated to the flow, the resources are not working on a specific job flowing through the process. These "off-task" behaviors cannot be modeled explicitly by a workflow representation, because the moment of occurrence is not known, and because such tasks do not "touch" the jobs flowing through the model. Work consists of many such off-task behaviors. Being interrupted in the task you are doing and resuming it after some time is part of every day work practice. We can model interruptions and resumes by adding delay tasks, but this is *not* how things happen in the real world. A delay task means that after the previous task there *is always* some delay. Interruptions, such as a phone call, or going to the bathroom, are serendipitous and cannot be modeled by delay-tasks. Not being able to represent this aspect of work means that the workflow simulation can never be an accurate reflection of the *work practice*—what people are actually doing.

2.1.2.3 Input Model

To simulate work using the workflow paradigm there need to be jobs "flowing" through the model. The number of jobs and the time intervals at which these jobs "enter" the model have to be defined up front, and monitored by the simulation engine. In SPARKS™ this is done by a special type of task, called a *start task*. Start tasks input jobs of a particular job-type into the task flow model. They specify the type of the job, the time intervals the jobs are entered into the task flow (job start times), the number of jobs that will be entered (job volume), and the distribution of the number of jobs over the time interval. *Job types* are used to define and categorize different jobs, representing different types of objects processed by the resources in the model. For instance, there might be different types of jobs in a customer service department, such as X-type jobs representing orders from customer X, being processed by the X-group, and Y-type jobs representing orders from customer Y, being processed by the Y-group. By separating jobs into types, the simulation engine can gather statistics for each of the job type.

2.1.2.3.1 Jobs are conceptual

A job is an abstract concept of “what is flowing” through the process. For example, an order form is faxed, after which the information is put in a database. The database information is used by the billing system, and is re-typed into that system. Abstracting different artifacts people are working on during the course of the process creates the notion of a job. The transformation from one artifact to another is what makes things “flow,” although it is arguable that it is not the artifacts that are “flowing,” but the information contained in the artifacts. Consider a piece of paper representing an order form with customer order information. This form is faxed to a department and there the clerk enters the information from the faxed form into the company's order tracking system. The form stops “flowing” through the process, but the order information, now in the computer system, keeps flowing through the process. In most workflow models, the artifacts that make up a job are not represented, and therefore, neither are the changes between artifacts, nor the flow of information. What is represented is a conceptual object representing the job. This limited representation of what constitutes a job makes it difficult to model the context of work. For example, in modeling a “hand-off” we would need to model what is being “handed-off”. The form of the artifact that is being handed-off plays an important role in how people do the work. For example, it makes a difference whether the order form is hand-delivered, or whether the information is sent through Lotus Notes™⁹. When hand-delivered, there could be a two-way conversation around the order. Whatever the impact of this conversation, the point is that this conversation is not represented. In other words, it is abstracted away from the actual work practice in the organization. By abstracting away the media (i.e. artifacts) of a work product, the context of the work is being left out, the information, or wrong information being passed down, is not represented, and neither are the actual work activities of the individuals in the organization. This is a severe limitation in using workflow models to represent how people actually work.

2.1.2.4 Output Model

There are two types of output from simulating a workflow model. The first is showing (using animation) jobs flowing through the workflow. The second, and most emphasized in workflow modeling, is the calculation of a wide range of statistics at various levels of granularity. There are two categories of statistics; statistics for the resources in the model and statistics for the workflow process itself. Most workflow simulation tools allow the end user to customize the output model.

Statistics about the workflow process are used to capture information about the jobs passing through the model. These statistics typically include, the number of jobs passing through the model, the time spent on the tasks, and the cost associated with the process as it is represented. Statistics also include detailed information for each task, such as: the number of jobs arrived at the task, the number of jobs finished by the task, and the actual and average task times during the simulation run.

Resource statistics are used to capture information about resource utilization and costs. There are statistics at the individual resource level, as well as at the accumulated group levels. Statistics of this kind include utilization of the resource, including percentages, availability of the resources throughout the simulation, and the work loads (i.e. backlogs) of the resources.

2.1.2.4.1 Issues with statistics

The statistical output of a workflow model is completely dependent on the resource and task data for the model. This phenomenon is known as the *garbage in, garbage out* phenomenon (Banks et al. 1996). Because a workflow model is so dependent on the data used to define the cost and time estimates for resources and tasks, this problem is real and almost unavoidable, as is illustrated by the statement of the model-builder of the T1-redesign model:

I must put forth a CAUTION statement here! BE SURE that you release statistics to the world wisely and with the caveats about the meaning of specific SPARKS data. There are so many assumptions built into the model that affect the numbers' applicability to the real world. The best use of the numbers is as a relative measure between one SPARKS model of the T.1 process and another model. However, there is more and more pressure to use the numbers in an absolute sense, so do so wisely. (Torok 1992)

⁹ Lotus Notes is a trademark of IBM Corporation.

The modeler found this warning so important that he repeated it twice in his document about the T.1 SPARKS™ model.

Because of the emphasis on the dynamics of jobs in workflow models, instead of the dynamics (i.e. behavior) of resources, and because of the limitation in modeling the way people and groups actually work, most *resource* statistics are, therefore, inherently inaccurate. Unfortunately, most of the time there is strong pressure from upper levels of management to show "head count" across models, and to map any kind of cost savings into head count reduction (Davenport 1995). One has to be careful in basing management decisions on simulation model statistics. Data and model verification and validation is one of the most important, but at the same time most difficult aspects of simulations (Banks et al. 1996) (Zeigler et al. 2000). The goal of the validation process is twofold: 1) produce a model that represents the actual system close enough so that it can be used as a substitute, and 2) develop a credible model so that it can be used with confidence in decision making.

I do not want to claim that statistics are completely useless. However, there are so many data related issues around the calculation of statistics that the use of a workflow simulation model in a process redesign project should always be looked upon with a dose of healthy skepticism.

2.1.3 Problems with workflow

"Work" consists of more than a functional transformation of work products by resources (Sachs 1995; Suchman 1987). One just has to think about the informal, circumstantial factors that influence work quality. Think about the effects of collaboration, types of hand-offs, geographical location of the people, the willingness of people, and sometimes organizations, to work or not work together ("engineering believes that they own the company"). All this is missing from a functional view of the work. In this section, I discuss some of the problems we have identified using workflow modeling and simulation in work redesign projects at the former NYNEX telephone company. I draw from our experiences using the SPARKS™ environment to model the current work process and the redesigned work process during two process redesign projects at the former NYNEX telephone company (the T1 re-design project (Corcoran 1992), and the BNA re-design project).

2.1.3.1 Data acquisition

It is very difficult to gather statistically significant and/or valid data for the development of a workflow model. This is true for the design of a future work process, but it is even true for the development of a workflow model of a *current* work process. Often, the work process organization does not keep historical data of the process. This means that the process analysts will have to gather the time and cost data for job-flow and resources. To do this correctly is not an easy task. When there is no data already available one should question why this is the case. Most of the time you will find that this is because it is not easy to gather this type of data. To do an extensive statistical data analysis of the work process might take longer than the life of the project itself. One approach is to do "spot observations" for jobs, tasks and resources. The data that one collects with this approach is, most likely, *not statistically valid*, meaning it is:

1. Not collected from the correct population,
2. Collected from incorrect or not representative observations,
3. Collected from an unreliable source.

2.1.3.2 Biases in simulation data

When analyzing simulation data, it is important to understand the types of biases that may be incorporated in the data from a simulation run (Banks et al. 1996). The two biases that are apparent are the "snap shot" bias and the "average vs. bucket" bias.

The *snapshot bias* occurs when, during the simulation, the data is examined at a "snapshot" in time. The bias occurs when comparing data between two measurement points in the workflow model, such as looking

at the difference in average elapsed time. The data is biased, because at any given time the data measured at the latter point in the flow does not include the data from jobs that are, at that moment, in between the first and the latter point, and therefore have already past the first measured point, but not the second. In other words, the data measured at the second point is "lighter" than the data measured at the first point. This bias increases when either the elapsed time between the points increases, or the number of jobs measured decreases. Therefore, to gain truly accurate difference measures, the simulation should be run for a large number of jobs.

The *average vs. bucket bias* occurs when the averaged data does not represent the actual distribution at a given point. An excellent example of this is shown in the T.1 model for the average elapsed time at the completion of the "order-negotiation" process. The average elapsed time was approximately five hours. However, a look at the distribution of elapsed times showed 1378 of 1679 orders took under 30 minutes, and 301 orders took 24 hours or longer, with absolutely no orders completed between 30 minutes and 24 hours! This type of bias can occur with any average measure (cost, time, etc.) and is highly dependent on the actual work practice that is modeled. This bias can also occur when mixing jobs of different types.

2.1.3.3 Resource issues

The workload in a workflow simulation is dependent on the resources available to do the work. For a workflow simulator, resources are all the same. The only use of a resource is to balance the workload; i.e. the more resources that can be chosen from for a particular job in a task, the more jobs can be done in parallel. The type of resource has no influence on the time it takes to finish a task—some tools, such as SPARKS™ try to model personal, social and cognitive factors of resources as numerical measures. However, these measures are very subjective and unscientific, and can do more harm than good (see 2.1.3.3.3). The use of types of resources in a model is therefore mostly for the gathering of cumulative statistics at the group level. For example, the cost of one type of resource is higher than others.

2.1.3.3.1 Motivation and culture

We all know intuitively that experience levels of individuals, as well as culture and work ethics, have impact on the work process. In addition, we know intuitively that some groups are more adaptable to change than others. So, what happens to the performance of groups when their work is changed? More importantly, can we design a new work process with these kinds of factors in mind? These are important issues in the redesign of a work process. Workflow models do not include these types of factors. The business community starts to realize that the *intellectual capital* of a company consists of the people, and their work practices (Stewart 1997) (Nonaka and Takeuchi 1995). Modeling and simulating the work of an organization without taking the capabilities (cognitive, social, and cultural) of the individuals into account leads to work process designs that most likely fail to produce the expected results after implementation. In short, the change from the current to the future situation impacts the work at a deeper level that cannot be predicted by a model of the future work system.

2.1.3.3.2 Resource approximations

One dilemma in modeling the resources in a workflow model is the dilemma of how accurate to model the resources of an organization. The tasks modeled in a taskflow model are usually just part of the total work of an organization. We referred to the work *not* modeled in a workflow model as "off-task" behaviors. For example, the taskflow of the T1 model only represented a part of all the jobs that were being worked on by the different organizations in the model. Therefore, if the resources would be modeled very accurately (i.e. the number of people working, etc), there would always be enough resources to handle the job-load in the simulation model. So, how many resources is a realistic measure? The modeler for the T1 model "compressed" the resource model to an approximate number of resources needed to work "full time" on T1 provisioning jobs. Again, these kinds of approximations make the output of a workflow simulation less valid.

2.1.3.3.3 Modeling social, cultural and cognitive factors

The way people work is very much defined by factors other than time and money. For one thing, people build relationships in the workplace that influence the way they work. (Suchman 1987) (Wenger 1997). The point is that work is not just the flow of jobs through the process, or the flow of information between people and systems. Work is about people, their habits, their norms and relationships, their physical environment, et cetera. A new design of a work process that is going to be a success when implemented has to consider these factors. Efficiency in the workflow does not necessarily come by removing some redundant tasks.

