Appears in *Proc. of ESS'99, held in Erlangen, Germany, October 26-28, 1999*

Simulation for Document Print Management

Stefania Castellani and Jean Marc Andreoli
Xerox Research Centre Europe, Grenoble, France

Abstract

Intelligent scheduling and coordination of print tasks has become a major challenge for printshops decision-makers given the dynamic nature of the job requests coming from customers and other printshops. Entirely automated solutions have proven difficult to implement in a real life environment as in other production management scheduling problems. The reason is that the parameters an automatic system would have to deal with and the constraints relating them are not always easy to explicit, and also these parameters are subject to frequent changes. Furthermore, automated solutions do not easily let decision-makers interact and bring their non-formalized knowledge into the process. In this paper we shortly describe how a simulation tool like Zippin can help decision-makers, allowing them to flexibly model, interactively simulate and explore work processes. Decision-makers can use simulation results to produce job schedules, organize the work locally at their printshop, negotiate time and costs with the customers, prepare subcontracting offers to other printshops, decide which jobs are worth outsourcing etc.

Keywords: Simulation, decision-making, interactive scheduling.

1 Introduction

Intelligent scheduling and coordination of print tasks has become a major challenge for printshops using more and more sophisticated and dedicated pieces of hardware, given the dynamic nature of the job requests coming from customers but also from other printshops (esp. when printshops are organised in alliances). Entirely automated solutions have proven difficult to implement in a real life environment, as in other production management scheduling problems. The reason is that the parameters an automatic system would have to deal with and the constraints relating them are not always easy to explicit and also, these parameters are subject to frequent changes. Furthermore, automated solutions do not easily let decision-makers interact and bring their knowledge into the process. The kind of scheduling we consider here is not limited to assigning begin and end dates to a given set of tasks in order to satisfy some constraints or to optimize some cost function. It is also concerned with supporting the decision-maker (DM) in determining which sets of tasks are capable of achieving a given goal, and with letting him/her interactively work with the scheduler, choosing among alternative solutions. Interaction with field workers is of crucial importance to guide a computer assisted scheduling process. In fact, the scheduling process may take advantage of information that only the DM can provide using non formalized knowledge built from past experience and expertise, e.g. which customer to privilege if a choice has to be made. Also, the DM may have to decide among different ways of realizing a print job: it can be done in one run or be split into a combination of several pieces. The DM may choose to dynamically split a print job while it is executing, to allow a more urgent job coming from a major customer, and requiring the same resources, to be performed in time. Moreover, a DM may want to be able to decide if a job has to be performed locally or remotely, i.e. if the job has to be outsourced. In general, it may not be obvious to determine a priori which is the optimal way to achieve some initial goal (i.e. a print job). Instead, the DM would like to be able to simulate several alternatives, navigate through the space of alternatives, get a synthetic view of the scheduling possibilities in each case, in order to make informed decisions.
Our Zippin simulation tool aims at providing an environment of this kind. Zippin is both a "map-drawer" and a "map-viewer". It allows the DM to flexibly model a work process as a space of possible alternatives for achieving a goal or a set of goals; it also allows to interactively explore the work process space graphically and simulate the schedule for the various alternatives. The space of possible work processes is visualized through a number of graphical views, each presenting the process alternatives under a different aspect. The DM can interact with the views to modify the process space. The process model, underlying the space of work processes, can be modified in the midst of simulation: changing the model is experienced by the DM as yet another mode of interaction with

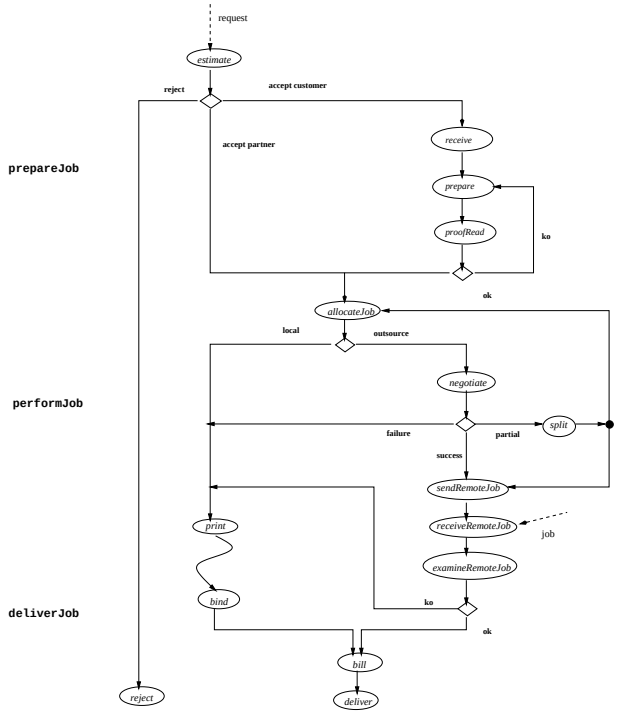

Figure 1: A printshop management scenario

the simulation. The Zippin scheduling facility searches for a solution to the resource allocation problem upon DM request. Decision-makers can use simulation results to produce job schedules, to organize the work locally at their printshop, to negotiate time and costs with the customers, to prepare subcontracting offers to other printshops, and to decide about which jobs are worth outsourcing.
Sec. 2 introduces the printshop scenario; Sec. 3 and 4 discusses how the simulation environment provided by Zippin allows a DM to model printshop activities and provides support for decision-making and process scheduling; Sec. 5 discusses related work; Sec. 6 concludes the paper.