Sure, it might help, but maybe the tasks are there for a good reason, and taking them out might actually hurt the process.

Some modeling tools try to model social, cultural and cognitive factors using quantitative measures. For example, in SPARKS™ we can model the “diligence” of a resource by specifying in seconds how long the resource will work on a task *after* the time model specifies that it stops working for the day. This way, a resource that just started a task that takes thirty minutes, at five minutes to five, and who will normally stop working at five o'clock and has a diligence factor of 30 minutes, will finish the task before stopping. Although this may sound like a good way to model the flexibility of people, it is not a very realistic way of modeling flexibility. It means, for instance, that if the task would take 31 minutes the resource would not be diligent enough and stop. How does someone know exactly, to the minute, how long a task will take? This is not realistic and not how people do work, even if they have to “punch the clock.”

2.1.3.3.4 Multi-tasking: interrupt and resume

Work consists of interruptions, switching to and from different activities; picking up a phone while checking the status of a job, answering someone's question while entering data in the order system, et cetera. It is hardly ever that we are engaged in just one task at a time. People are masters in “juggling” many tasks at once. However, representing the multi-tasking of individual resources in a workflow model is not possible. Resources cannot be controlled at an individual level. The workflow simulator schedules the resources. Which resource does what and when depends on the number of resources available and the workload (the number of jobs flowing through the simulation), and not on a model of the cognitive behavior of the resource.

2.1.3.4 The Turf Coordinator problem

The TC role was briefly mentioned in the introduction chapter, and is described in more detail in Sachs (Sachs 1995). The problem of representing the work associated with this new role in a workflow model was one of the driving forces to start our research on a new modeling paradigm for simulating work practice. In this section, we describe the problem in more detail.

In the radical new design of the T1 process, the design team came up with a new concept called a “turf.” The city of Manhattan was divided in a number of geographical turfs. Each turf has their own field force that covers that specific geographical area and the central offices in that area. In the redesign model, the T1 jobs are assigned to a specific turf. Consequently, the field force is now dedicated to a turf. In the model, this meant that jobs, as well as the tasks and resources had to be duplicated, making the model more complex and more difficult to maintain¹⁰. The next big change in the design was the introduction of the TC role. For each turf, there is a TC. A TC has end-to-end responsibility for a job. The work associated with this role is not what a business manager would call “value added” work. The tasks of a TC are not directly related to the flow of jobs through the work process. His or her tasks do not transform the job, but are mostly related to the activity of coordinating conversations between the field and the central office. Therefore, most of the tasks of the TC have no place in a workflow model. This is best explained with the example of coordinating the testing of a T.1 circuit.

2.1.3.4.1 Coordinating a T.1 circuit test

One of the identified problems in the old (current) T1 work process was the coordination of testing circuits between the central office and the field (the customer premise). Before the concept of a turf and the TC role, nobody in the process was specifically assigned to a job. The Trouble Ticketing System (TTS) controlled the whole work process. The TTS was designed to eliminate what management felt were “off-task” conversations between workers. The TTS was setup with the assumption that *any* worker can perform a specific task for a job, and that the installation, testing and completion of a job did not have to be done by one team of workers. It was believed that it is more efficient to break the work up and have people work on one specific task in a job. This way, orders could be handled as they come up, and people wouldn't be caught in time-consuming troubleshooting. The process worked by sending and receiving “tickets.” Each ticket would be dispatched by a central dispatch system. Every time someone had performed a task from a ticket, the ticket was sent back to the central dispatch system, which would then send it to another person,

¹⁰ In SPARKS™ each task can only be worked on by one resource, selected from one group. Therefore, modeling seven turfs entailed creating seven identical process flows. Just as in computer programming, duplication of “code” creates a maintenance overhead.

leaving the first person to pick up a new ticket. In practice, this meant that when a worker encountered a problem (like not hearing a dial-tone) (s)he would send a trouble-ticket into TTS, and assume that someone else would pick up this trouble and fix it. The person who sent the ticket in the first place would then have time to pick up the next ticket and start working on a new job. From the worker's perspective, the electronic dispatching got in the way of being able to solve problems collaboratively. Unlike problem-solving conversations in which two people can discuss a problem and explore the possibilities to solve it, the TTS transforms a problem-solving conversation into a number of sequential tickets picked up by an array of workers, who do not speak to one another.

The translation of each turn of talk into a single ticket reduced an effective network of co-workers who could troubleshoot together into something like a relay-race, handing-off pieces of work to the next runner, creating an aggregate of dissociated workers. It changed a troubleshooting conversation into a series of solitary commands disembodied from context. Not only was the conversation—the story line of the problem, if you will—lost, so was the work community. (No one knew who else was working on the job.) (Sachs 1995)

Therefore, whenever the technician, installing the T1 line in the field (at the customer premise), had to run a test to verify the complete circuit from end to end, (s)he had to send a trouble-ticket to the central office to request the test. The tester who would pick-up the ticket would setup the circuit so that the test could be run. There was no direct contact between the tester, and the field technician. When the test would go wrong, (s)he would have to go through the whole process again, while the second time the ticket might be picked up by a different tester.

In the new design, the TC is an experienced technician whose role is to coordinate the telephone communications between, up to five, field technicians installing and testing a circuit. The problems that can occur when installing and testing a circuit are unpredictable and of a wide range. The TC needs to know whom to contact, and where people are geographically located in order to coordinate getting the right people involved. The TC, field technicians, and turf managers meet regularly to identify recurring problems and solve them.

2.1.3.4.2 Modeling the turf coordinator

The difficulty with modeling the TC in a workflow model has to do with the fact that the work activities of a TC are not necessarily identifiable with specific jobs flowing through the process. True, when a TC coordinates communication between several people in the field due to a problem that occurred, (s)he is working on a job, and the task can be seen as such. However, *when* this happens is unpredictable and cannot be pre-specified in a task sequence. Furthermore, representing a three or more person phone conference is not easy, if not impossible, in a workflow model. Representing the coordination of such a conference call is so situated that it is even hard to think about pre-specifying the steps that are involved. As mentioned before, in a workflow model we cannot specify a specific resource working on a specific job. Therefore, the only way such a conference call would be possible is if there are enough resources available at that moment in the simulation to work on that specific job. The chances that the right resource, from the right location, performs the correct task in the task sequence, is next to null. Modeling a phone conference call as a pre-specified sequence of tasks is impossible, since it is unpredictable how such collaboration might play out over time.

All this resulted in the fact that the TC's work hardly showed up in the SPARKS™ model of the radical new design. Because of that, it was hard for the design team to justify a full-time person playing the role of a TC. The resource statistics of the model did not justify a full-time person. Nonetheless, the design team knew that the importance of coordinating the technicians was one of the major factors in the time and cost savings of the new design. Although management was very impressed with the time and cost savings of the model, the team had a hard time convincing management of the need for a TC.

This experience led us to start working on a modeling paradigm and modeling tool that would allow the representation of coordination activities, such as the TC. Work needs to be represented based on the activities that individuals engage in, and the flexibility of people's decisions to change the sequence of these activities. Collaboration cannot be pre-specified, but is emergent. The next section discusses the main

difference between the emergence of behavior versus the pre-specification of the sequence of tasks, possible error situation, and the collaboration between individuals in a work process.

2.1.3.5 Emergent behavior vs. static taskflows¹¹

As I have indicated, workflow models typically describe only idealized tasks in transformation of a work product. In the following illustration (Figure 2-7), for example, an engineer receives an order form from a representative, assigns a circuit loop using a computer tool; later the representative enters more data about the order. Figure 2-7 presents an excerpt of a SPARKS™ model for Business Network Architecture (BNA) order processing¹².

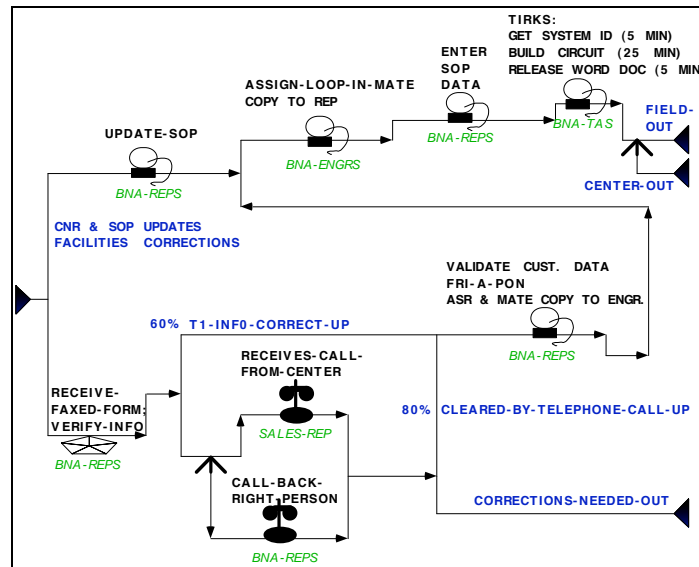


Figure 2-7. Order processing in the business network architecture (BNA) organization, showing flow of orders from left to right and conditional branching (indicated by arrows hitting a vertical bar). Top section shows updates by representatives (BNA-reps) and engineers for customer-not-ready (CNR) and other revisions to orders. Lower section shows standard process for handling faxed order from sales center, followed by correcting 40% with missing or invalid information by calling the sales representative. After validating customer data (center right), orders are handled by circuit allocation process (top). Other acronyms (e.g., SOP) are internal databases.

A critique of this diagram from the perspective of situated action (Suchman 1987) would inquire why order processing occurs this way and how it might be improved. Perhaps surprisingly, the figure leaves out what a problem-solving (cognitive task) model would typically focus upon. For example, what information does the engineer read from the order form and what deductions are required in order to assign the circuit? This particular model leaves out *how orders are planned and assigned*, *multi-tasking* (the fact that a rep or engineer works on several jobs at once before completing them) and *how people interrupt and resume their work*. A cognitive model of the same business process might consider some of these factors, but would leave out how people come to be synchronized in a phone conversation, how an engineer might help a representative do his job, and broader considerations of how a representative actually spends her day. In particular, because interpreting and executing orders can be problematic in unexpected ways, people need to improvise in ways that work system designers might not have anticipated:

Information flow charts show "information" moving in little blocks or triangles moving along arrows to encounter specific transformations and directions along the diagram. In reality, it seems, all along the arrows as well as at the nodes, that there are people helping this block to be what it needs to be—to name it, to put it under the heading where it will be seen as a recognizable variant, deciding whether to leave it in or take it out, whom to convey it to. (Wynn 1991)

¹¹ This section is adapted from Clancey, W. J., Sachs, P., Sierhuis, M., and van Hoof, R. (1998). "Brahms: Simulating practice for work systems design." *International Journal on Human-Computer Studies*, 49:831-865.

¹² The BNA project was another business process redesign project within the, by then, Bell Atlantic company.

Wynn's complaint might be viewed as an issue of modeling granularity—she is asking for more details. But her broader issue is how people think about work and how they solve problems cannot be reduced to information processing tasks and reasoning. Additional concepts are required.

To analyze the example more precisely, consider what the various branches and joins mean:

- Proportional mix of different kinds of orders, customers, or services (e.g., the first branch indicates how the order comes into the organization, as an update/correction or as an initial faxed order).
- Hand-off to the next (possibly dependent) step in a functional sequence (especially clear in the modeler's use of a step notation in the top portion of Figure 2-7).
- Condition of job being processed (e.g., incorrect information, indicated by a branch showing 60% correct and 40% requiring troubleshooting).
- Events that occur during troubleshooting (e.g., receive-a-call or keep-calling-back).

Although this abstraction is useful, notice that everything in this diagram was specified and connected by the modeler. The model essentially leaves out the logistics, *how these conditions come to be detected and resolved*, such that work and information actually flows. What is wanted is a model that includes aspects of reasoning found in an information-processing model, plus aspects of geography, agent movement, and physical changes to the environment found in a multiagent simulation (Tokoro 1996). The designed flow of Figure 2-7 assumes that people are always on the spot, picking up faxes and handing them over to others, reviewing the status of database entries on-line, responding to phone calls, et cetera. In reality such behavior is an emergent property of the situation; people come together and their work is constrained by the environment. A designed work and information flow diagram leaves out the accomplishment of *synchronization* and the effect of *juxtaposition* of materials, such as the following:

- Parallel-dependent processing (e.g., in practice, people start a time-consuming step in processing order before approval/clearance for doing the work (Dourish 1996))
- Cognitive interpretation, social knowledge, and variability (e.g., how do people know that information is not correct? How are intra-group variations in how the work is done a resource for handling difficult situations, such as the unavailability of a computer system?)
- Interactional logistics and daily activities (e.g., the steps marked "Receives call" and "Call-back-right person" in Figure 2-7 omit the activity of "making the call" in the first place and when it occurs during the day. Is a pager and cellular phone used or voice mail at a desk phone?)
- Informal help and "keeping an eye" on the work (e.g., stepping outside defined roles, especially being concerned about the end result even after doing one's own step in a process).

By ignoring the movement and transformation of information through human action, especially conversation, a designed workflow not only fails to explain how flows actually can happen at all, but leaves out the emergent effects of *serendipity*, such as stumbling on one order while looking for another or bumping into someone in the hall and learning about a new organizational priority.

2.1.4 Discussion

In this section, I discussed workflow modeling and simulation as a tool for business process re-design projects. Although the purpose of a workflow model is to represent the work of people in a work process, I have shown the limitations of this paradigm to model and simulate work as it really happens. I discussed the problems surrounding the validity of the statistics generated by a workflow simulation. Furthermore, we discussed many of the other limitations in representing how people actually work. A workflow model focuses on the sequential movement of jobs through process steps; therefore, the paradigm is very limited in representing the work from the point of view of a worker. People are seen as statistical resources. The representation of the flexible behavior of humans is not possible, and makes workflow models not a realistic

representation of how people work. This problem was emphasized by the example of trying to model the TC role in the T1 re-design project at NYNEX. Although the work activities of a TC were central to the work process, a workflow model was not able to show the “off-task” coordination work of the TC. Therefore, the model was a bad representation of the new work design. A simulation of work practice should emphasize:

- What people actually do, not just official job functions;
- What people are doing every minute of the day, where they are, and what they are perceiving, not just working on one task at a time;
- The collaboration between two or more agents, such as face-to-face conversations, telephone calls, etc, not just communication as a stochastic event;
- That people have personal identity, and are not interchangeable resources.

I conclude this review of the workflow model-based paradigm showing in Table 2-1 its limitations to represent people’s collaboration, “off-task” behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge and geography.

In the next sections, I describe several agent-based modeling approaches for modeling individual intelligent agent behavior. If we want to model the work of a group of individuals, it is obvious that an agent-based approach is warranted.

Table 2-1. Workflow modeling limitations

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
Task Model	Tasks <i>cannot</i> represent collaboration between people	Tasks that do <i>not</i> directly "touch" jobs flowing <i>cannot</i> be represented	Not supported	Tasks <i>cannot</i> be interrupted and resumed	Tasks <i>cannot</i> represent informal interaction between people	Not supported	Tasks are <i>not</i> associated with location
Input Model	N/A	Jobs <i>have</i> to "flow through" tasks to be worked on	N/A	N/A	N/A	N/A	Jobs are <i>not</i> represented as located
Resource Model	Resource interactions <i>cannot</i> be represented	Resources <i>cannot</i> be associated with tasks outside of the workflow	Resources <i>cannot</i> work on more than one task at the same time	Resources <i>cannot</i> be interrupted when working on a task	Resources <i>cannot</i> participate in informal interactions between other resources	Resources are numeric, <i>not</i> symbolic entities. Resources do <i>not</i> have cognitive capabilities	Resources are <i>not</i> located
Timing Model	Timing and coordination of tasks of different resources is <i>not</i> possible	N/A	N/A	When started, tasks take a fixed amount of time, and <i>cannot</i> be interrupted and resumed	N/A	Resources have <i>no</i> notion of time	There is only one timing model, and therefore we <i>cannot</i> represent multiple time zones based on location

2.2 COGNITIVE MODELING

Ever since modern cognitive psychology took form in the 1950s and 1960s, it has focused on aspects of understanding human cognition. In the early 1970s, it was Allen Newell who started to work on a unified theory of cognition that would address all aspects of cognition. Newell felt that the only way cognitive psychology would ever come to a unified theory, it needed to understand from the beginning how all the different pieces fit together (Newell 1973b). Newell, at the same time, introduced his answer to this dilemma. He described his first production system theory of human cognition. It was a single system that was able to perform a diverse set of tasks that occupied cognitive psychology. He described production systems as follows (Newell 1973a, pp. 463-464):

A production system is a scheme for specifying an information processing system. It consists of a set of productions, each production consisting of a condition and an action. It has also a collection of data structures: expressions that encode information upon which the production system works—on which the actions operate and on which the conditions can be determined to be true or false.

A production system, starting with an initially given set of data structures, operates as follows. That production whose condition is true of the current data (assume there is only one) is executed, that is, the action is taken. The result is to modify the current data structures. This leads in the next instant to another (possibly the same) production being executed, leading to still further modifications. So it goes, action after action being taken to carry out an entire program of processing, each evoked by its conditions becoming true of the momentarily current collection of data structures. The entire process halts either when no condition is true (hence nothing is evoked) or when an action containing a stop operation occurs.

Much remains to be specified in the above scheme to yield a definite information processing system. What happens (a likely occurrence) if more than one production is satisfied at once? What is the actual scheme for encoding information? What sort of collection of data structures constitutes the current state of knowledge on which the system works? What sort of tests are expressible in the condition of productions? What sort of primitive operations are performable on the data and what collections of these are expressible in the actions of productions? What sort of additional memories are available and how are they accessed and written into? How is the production system itself modified from within, or is this possible? How much time (or effort) is taken by the various components of the system and how do they combine to yield a total time for an entire process.

Over the years, Newell explored a number of variations on his production system concept, concluding with his Soar theory of human cognition (Newell 1990). Right now, there are at least four current and active production system theories: Soar (Newell 1990), ACT-R (Anderson 1993), 3CAPS (Just and Carpenter 1992), EPIC (Meyer and Kieras 1997). All these computational cognitive modeling systems are developed as implementations of a theory of cognition. As such, domain specific models of problem-solving tasks that are developed in these systems are seen as theoretically valid models of how humans perform problem solving. In this chapter, I briefly describe the Soar and ACT-R systems, in order to give a brief overview of the field of cognitive modeling. The reason for not describing all four systems, mentioned above is, a) because Soar and ACT-R are the best known and mostly used production systems for cognitive modeling, and b) to limit the space used to describe the nature of cognitive modeling.

2.2.1 Soar

Soar is a general cognitive architecture for developing systems that exhibit problem-solving behavior. Researchers all over the world, both from the fields of artificial intelligence and cognitive science, are using Soar for a variety of tasks. It has been in use since 1983.

Soar attempts to approximate rational behavior, using the following guiding design principles (Laird et al. 1999):

- The number of distinct architectural mechanisms should be minimized. Soar has a single framework for all tasks and subtasks (problem spaces), a single representation of permanent knowledge (productions), a single representation of temporary knowledge (objects with attributes and values), a single mechanism for generating goals (automatic subgoaling), and a single learning mechanism (chunking).
- All decisions are made through the application of relevant knowledge at run-time. In Soar, every decision is based on the current interpretation of sensory data, the contents of working memory created by prior problem solving, and any relevant knowledge retrieved from permanent memory.

Soar is based on the general hypothesis that *all* goal-oriented behavior can be viewed as the selection and application of operators to a state. A *state* represents the current problem-solving situation in memory, at a specific moment in the problem-solving process. The application of an *operator* changes the current state to a new state, which means that it is changing the representation of the state in memory. A *goal* (or subgoal) is seen as the desired outcome of an operator. Trying to reach its goals, Soar continually applies operators selected to achieve a goal. When an operator succeeds or fails, Soar selects the next operator for a subgoal of the current goal, or for a new goal. Reaching a goal can be seen as having the system reach a goal-state, i.e. the desired representation of objects, attributes, and values.

Soar has two types of memories for storing the different representational objects of a problem-solving process,

- *Working memory* contains descriptions of the current situation—the current state—including data from sensors, results of intermediate production firings, active goals and operators. Working memory is organized into objects. Objects are described in terms of attributes. The current state is the total of objects in working memory with their current attribute-values.
- *Long-term (or production) memory* contains *productions* that define how to react to the objects and their attributes in working memory. The productions in long-term memory can be thought of as the Soar program.

A Soar program contains the knowledge that is relevant in a particular problem-solving task (or a set of tasks), including the knowledge about when to select and apply operators to transform the states of working memory in order to achieve its goals.

2.2.1.1 Problem solving in Soar

Soar's long-term knowledge is organized around the functions of selecting and applying operators. These inherent Soar functions are performed using five distinct types of knowledge, *operator proposal*, *operator comparison*, *operator selection*, and *operator application*. Soar also has generic knowledge about making monotonic inferences about the state (i.e. *state elaboration*). Inferences that result in state changes in working memory have an indirect effect on the operator selection and application functions, because new state descriptions can cue new operators.

The knowledge used in the selection and application of operators is represented in terms of *production rules* encoded in the Soar program. Production rules are declarative "if-then" statements. The if-part of the rule is called the *condition* or *precondition*, and the then-part is called the *action* or *consequence*. The execution of production rules is referred to as *rule firing*. A rule *matches* when its conditions are met based on the current state in working memory. At that moment, the rule fires and its actions are executed. Executing actions means changing working memory (see Figure 2-8).

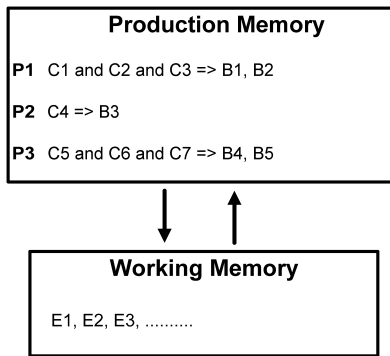


Figure 2-8. The Soar production system (borrowed from (Newell 1990))

Thus, by continuously testing conditions of the rules in long-term memory, matching the rules, based on the current state, the current-state of the system changes continuously. This cycle continues until there is a kind of equilibrium state between long-term and working memory. This means that the current state does not result in firing any of the rules in long-term memory. When this happens, and a specific goal-state has not yet been reached (i.e. the operator has not been fully applied), Soar is unable to make further progress and reaches an *impasse*. There are three types of impasses possible:

1. No operator can be selected, because none is proposed.
2. No single operator can be selected, because there are multiple operators possible and there is not enough knowledge to distinguish between them.
3. An operator is selected, but there is no additional knowledge to apply the operator.

When Soar is in an impasse, the architecture creates a *substate* (i.e. a sub search-space) in which a new operator selection-apply cycle can be started, with as the subgoal to resolve the impasse.

2.2.1.2 Learning in Soar

Learning is an important part of a theory of cognition. There are many aspects of how learning occurs in human cognition that are not well understood today. However, Soar has a primitive notion of learning called *chunking*. Chunking is a form of learning from experience. It is a way to transform specific problem-solving scenarios into productions stored into long-term memory for future use.

A chunk is a newly created production rule. The conditions of the chunk are parameterized working memory elements (WME) of a state that lead (through some chain of production firings) to a specific impasse resolution. The action of the chunk is the WME that actually resolved the impasse. In other words, chaining backwards over the specific WME's that were used to resolve a specific impasse creates the chunk. The first WME in this backward-chaining process is the action of the chunk being created. By parameterizing (replacing objects by variables) the chunk is made more generically applicable as a production rule in long-term memory.

2.2.2 Act-R

ACT-R is the result of long cognitive science research at Carnegie Mellon University, started in the mid-seventies with the introduction of the ACT production system (Anderson 1976). Like Soar, ACT-R is a theory of cognition trying to deal with the empirical knowledge of human cognition that has evolved in cognitive science. ACT-R consists of a theory of the nature of human knowledge, a theory of how this knowledge is deployed, and a theory of how this knowledge is acquired (Anderson 1976). ACT-R is also, like Soar, a computer system that implements the ACT-R theory. ACT-R can be used to develop computer simulations of a wide range of cognitive phenomena in memory, problem solving and skill acquisition.

The ACT-R theory assumes that there are two types of knowledge, declarative and procedural. *Declarative knowledge* is knowledge that we are aware of and can usually communicate to others. It is sometimes

referred to as *factual knowledge* or *axioms*. Examples include “George W. Bush is the 43rd president-elect of the United States,” and “one plus two is three.” *Procedural knowledge* is knowledge that we are not conscious of, and is applied in our problem solving behavior using our declarative knowledge.

Declarative knowledge is represented as *chunks*. Chunks in ACT-R are different from the notion of chunks in Soar. In ACT-R, chunks are WME that represent the factual statements in working memory. For example, the fact $1 + 2 = 3$ is represented as the following chunk (Figure 2-9):

Fact1+2	
isa	ADDITION-FACT
addend1	One
addend2	Two
sum	Three

Figure 2-9. Representation of an ACT-R chunk

Procedural knowledge in ACT-R, just as in Soar, is represented using *production rules*. A production rule is a condition-action pair. The condition specifies what must be true for the rule to apply, and the action specifies a set of things to do if the rule applies. Conditions in a production rule test for the state of the current goal and chunks in declarative memory, whereas the actions change the goal state.

From this brief description of the ACT-R architecture, we can infer that there are many similarities with the Soar architecture; ACT-R's declarative and procedural memory is synonymous with Soar's short-term and long-term memory, where its declarative and procedural knowledge is similar to Soar's objects in working memory and production rules in its long-term memory. The third comparison that holds, although should not be seen as a one-to-one comparison, is the similarity between *goals* in ACT-R, and *operators* in Soar. Goals in ACT-R are held in a separate memory structure, namely the *goal-stack*. In ACT-R, goal structures provide a higher-level organization to control the execution of production rules (relatively similar to operators in Soar).

2.2.2.1 Problem solving in ACT-R

Problem solving is organized through the *current goal*, which represents the focus of attention at each step in the problem solving procedure. ACT-R is always trying to achieve the goal that is at the top of the goal-stack. The current goal is pushed onto the stack, and is the next goal to be achieved. When it is done achieving a goal it pops the goal of the stack, which means that it will start achieving the next goal on the stack (see Figure 2-10). Thus, the goal-stack encodes the hierarchy of intentions that guide the problem-solving behavior.

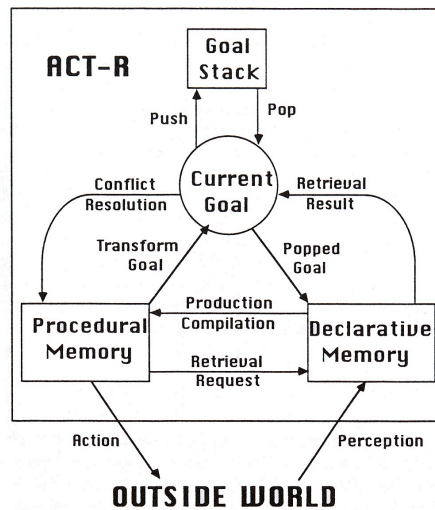


Figure 2-10. The ACT-R production system (borrowed from (Anderson and Lebiere 1998))

To select a production rule to achieve the current goal, there is a *conflict resolution* process that chooses one production from among the productions that match the current goal. The selected production is executed and can result in a transformation of the current goal, possibly resulting in pushing subgoals on the goal stack, or popping the current goal of the stack. It can also result in retrieval requests to declarative memory, which can be returned to the current goal satisfying it, thus causing popping the current-goal. Soar's type of chunking (i.e. production learning) is performed through a process called *production compilation*. Last, execution of a production can result in *actions* to be taken in the outside world, while *perception* of facts in the outside world can independently create chunks in the declarative memory. This last piece is the subject of the next section.

2.2.2.2 The total cognitive system

Newell describes a larger system that includes *perception* and *motor* behavior as the total cognitive system, shown in Figure 2-11 (Newell 1990, pp. 194-195). Here is where we identify a significant difference between the ACT-R and Soar systems implemented in software. Although, both theories describe the need for a total cognitive system, only ACT-R has implemented such a total system.

ACT-R has a number of extensions that have been created by different researchers. First, there is a *visual interface* that incorporates ideas on visual attention and perception. This extension implements in software the capability of accessing information from a computer screen and dealing with a keyboard and mouse, in a similar way human subjects do in psychological experiments. It parses "screens" as humans do and enters key-presses and mouse gestures. The data record that is created using this interface is indistinguishable from the data record that human subjects create. The visual interface is extended more generally to also deal with *audition*, *speech*, and other *hand gestures*.

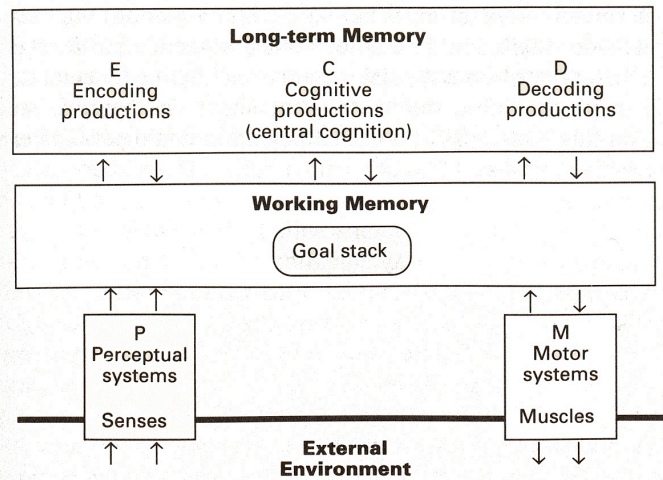


Figure 2-11. The total cognitive system (borrowed from (Newell 1990))

2.2.2.3 Human performance models

The grounding of ACT-R in real experimental data has created the ability for developing models whose correspondence to actual cognition can be validated with a subject performing similar experiments in the real world. This, in some sense, has created a level of reality in cognitive modeling. ACT-R makes predictions about every aspect of the empirical phenomena that are being simulated. For example, in simulating memory experiments the ACT-R model predicts not only the correctness of response choices and response latency, but also the steps involved in the problem solving process.

The use of this is mostly relevant in the field of human factors, where the focus of research is on understanding the impact of a particular interface design on human performance.

2.2.3 Discussion

The home page of the ACT group¹³ at Carnegie Mellon University states: “The architecture takes the form of a computer simulation that is capable of performing and learning from the *same tasks* worked on by human subjects in our laboratories.” This says that ACT-R’s main focus is to simulate psychological laboratory experiments. The reason for this is the following. Using simulation models of psychological experiments that can be done in a laboratory setting with human subjects, the simulation models can be validated by comparing the simulation output data with data from these real-life laboratory experiments.

Thus, the sole purpose of implementing the ACT-R theory into the ACT-R computer simulation system is to verify and validate the theory. In other words, the ACT group’s method for researching unified theories of cognition is through *modeling and simulation*. I make this point, because of two reasons; 1) I apply a similar research method for developing a theory of work practice, and 2) it helps me to show the limitations of ACT-R (and Soar, for that matter) in applying it for a different research problem, namely the development of a theory of work practice.

¹³ <http://act.psy.cmu.edu/ACT/act/actr.html>

Table 2-2. Limitations of cognitive modeling and simulation

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
SOAR	No There is <i>only one</i> actor	No Tasks are performed through goal-directed reasoning	No Only through complex goal switching	No Goals <i>cannot</i> be interrupted and resumed	No There is <i>only one</i> actor	Yes	No
ACT-R	No There is <i>only one</i> actor	Yes The actor can react to perceptions from the outside world	No Only through complex goal switching	No Goals <i>cannot</i> be interrupted and resumed	No There is <i>only one</i> actor	Yes	No

The limitations of applying cognitive modeling architectures to work practice modeling and simulation are given in Table 2-2. However, the main reason for being skeptical about using cognitive modeling environments, like Soar and ACT-R, for modeling work practice has to do with the level at which cognition is modeled. As described in this chapter, using Soar and ACT-R as examples, cognitive models are represented at a level that allows the cognitive science researcher to model cognitive cycles in the human brain. These cycles have a length of no more than 50-100msec. Even though I have not yet defined what needs to be included in a model of work practice, it seems clear that work practice needs to be shown at a higher-level of human activity than the low-level cognitive cycles of a person. One of the reasons for believing this is because of the scale factor. If we want to model a work practice in an organization of, for example, tens of people, it seems obvious that modeling each person in this organization at the level of their cognitive cycles would not be useful, even if it would be possible. The objective of a work practice model must lie at the level of showing what the total system behavior is. It seems obvious that this should be done by modeling each person's activities in the physical world, based on some relatively high-level representation of the reasoning behavior of each individual, and the impact this has on the individual's action in the physical world.

The notion of multiple actors or agents being involved in problem solving, and the research into organizations of multiple actors is the topic of the next section, in which I will discuss the research field of distributed artificial intelligence.

2.3 DISTRIBUTED ARTIFICIAL INTELLIGENCE

I am interested in modeling and theorizing about activities of people. People are inherently social actors and we, therefore, must be concerned with the social dimension of action and knowledge. Gasser states that classical artificial intelligence (AI) research is largely *a-social*, meaning that the unit of analysis is a computational process with a single locus of control and knowledge (Gasser 1991). Because of this, it has been inadequate with dealing with social human behavior. Gasser investigates how contemporary DAI deals (and should deal) with the *social* conception of knowledge, action, and interaction. In (Gasser 1991) he makes an argument that distributed artificial intelligence (DAI) is fundamental in the research on how agents coordinate their actions, use knowledge about beliefs, and reason about the beliefs and actions of other agents. Here, I draw a similarity between the research problems in the DAI literature and the problems in simulating work practice.

The notion of “social” in DAI is in my opinion too limited. The term social in DAI is meant as “more than one.” That is, DAI does not concentrate on problem-solving behavior as sitting in the head of a single agent, but investigates problem-solving behavior as a distributed multiagent process. As such, the word “social” means that there is communication and coordination between multiple agents in a problem-solving task. Detail can be found in (Bond and Gasser 1988; Gasser and Hill 1990). However, the definition of “social” is broader, and relates more closely to the way the group as a whole acts and interacts in the environment. The notion of social conception of knowledge and action is not a new idea. Social psychologist George Mead stated (Mead 1934):

We are not, in psychology, building up the behavior of the social group in terms of behavior of the separate individuals composing it; rather we are starting with a given social whole of complex group activity, into which we analyze (as elements) the behavior of each of the separate individuals composing it. We attempt, that is, to explain the conduct of the individual in terms of organized conduct of the social group, rather than to account for the organized conduct of the social group in terms of the conduct of the separate individuals belonging to it. For social psychology, the whole [society] is prior to the part [the individual], not the part to the whole; and the part is explained in terms of the whole, not the whole in terms of the parts.

The traditional techniques and methods of AI do not include any fundamental social elements. The focus is on the individual as the object of knowledge, truth and knowing. Gasser provides a great example of this limitation in AI research. The example is centered on the concept of *commitment*, and provides an excellent description of how an individual’s commitment is not just based on his or her individual *relativized persistent goal* (Cohen and Levesque 1990). In contrast, an individual’s commitment is based on an agent’s overall participation in many settings (activities) simultaneously (Gerson 1976), exemplified by Gasser’s example (Gasser 1991):

For example, imagine that a Los Angeles industrialist takes off in an airplane from Narita airport, bound for California, after formulating preliminary business deals in Tokyo and telephoning her associates in Los Angeles. While flying, she is participating in many settings simultaneously: the activity in the plane, the ongoing business negotiations in Tokyo and in Los Angeles (where people are planning for her arrival and making business judgments while considering her views, even in her absence). Her simultaneous involvement in interlocking courses of action in all of these situations provides the commitment to her arrival in California. Both she and others balance and trade off her involvement in joint courses of action in many different situations. Moreover, whether she makes a choice or not, she is committed to landing in LA because the plane is not in her control. Her commitments in any of these settings *amount to* the interaction of many activities of many agents in many other settings. Since this multi-setting participation occurs *simultaneously* in many places, it can’t be located simply to where she physically ‘is’. In other words, the notion of commitment is distributed because the agent of commitment—‘she’—is a distributed entity.

Although this example focuses on the notion of commitment, it is an excellent example of *collaboration* between multiple people in a work system. This example shows the importance of individual participation, knowledge, and location in the system as a whole. It shows that resources are not interchangeable, and that the work practice cannot be understood without a close investigation of the collaboration between the individual agents and the context in which this collaboration takes place. As Gasser states (Gasser 1991):

[...] since continued participation is distributed and simultaneous, it isn't based on localized, individual choices and goals.

The next few sections discuss a number of agent-based systems from the field of DAI. None of these systems provide a complete solution for dealing with simulating work practice, however they have formed a basis for our Brahms multiagent system.

2.3.1 TacAir-Soar

In (Tambe et al. 1995) an intelligent agent simulation environment is described for simulating battlefield scenarios and the knowledge intensive reasoning of independent pilot-agents. This environment is called TacAir-Soar, a kind of virtual world that can be populated with not only humans, but also with intelligent automated agents (AI systems). Such synthetic environments provide a new laboratory in which intelligent agents can be studied. Intelligent agents can be substituted for humans, such that a large number of entities can be used to populate the virtual world. A benefit of such an environment is that artificial agents can simplify and speed-up experimentation by providing more control of behavior, repeatability of scenarios, and an increased rate of simulation, faster than real-time simulation.

The goal of TacAir-Soar is the production of behavior that is close enough to that of humans, and to force the other entities to interact the same way as they would interact with humans. Although ambitious, they do not have to deal with low-level perception and robot control. There is also no verbal interaction between opposing entities, and cooperating agents restrict their communication to the details of the current mission. TacAir-Soar was to provide synthetic pilots (IFORS) for all the missions in STOW-97 (Simulated Theater of War), a large-scale simulation of a tactical exercise that took place in 1997, including fighters, troop transports, reconnaissance aircrafts, and helicopters. The set of requirements for the automated pilots include:

- Goal-driven and knowledge-intensive behavior;
- Conformance to human reaction and limitations;
- Performance of multiple simultaneous tasks;
- Episodic memory.

Pilot agents in TacAir-Soar are created as individual knowledge-based systems within the Soar integrated architecture (Laird et al. 1987) (Newell 1990). TacAir-Soar represents a generic automated pilot. Specializing it with specific vehicle parameters provided to them during the briefing process creates specific automated pilots. These pilot agents then participate in battlefield simulations by flying simulating aircrafts provided by ModSAF (Calder et al. 1993), a distributed simulator that has been developed commercially for the military.

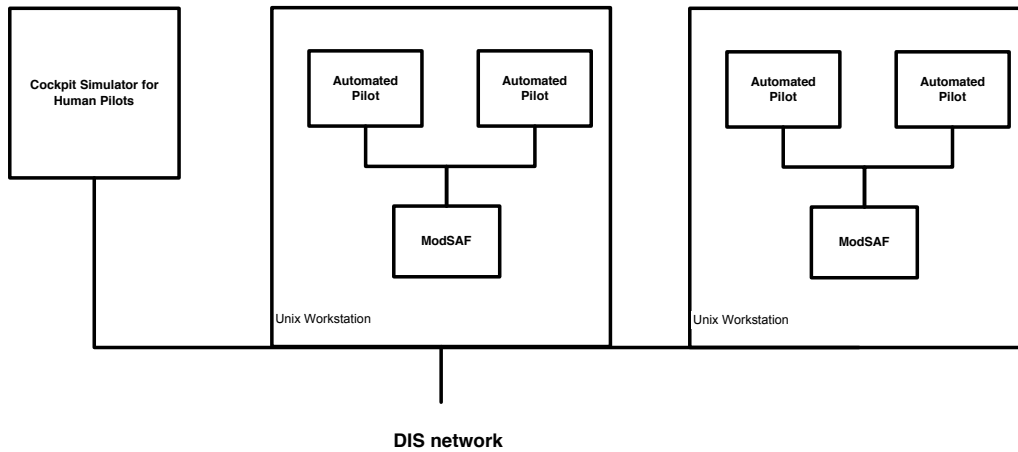


Figure 2-12. Human and Automated Pilots interact with the DIS environment using Distributed Simulators

TacAir-Soar has been constructed from Soar through the addition of perceptual motor interfaces (in the form of C-code) that allow pilots to fly ModSAF planes in the DIS (Distributed Interactive Simulation) environment (Thorpe et al. 1989). Each automated pilot is implemented as a Soar agent, interfacing with the simulated plane it is flying. To do this, they implemented a cockpit abstraction on top of ModSAF that allows TacAir-Soar to focus on behaving like a pilot, while ModSAF simulates planes, sensors, and weapons (see Figure 2-12).

2.3.1.1 Goal-driven task behavior

Each automated pilot agent has specific knowledge about tactical air-combat, and is implemented as a Soar knowledge base, having independent reasoning behavior. Goal-oriented and knowledge intensive behaviors are addressed by the manner in which Soar uses its architecture and knowledge in the process of dynamically expanding and contracting a goal hierarchy (Laird et al. 1987). Task switching also arises from Soar's decision-making abilities, but then specifically applied to the selection and switching of tasks. Tasks (also called goals) are represented as operators in Soar, and are the main foci of its decision-making. Task decomposition arises from Soar's ability to automatically generate a new task context when a decision is problematic. Task decomposition is achieved by combining task context generation with rules that generate preferences about which subtasks are appropriate for parent tasks. Real-time interaction with the DIS environment arises from the combination of Soar's incorporation of perception and action within the inner loop of its decision making capabilities-thus allowing all decisions to be informed by the current situation (and interpretations of it, as generated by rule firings) and the use of ModSAF as the interface to the DIS environment.

2.3.1.2 Conformance to human reaction and limitations

Reactivity of the individual pilot agents is addressed through a combination of Soar's use of productions to enforce context sensitivity in the representation of the knowledge, and Soar's decision-making procedure. Soar can react to changes by suggesting new goal-operators be pursued at any level in the same goal hierarchy, generating preferences among suggested operators. The decision procedure determines what changes have to be made to the goal hierarchy. Reactivity is limited to changes from within the decision-making process, and not from external available cues.

For example, communication between agents is simulated by the transmission of radio messages via the ModSAF simulation substrate, through the DIS network. That is, a discrete event simulates the transmission of a radio message between agents. The content of the radio message is received by an agent through the ModSAF interface into the working memory of the agent. Agents can react to radio messages through the activation of rules in the current task context that can propose new operators to be evaluated. This limits the agent to reacting only to radio messages that are relevant in the current context.

As a result, one of the limitations of TacAir-Soar in its use for modeling work systems is its inability to react to general external cues, other than the ones built in through the ModSAF cockpit abstraction and the DIS network. People, on the other hand, constantly react to the state of their external environment, regardless of their goal context at that moment. For example, when the telephone rings, people react to it. Depending on the activity we are involved in at that moment, we either pick it up, or let our answering machine answer. If we're not in the same location as the telephone, we do not notice the telephone ringing. When we decide to pick up the phone, we will most likely engage in a total different task context. This poses a problem for the way the Soar decision-making process works.

2.3.1.3 Performance of multiple simultaneous tasks

The Soar architecture is inadequate in certain requirements. One requirement that poses an inherent problem in modeling social human behavior is the inability of the simultaneous performance of multiple tasks. People inherently work on multiple tasks at once. TacAir-Soar agents need to simultaneously execute maneuvers to destroy the opponent, survive opponents' weapon firings, as well as interpret opponents' actions. The limitation of Soar is that it cannot create multiple goal hierarchies to serve multiple high-level tasks. The approach taken in TacAir-Soar to handle this problem is to create operators for simultaneous goals as needed, and to add these operators at the bottom of the active goal hierarchy. Although this may work, with sufficient care taken, it is not a real solution to the modeling of realistic human behavior. Soar interprets "lower" goal operators as being dependent on the higher-level goals, although these lower-level goals are supposed to be interpreted as goals for independent tasks. Jones, et al worked on changing the architecture to allow "forests" of goal hierarchies (Jones et al. 1994). Covrigaru approached the problem by being able to flush the current goal hierarchy whenever a new one needs to be established (Covregaru 1992). However, these approaches mean a change to the Soar architecture that would allow the dynamic compilation of new goal hierarchies.

2.3.1.4 Episodic memory

Endel Tulving introduced the term *episodic memory* (Tulving 1969). The notion of episodic vs. semantic memory arises from the central learning tradition of American experimental psychology. Tulving made clear that learning of verbal material was tied to a specific episode, i.e. a specific time and context in which the learned material was memorized, and thus was associated with. Episodic memory in TacAir-Soar is used primarily to support explanation. It is also used to a limited extent during simulation, so that an operator's current actions are interpreted based on its actions in the past. The general constraints are that episodic memory should add minimal processing overhead, and it should not substantially increase working memory. Episodic memory is a basic characteristic of human cognition, something a unified theory of cognition ought to provide for (Newell 1990). This brings tough issues for the Soar architecture especially. The approach taken in TacAir-Soar is that the sequence of events is recorded in working memory so that it can be recalled accurately. The states in which events occur are stored by committing state changes to long-term memory. Long-term memory employs chunking, which allows agents to learn new productions from episodic memory.

This is a point where TacAir-Soar wins over Brahms. Currently, Brahms agents have no significant episodic memory. Brahms agents only have a list of "current-beliefs" which is changing constantly. A trace of past activities, and states is not memorized. Brahms agents have no long-term memory, nor the ability of chunking. This is a future research issue that is closely related to the issue of learning. Although learning is an important piece of modeling human behavior, it falls outside the scope of this thesis. I return to a short discussion of learning as a topic for future research in the conclusion chapter of this thesis. However, for now suffice it to say that the Brahms simulation engine does keep track of all changes to the system, and that these historical events are created so that the history of a simulation can be saved in a database, and investigated post-simulation. However, at this moment, Brahms agents cannot access these historical events.

2.3.2 Phoenix: simulating fire-fighting in Yellowstone National Park

Paul Cohen, et al (Cohen et al. 1989) developed an agent-based simulation environment for simulating how fires in Yellowstone National Park are fought. The research objective of the Phoenix project is to understand

how complex environments constrain the design of intelligent agents. In contrast to the objectives of this research, in which the goal is to understand how work processes are created in terms of the work practices of people, the aims of the Phoenix research are more of a technical AI nature:

- Real-time adaptive planning
- Approximate scheduling for coordination of multiple planning activities
- Knowledge representation of plans and measuring progress towards the goals

They describe the Phoenix environment as follows:

We began the Phoenix project by designing a real-time, spatially distributed, multi-agent, dynamic, ongoing, unpredictable environment. (Cohen et al. 1989)

Phoenix is a multi-layered system. Figure 2-13 shows the architectural layers.

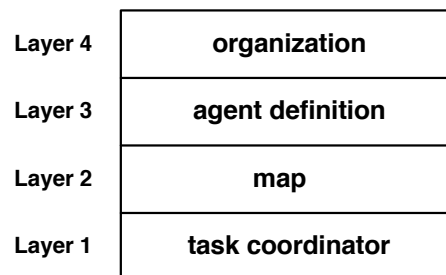


Figure 2-13. Phoenix layers

Layers 1 and 2 implement the environment (i.e. the forest and forest fires), in this case a part of Yellowstone National Park. Layer 3 is the agent design layer, which is the layer that creates the cognitive and reflexive behavior of the agents. Layer 4 is a model of the organization of multiple fire fighters.

2.3.2.1 Task coordinator layer

The task coordinator layer is responsible for creating the *illusion* of simultaneity among fires, and the agent's physical and internal actions. The task loops over all agents in each clock tick, changing the environment and the agent's actions accordingly. The clock grain size of the Phoenix simulator is 5 minutes. This means that the smallest action or change in the environment takes a minimum of 5 minutes of simulated time. You can increase the clock grain size to make the simulation more efficient. However, increasing the clock grain size will make agents become discordant with the environment, and with each other. When the clock grain-size becomes too large, it is possible for more than one action for an agent to have taken place in one simulation clock-tick. Communications between agents might have been missed, and changes in the environment might not have been recorded by the individual agents, leading to a situated-specific model that is not in sync with what actually happened in the simulation.

2.3.2.2 Map layer

The fire simulator resides in Phoenix's *map layer*. A Phoenix map is a composite of a two-dimensional data structure in which, for each map coordinate, information is stored about the environment. The environment is described as a set of symbolic features found at that specific coordinate on the map. These features include values for type of ground cover, elevation, features such as a road, a river, houses, et cetera., and the state of the fire at that coordinate (fire intensity).

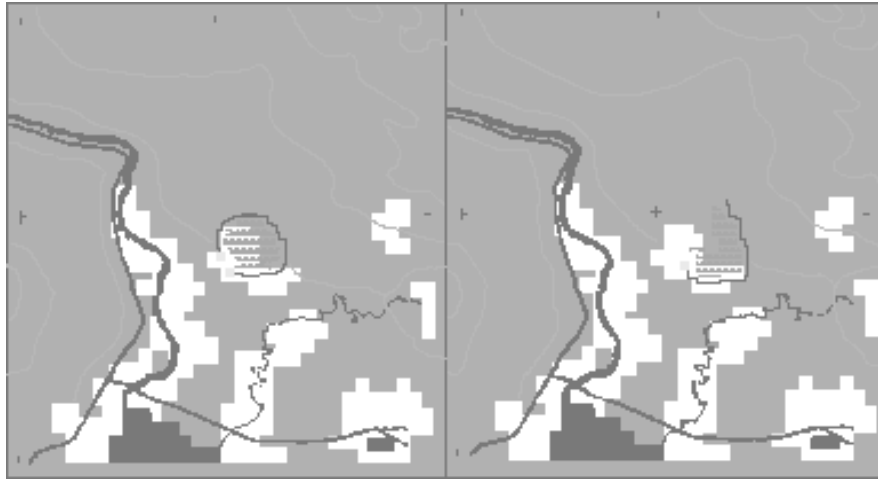


Figure 2-14. Screen display of Phoenix's situation-specific model (here gray and white indicate forest, which is seen hugging a river on the left side; the fire is near the middle (darker gray dots), nearly surrounded by the bulldozers in the view on the left). The left pane is the view of the real world, the right pane is the view of the fire boss.

The process of changing the map (i.e. changing the environment) during each simulation clock-tick is parameterized according to these defined concepts (Cohen et al. 1989):

Fires spread in irregular shapes and at variable rates, determined by ground cover, elevation, moisture content, wind speed, and direction, and natural boundaries. For example, fires spread more quickly in brush than in mature forest, are pushed in the direction of the wind and uphill, burn dry fuel more readily, and so on.... Fire-fighting objects are also accurately simulated; for example, bulldozers move at a maximum speed of 40 km/h in transit, 5 km/h traveling cross-country, and 0.5 km/h when cutting a fire-line. (p. 34-35)

The fire itself is implemented as a cellular automaton in which each cell at the boundary of the fire decides whether to spread to its neighboring cells. The simulator is generic in the sense that it can be used to simulate any type of environment involving maps and needs to simulate changes to the environment, based on the changes to the symbolic representation of the environment.

As is shown in Figure 2-14, the map layer also controls the movement of the agents, such as the bulldozers trying to surround the fire and cutting a fire-line, based on the situation-specific description of the terrain.

2.3.2.3 Agent Design Layer

Agents have two independent parallel mechanisms for generating actions: a reflexive component, and a cognitive component. The *reflexive component* generates actions for quick changes of direction on the order of seconds. The agent's sensors trigger reflexes. For example, when a bulldozer agent is about to drive into a fire, a reflex stops it and further reflexes handle the fine tuning of the movement of the agent to keep following the road without getting into the fire. Reflexes hardly cost any CPU time, but have no memory. They are merely a quick action (i.e. a reflex), based on triggers from the sensors.

The *cognitive component* generates and executes *lazy skeletal plans*, which are stored prescribed action sequences that, when executed, are instantiated with situation-specific data (Freed 1998). The cognitive component executes plans using a selection process that first decides which action to execute, next it finds out how much time is available to execute that action, and last it decides what execution method should be used for the action to execute based on the time available. An agent can execute several plans simultaneously; for example, when there are multiple fires to combat. The planning process goes as follows (Cohen et al. 1989):

Planning is accomplished by adding a selection action to the timeline to search for a plan to address some conditions. Executing the selection action places an appropriate plan action or primitive action

on the timeline. If this new entry is a plan action, then it expands into a plan when it is executed by putting its subactions onto the timeline with their temporal interrelationships. If it is a primitive action, execution instantiates the requisite variables, selects an execution method, and executes it. In general, a cognitive agent interleaves actions from the several plans it is working on. (p. 43)

Just as in TacAir-Soar, plans cannot be interrupted, because saving a state of the world and context switching is prohibited in the architecture. Therefore, to change an action the agent has to generate an error selection action “deal-with-error.”

The only way an agent can change its activities (i.e. re-plan) is to view a change as an error condition, and try to fix the current plan by expanding it into a kind of error recovery plan. Activities in Phoenix are sequential non-interruptible actions. However, the error-conditions in Phoenix are dynamically generated. This architecture allows for more flexibility in simulating the work activities of individual agents than workflow simulation models.

The reflexive and cognitive components interact via flags that are set by the reflexive component when reflexes execute. Since plans executed by the cognitive component can take several simulation hours to complete, the reflexive component takes over and changes the directional behavior of the agent. When the cognitive component notices the flag, it might react by changing its plan by calling the “deal-with-error” action. A Phoenix agent deals with two time scales requiring micro-actions, such as following a road, and cognitive processing, such as route planning. The combination of the reflexive and the cognitive components is designed to handle these time-scale mismatches.

2.3.2.4 Organization Layer

Agents in Phoenix are centrally organized in a hierarchical organization of fire-fighting agents. There is always one coordinating agent for each organization of fire fighters. This agent is called the *fire-boss*. The fire-boss is a purely cognitive agent that coordinates all fire-fighting agents’ activities, scheduling and communicating action directives. The fire-boss receives status reports from the fire-fighting agents, including fire sightings, position updates, and action completions. The fire-boss has a global view of the fire situation, while each fire-fighting agent has a limited individual view of the fire. Based on this global view and the updates, the fire-boss chooses global plans from its plan library. It communicates the actions in these plans to the agents. When an agent receives a plan, it selects a plan from its own plan library that implements the received action. Although the fire-fighting agents and the fire-boss communicate, there is no communication amongst the fire-fighting agents (i.e. there is no *cross-talking*).

The organization model in Phoenix is very limited, and is not conducive to collaboration amongst the fire fighters. The fire-boss orchestrates the work, and is a kind of meta-agent, whose sensors and detectors are the fire-fighting agents. It has the overall picture, the size of the fire, weather conditions, and action that already have been taken. None of this information is shared across the group of agents. Although this seems a good model to represent the TC in the T1 model (see 2.1.3.4)—it allows us to describe the coordination work of the TC—it does not allow for the agents in the central office to collaborate on a test with the technicians in the field.

This brings us to the next section in which we discuss the limitations of the Phoenix and TacAir-Soar systems in simulating the work practice of humans.

2.3.3 Discussion

Both TacAir Soar and Phoenix are multi-intelligent agent environments. The reason for choosing these two environments in my description of relevant previous work is that they are on the opposite ends of the spectrum, as far as simulating human behavior is concerned. TacAir Soar is based on the Soar architecture. Soar was developed as a theory of human cognition, stating that human cognitive processing is symbolic. Newell’s claim is that the Soar architecture is representative of how the human cognitive process works. As such, TacAir-Soar models pilots at a very fundamental cognitive level. The focus in TacAir Soar is on how the reasoning process of combat pilots can be simulated to the level of cognitive reaction times (i.e. human performance). The fact that there are multiple agents, simulating multiple pilots flying in a squadron, is to

allow for training of individual pilots. The focus is on the individual, and less on the collaboration between pilots. For example, it would be hard to simulate one pilot being attacked by an enemy aircraft, and another friendly pilot collaborating with this pilot to shake off the enemy aircraft.

Phoenix, on the other side of the spectrum, simulates a group of fire fighters who together fight a fire, coordinated by a fire-boss. At first, it seems that the Phoenix environment simulates collaboration. However, when we look closer we see that the fire-fighting agents in Phoenix do not even communicate together, a necessary prerequisite if we want to simulate collaboration. It is self evident that real fire fighters communicate together to setup strategies, change plans in times of despair, et cetera. A Phoenix agent does not represent how a fire fighter fights fires; instead it simulates how distributed lazy skeletal planning can be done. It would be very difficult to have a fire-fighting agent select the appropriate plan on the fly when suddenly, out of nowhere, a burning tree is falling down on a colleague ten feet away from where it is standing. Phoenix agents perform pre-specified plans that are assigned by a coordinating fire-boss agent. Because the overall view of the situation by the fire-boss, plans are being selected and agents directed. The agents are like robots acting out what the fire-boss instructs them to do, with local reactive behavior dependent on the environment they encounter, but independent of the fire-boss. In this sense, fire-fighting agents are fighting fires in groups without being aware that they are working in collaboration with other agents. An agent might be aware of some other agent, but it does not know what it is doing, and even more so, why it is doing what it is doing and that they are actually working together to fight the fire. In comparison with TacAir-Soar, Phoenix agents do not simulate the way human cognitive processes happen.

The goal of Phoenix is to design the right software-agent architecture for the needed agent behaviors and environmental characteristics to fight fires. Actually, the Phoenix architecture is a generic simulator for the *central* coordination of distributed agents in a natural environment. As such, the Phoenix environment contains all the components that are necessary to design an environment to simulate the work practices of individuals and groups. The problem is in the view the developers of Phoenix take in agent cognitive behavior. Phoenix, not surprisingly, uses a more classical AI planning approach, i.e. lazy skeletal planning.

With Table 2-3, I conclude this section on DAI by showing the strengths and limitations of the two reviewed systems, TacAir-Soar and Phoenix, in particular with regards to their ability to represent people's collaboration, "off-task" behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge and geography.

Table 2-3. Limitations of Distributed Artificial Intelligence

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
TacAir-Soar	No Communications of radio messages. Agents are <i>not</i> aware of each other's tasks	No Only high-level goal/tasks of the mission	No	No When contexts are switched tasks are stopped	No Only specific goal-directed behavior	Yes	Yes But, there is <i>no</i> separate geography model
Phoenix	No	No	Yes	Yes	No	Varies Firefighter agents for the most part are reactive agents. The fire boss agent is a deliberative	Yes

						agent	
--	--	--	--	--	--	-------	--

2.4 COMPUTATIONAL ORGANIZATION THEORY

Organizational issues were one of the first issues researched with the help of computational modeling. In the fifties, it was Herb Simon and his colleagues March and Cyert, at what was then Carnegie Tech, who started using computational modeling to create a behavioral theory of organizations (Simon 1955). Computer simulation is becoming a more accepted and indispensable method for organizational research and theory building. The need to understand organizational behavior is moving into the realm of detailed computational models of people, tasks, dynamic and adaptive ecological systems. Within organization theory models, organizations are often too complex to be analyzed by conventional techniques that lead to closed-form solutions. Computational organization theory (COT) researchers view organizations as a computational system of multiple “distributed” agents that collectively work on organizational tasks, use resources, have knowledge, skills, and communication capabilities. In (Carley and Prietula 1994b) and (Prietula et al. 1998), different organizational multiagent simulation models are presented as simplified descriptions of what happens in real organizations. Although simplified, these models are still sufficiently complex to simulate the dynamic behavior of organizations that allows for predictions and theory development. There are three basic reasons for using multiagent simulation as one of the main technologies in COT research (Carley and Prietula 1994b):

1. Many models contain non-linearities that cannot be eliminated. In modeling reality the non-linear behavior of individual agents and groups are central to the aspect that is being studied.
2. Differential equations do not deal with the differences of the discrete items in organizations, such as the people, tasks, and organization.
3. Agents in an organization act in parallel and adapt to the behavior of others. The behavior of a group is thus recursively defined. Controlling this fine order of agent interaction, and enabling agents to adapt is most effectively handled by simulation. Especially, if we are interested in investigating the behavior of large groups of agents, it is almost necessary to turn to simulation.

2.4.1 ACTS theory

Carley and Prietula, in (Carley and Prietula 1994a), describe an extended model of *Bounded Rationality* (Simon 1955). The model of bounded rationality states that agents in an organization may be rational in intent, but less than rational in execution because functional limits on cognition restricts their ability to achieve optimality in the pursuit of their goals (Simon 1976). The theory of bounded rationality was developed to replace the limited models of agents in theories of economics and organizations with a better approximation of the actual capabilities of people’s decision making. The ACTS theory extends the model of bounded rationality and incorporates a general process theory of organizations. In the ACTS theory organizations are viewed as collections of intelligent agents who are cognitively restricted, task-oriented, and socially situated. The ACTS theory extends the model of bounded rationality in two ways:

1. It replaces the general principles of bounded rationality with a broader perspective of a cognitive agent.
2. It replaces the general notions of environmental constraints with specific environmental perspectives, a) the task, and b) the organizational social situation within which the task and the agents are situated—situatedness.

Within ACTS, organizations are build-up of individual intelligent agents whom together perform tasks, collaborate, and communicate in a social environment. The agent’s knowledge, which is constantly changing, mediates the effect of the task and the social situation on the individual agent (micro-level) and organizational behavior (macro-level). Agents and tasks are situated in the organizational environment.

ACTS tries to explain such individual and organizational behavior and performance by supporting the development of a set of computational models:

- a cognitive model of an agent,
- a task model,
- a model of the social situation.

At the micro-level, the ACTS theory focuses on how a given organizational design affects the behavior and performance of individual intelligent agents whom communicate and reason within a social environment and situation. At the macro-level, the ACTS theory focuses on the behavior and performance of groups and organizations, given the fact that groups and organizations are comprised of intelligent agents whom are socially situated and task-oriented.

In ACTS theory the actions and decisions of intelligent agents are a function of the agent's cognitive architecture and knowledge (Newell 1990). The mechanisms by which an agent processes information, learns and makes decisions are a function of the cognitive architecture of the agent, the (social) position of the agent in the organization, and the tasks in which the agent is engaged. Thus, the ACTS theory refocuses the attention of the researcher interested in organizations on the details through which the task and social environment influence the individual agent and the group performance.

Many COT researchers use a multiagent version of the Soar system (Laird et al. 1987), because Soar is an implementation of a cognitive architecture that simulates the reasoning behavior of an individual. By integrating multiple copies of the Soar architecture in a distributed communication environment, a multiagent simulation environment for COT research can be created. In the next two sections, I briefly describe two of such environments: Plural-Soar, and Team-Soar.

2.4.2 Plural-Soar

Plural-Soar models a warehouse where multiple workers fill orders by retrieving items stored in stacks located in different places (Carley et al. 1992). In Plural-Soar, each Soar agent runs on a separate workstation, with the agents communicating over network connections. Plural-Soar consists of Soar production rules that define the agent's task knowledge. The task agents perform involves moving to a particular location (the order stack), and after possibly waiting in line, picking the next order from the stack, and then determining where the item from the order (in any of the known item stacks) may be in the warehouse, and last, moving to the item location to pick up the item to fill the order. Agents have a memory of the contents of the item stacks they have encountered. They can also broadcast requests for item locations to the other agents. This research focuses on the examination of how different combinations of cognitive constraints (such as communication and memory capability) combined with varying organizational structures (for example, different sizes) result in different organizational behaviors.

In subsequent work, adding elements (social knowledge) extended Plural-Soar in order to bring social elements of groups to the system (Carley et al. 1993). A social agent is defined by the decreasing information processing ability of the agent; omniscient, rational, bounded rational, cognitive, or emotional-cognitive, and by its knowledge of the social environment; non-social, multiple agents, interactive multiple agents, organizational structures, group goals, cultural history (Carley and Newell 1990). This model of a social agent was implemented in Plural-Soar in two ways. First, an agent has a social memory about the *reliability* of other agents. Secondly, the reliability of another agent depends on the correctness of previous given information by that agent. For instance, if inaccurate information was given by an agent, on the location of a specific item, the reliability of that agent is "down graded." After two unreliable pieces of information, an agent is determined "unreliable," and no more information from that agent is accepted. Thus, while in the first version of Plural-Soar information from other agents was accepted unconditionally, in the extended social agent version, communication from other agents is accepted or rejected on the basis of "social historical knowledge" of that agent.

In order to test the extremes of social behavior they then varied the social characteristics of agents: honesty, cooperation, and benevolence. In the study, they measured concepts such as cognitive effort, physical effort, communication efforts, and wait time from five different organizations. Conclusions then were drawn about the differences in these measures from different types of agents in organizations.

2.4.3 Team-Soar

Team-Soar models the decision-making behavior of a team of four commanding officers (CO) from different units in a naval carrier group (Kang et al. 1998). The four CO's are modeled as interconnected individual Soar agents. The agents are distributed in a simple authoritarian hierarchical organizational structure, which means that there is one leader, the CO of the aircraft carrier, and three equal subordinates, a CO of the coastal air defense, a CO of an AWACS air reconnaissance plane, and a CO of an Aegis cruiser. Each agent can communicate with all other agents, however the leader controls the team-based problem-solving task. The objective of the Team-Soar model is to track an aircraft by radar, and evaluate and decide in a team effort a course of action. To make a judgment, a Team-Soar agent first interprets the raw data in terms of nine attributes; speed, altitude, size, angle, direction, corridor-status, radar-type, range, and IFF, and evaluates them on a scale from one to three. When an agent has made the evaluations, by applying its knowledge in terms of Soar production rules, it makes a judgment about which of the seven possible actions to recommend (i.e. communicate) to the leader. Recommendations range in degree from ignore (a value of zero) to defend (a value of six), with intermediate states of review, monitor, warn, ready, and lock-in. When the leader has received the recommendations from all three subordinates, including its own, it makes a team decision, based on a team decision scheme, such as majority win or average win.

Team-Soar uses two types of communication strategies; one-to-one communication and broadcasting. One-to-one communication is used when one agent sends a message to another agent. This happens when the subordinate agents communicate their decision to the team leader agent. Broadcasting is used to send a message to all agents simultaneously. There are four different types of information that can be communicated: raw data, evaluations, judgments, and decisions. The raw data are the values of the nine attributes. An evaluation is an interpretation of the raw data. A judgment is a team member's recommendation on a decision. The decision is the team's final decision made by the team leader.

By varying the competence model for different agents in terms of domain expertise, meta-knowledge about the other agent's expertise, member judgment, agent type, cooperativeness, and activity, the team performance model can be varied and tested. Examples of two studies of team decision-making that have been done with Team-Soar are:

- To examine the relationship of a team decision scheme used and the amount of information available to teams with measures of team effectiveness.
- To explore the relationship of meta-knowledge (knowledge about the knowledge of the other agents) and the amount of communicated information with how long it takes for the team to reach a decision.

2.4.4 Discussion

COT research studies theories of organizational behavior by creating *simplified* models of real-life organizational systems. The objective of such models is not to create a representation of how people really perform the work or task; instead the objective is to create a controlled experiment to test a theory. Ironically, the researchers that are using a distributed multiagent version of Soar (such as Plural-Soar, and Team-Soar) argue that they use Soar because it claims to be a computational architecture for human cognition. A multiagent Soar architecture provides an implementation of the ACTS theory of distributed human problem solving, which allows the experimentation of subsequent theories on human distributed problem solving applied to human organizations.

When we look at the problem domains that are being studied, we can see that these studies involve either a very simplified version of a real-life organization (e.g. order fulfillment), or an organization, like the military, that works according to very stringent and pre-defined procedures. In contrast, the objective of the research

described in this thesis is to create a computational architecture to study the way people work in *real-life* organizations. The focus of the research is on the work practices that exist in a workplace, and not so much on the cognitive problem-solving behavior of individual agents, or on the study of optimal organizational structures given certain constraints. The computational models that we are after are models that, at the micro-level, can describe a person's daily activities and interactions with his or her environment and others, and at the macro-level show an organizational behavior that can be interpreted as the work practice existing in that organization. Our models are not meant as an experiment of organizational theory, but instead, as a laboratory to study work practices of people in real-life organizations.

Just as in the previous two chapters, I conclude here as well with a table showing the limitation of COT (see Table 2-4).

Table 2-4. Limitations of Computational Organization Theory

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
Plural-Soar	Yes Represented as the communication between agents to help in picking items from the warehouse	No Only high-level goal/tasks of picking orders	No	No When contexts are switched tasks are quit	No Agents <i>only</i> interact with other agents about task related issues	Yes	Yes But, no geographical reasoning
Team-Soar	Yes But, <i>only</i> represented as communication of specific attributes using the hierarchical organization of the CO's	No finding values for the eight variables. Agents do <i>not</i> decide themselves what to work on	No	No	No	Yes	No Only values of the variables related to the location of the incoming missile

2.5 CONCLUSION

I conclude my description of related work with some general observations about the different modeling and simulation tools used in the different research fields. My objective here is to take this back to the research topic of this thesis and relate these tools to the requirements for modeling work practice. To do this, I generalize the data from Table 2-1, Table 2-2, Table 2-3, and Table 2-4 and give you an overview of the analysis.

Collaboration

The tools and models described, either do *not* represent the collaboration between people, or have a *limited* representation. Some of the agent-based models allow for some indirect representation of collaboration through hard-wired communication channels and/or communication message content. But, only the workflow modeling tool (Sparks) has an explicit form of showing two tasks (using a spawn) being performed in parallel. However, due to the actual semantics, this type of parallelism turns out to not always perform tasks in parallel.

In modeling how tasks are performed in real-life organizations, being able to represent collaboration between people is a necessity. However, what is meant with collaboration is not immediately clear. What is clear is that this is a topic that has not been well defined yet in any of the research fields mentioned here. In the theory chapter (chapter 3) I will return to the question of what is meant with collaboration.

Off-task behaviors

None of the tools and models that were described represent off-task behaviors. The workflow paradigm has a representational limitation in not being able to allow for representing off-task behavior. This is because of the constraint on representing only those tasks where work-products are being touched. The other tools and models all take a goal-driven approach, and therefore do not allow for tasks outside of the domain goal/task hierarchies. ACT-R is the only tool that would allow for the representation of off-task behaviors, by using its ability to allow for reactive agent-behavior through input from the outside world. In general, the field of cognitive science does not focus on the influence of off-task behavior on the problem-solving capability of humans.

The one observation that can be made from this is that in these research fields there is no explicit focus on how people behave in real-life tasks being performed in real-life organizations. If this would be the case, we would see a lot more interest in representing the influence of off-task behaviors in the performance of tasks. If we want to model how people really work we do need to include the effect of off-task behavior in the model, since it is very obvious that people are constantly interrupted by the need to perform tasks that are not part of the explicit work. For example, just think about the influence of getting a phone call, while performing a task, or the scheduling of group meetings in organizations, and how this impacts the “rhythm” of our work.

Multi-tasking

By allowing interruption and resuming of tasks we can approximate multi-tasking. Only the Phoenix system (chapter 2.3.2) allows for such interleaving of multiple tasks. Only Sparks allows for the execution of actual parallel tasks. However, the question is what the meaning is of performing tasks in parallel. In Sparks the purpose is for showing percentage of resources being associated with parallel tasks. If we want to represent one resource performing two tasks in parallel—driving a car, while being on the phone—we can use a spawn of the flow. However, in Sparks it is not possible to associate specific resources with a task. Resources are picked from a resource pool. Only if there is one resource in the pool can we be sure that the parallel tasks are actually performed by the same resource in parallel.

Although people can do multiple things in parallel, and we do need to be able to show this, the actual way people perform most parallel tasks is by switching between them in very short time intervals. A lazy skeletal planning approach, in which the commitment of what task to work on next is delayed as long as possible is one approach to allow for this form of multi-tasking. Task-priorities is a way to decide what task to work on

next. Most parallel activities can actually be seen as hierarchical. With this I mean that showing someone doing two things at once can be represented hierarchically, in the sense that the person is in activity $A_{1,1}$, while also being in activity A_1 . The on-phone-while-driving example could be thought of as such a hierarchical multi-tasking event.

Interrupt and resume

As pointed out above, only the Phoenix system allows for interruptions and resuming of tasks through a lazy skeletal planning approach. This is an essential part of work practice. We, humans, are constantly interrupted in what we are doing. When an interruption happens we do not stop a task we are working on to start another unrelated task and restart the original task when we come back to it. We interrupt tasks to come back to them where we left off when possible or wanted. This type of reactive, and unplanned interrupt and resume behavior is natural, and is part of the reason why we humans are not brittle in our task performances. Therefore, if we want to model work practice, the ability to represent interrupted and resumed activities is a must.

Informal interaction

In none of the tools and models are there representations of informal interaction between agents. Every tool and model described represents only the formal organization and tasks. Resources and agents only interact when the task being modeled asks for it. In real-life organizations there is an abundance of informal interactions between colleagues. For example, having lunch at work with a group of people is an informal activity (i.e. no formal work-task is being performed). However, during lunch there could be a lot of work related communication going on. Therefore, in modeling work practice it is essential to allow for the representation of informal tasks and communications and informal group behavior.

Cognitive behavior

There is a sharp distinction between models and tools that do or do not include cognitive behavior. The interesting observation to make is that there seems to be an either-or approach to this. I mean, either the models are totally reliant on deep cognitive problem-solving behavior, to the point that every cognitive-cycle is being represented, or the models are at a level where the cognitive ability of the individual agent is not represented at all.

The question in representing work practice is, what level of cognitive behavior representation is important. It seems that the low-level problem-solving behavior of Soar and ACT-R are not necessary relevant in showing the relationship between people's activities in a work process. On the other hand, it seems important to represent each agent in the process, and the agent's knowledge of when and how to perform tasks.

Geography

There is a range in the representation of geography in the models and tools described. The range is from *no* representation (in Sparks and Soar) to a *simple* abstraction (in Phoenix). None of the tools and/or models have a very detailed explicit representation of locations and spaces. The only system that has a separate geography model (i.e. the map-layer) is the Phoenix system (see chapter 2.3.2.2). In the other models and/or tools in which an agent's location is somehow represented, it is done through an indirect representation of the agent *knowing* about location. However, there is no explicit objective representation of location and space.

People's environment impacts their work. In modeling work practice, it is important that we have an explicit representation of the location of people and their artifacts. In ACT-R there is an explicit representation of the outside world, but this representation is domain specific and is *not* a part of the ACT-R modeling language. If we want a language for work practice modeling we need to have, at minimum, the capability of representing the outside world inside the model.

I end with a comparison between the tools and models in the four research fields. Table 2-5 lists the domain dependency, technology, environment, communication-, problem-solving and group interaction model for

each human behavior tool and model described. One last interesting observation is that in workflow and cognitive modeling research there is a tendency to develop generic modeling tools, while the DAI and COT research fields have a tendency to use the tools from the other two fields for developing their models. In doing so, the tools are being applied in ways they were not specifically designed for. This leads to interesting extensions and changes. What this shows is the benefit of a multi-disciplinary approach to science, as well as some of its shortcomings by the mere fact that we can never include or re-use *all* of the theories from other academic fields.

Table 2-5. Human behavior model comparison

		Domain Dependency	Technology Used	Environment Model	Communication Model	Problem-Solving Model	Group Interaction Model
WFM	Sparks	General	Monte-Carlo discrete event simulation	None	None	None	Fixed (using spawns-tasks to show interaction between resources)
CM	Soar	General	Production System	None	None	Soar theory of cognition	None
	ACT-R	General	Production System	Fixed (using P/M interfaces)	None	ACT-R theory of cognition	None
DAI	TacAir-Soar	Dependent	Multiple Soar production systems	Fixed (representation of cockpit model using ModSAF)	Fixed agent content messages (through DIS network)	Soar theory of cognition	Fixed (agents interface though simulated cockpit in ModSAF)
	Phoenix	Dependent	Reactive planning (i.e. lazy skeletal planning)	Cellular automaton representation layer with low-level reactive behavior to environment	Fixed hierarchical agent content messages	None	No interaction between agents at the same level
COT	Plural-Soar	Dependent	Multiple Soar production systems	Fixed representation of the stack locations	Fixed communications of item locations	Soar theory of cognition, combined w/ ACTS theory	Varied based on social knowledge about other agents
	Team-Soar	Totally dependent	Multiple Soar production systems	Fixed attributes representing radar information for aircrafts	Fixed hierarchical agent content messages	Soar theory of cognition, combined w/ ACTS theory	Fixed attribute-level interaction with radar for tracking aircrafts