2 A Printshop Management Scenario

We are interested in the management of the activities in a printshop, including its interactions with the customers and possibly with other printshops. More precisely, a printshop may receive print requests both from customers and from other printshops within an organised alliance. Conversely, the DM may want to outsource some of her/his print jobs to other printshops in the alliance. Figure 1 outlines the scenario. When a print request arrives with given parameters (e.g. description of job, delivery time, etc.) the DM estimates the workload it generates and makes a first evaluation of the options allowing the job to be performed, taking into account the job schedule, the availability of the resources, and trying to optimize the global cost. Using the results of this estimation, the DM may decide either to reject or to accept the print request. If (s)he makes the decision to reject the print request, (s)he must communicate that decision to the requester. If (s)he makes the decision to accept the print request (s)he must then allocate the job. However, if the request comes from a customer, the job has to be prepared before the allocation. First, a layout operator prepares the job configuring the right format, then a validator checks the resulting document. If the job has not been correctly configured the preparation is re-iterated. Once the job is ready, the DM will make the decision about where it has to be allocated. If the current schedule of the printshop allows inclusion of the new job, the DM can decide to perform it locally. However, it may happen that the job cannot be locally performed (at least not as a block), given the requirements, the printshop resource availability and technical capabilities, e.g. if the request includes a color print and the printshop has only black and white printers. Moreover, even if the execution of the job is compatible with the printshop schedule and equipment, the DM might still decide to outsource the job (or part of it) in order to save some of the available resources, e.g. for a job currently under negotiation with a major customer. In that case, (s)he must start a negotiation with the partner printshops for the outsourcing and decide if and how to split the job.

The outcome of a negotiation can be "success" (the job was fully outsourced), "failure"(no outsourcing agreement could be reached) or "partial" (only part of the job could be outsourced). An elementary negotiation scheme relies on an "invitation to tender". The DM collects offers from partner printshops, evaluates them and chooses the "good" solution (this is modeled in our scenario as a unique negotiate activity). The outsourced job is then sent to the selected remote sub-contractor making the best offer. It can also happen that none of the offers leads to a "good" solution: the DM can decide either to accept a sub-optimal offer anyway (and possibly face delays), or to re-allocate local resources in order to perform the job locally or to split the job. Locally or remotely, the job will be executed, i.e. printed and bound. If (a part of) the job has been remotely performed, a validator examines it, and, in case of problem, may try to re-print the faulty parts locally. The assistant finally closes the transaction: the job is billed and delivered to the client.
The flexibility of the modeling framework underlying Zippin and its simulation functionality can play an important role here. The DM needs flexibility for dynamically modeling the way jobs should be performed. Also the DM has to negotiate cost and time with customers, manage the allocations of the jobs and negotiate jobs with the other printshops. A simulation tool like Zippin can help the DM in making decisions along these processes. Zippin allows a DM to model non-deterministic behaviors, e.g. conditional branching and versioning in the correction cycles. The DM can simulate alternative configurations for jobs and resources and evaluate benefits and drawbacks. Thus a DM can dynamically produce job schedules for locally organizing the work; make the decision whether to perform a job locally or to outsource it; use the simulation results for negotiating time and costs with customers; and evaluate offers made by partner printshops.

3 Modelling the Scenario with Zippin

3.1 The Process Description Language

The formalism we use in Zippin to represent our work processes is based on the Generalized Process Structure Grammars (GPSG) [6, 1, 2]. According to this approach, a set of grammar rules describe how work processes break down into sub-processes and under what conditions. For example, the process of accomplishing a customer request could be broken down into preparing the job, performing the job, and delivering the print. Each of these subprocesses could be further broken down using additional rules. Formally, the rules have the following syntax:

<*rule*>	=	<*activity*> '-->' <*activities*> '::' <*constraints*>
<*activities*>	=	<*activity*>
	|	<*activity*> <*activities*>

An activity may be an occurrence of a work unit ("task") or else been decomposed into further subactivities by one or more rules ("goal"). In both cases, an activity is represented as a feature term, i.e. a structure defining a set of attribute/value pairs. The goal in the left-hand side of a rule defines the matching conditions for the application of the rule. The right-hand side comprises the set of the subgoals or tasks of the matched goal and a set of constraints expressing dependencies among activity features, both in the head and the body of the rule. An arbitrary number of features can be associated to an activity. There is a set of predefined features for each task (e.g. start-date, end-date, duration) and goal (e.g. start date obtained as the start date of its earliest subgoals). For instance, the following rule shows how the goal of locally performing a print job decomposes into the printing and the binding tasks.

```
(1) | goal = localJob | --> | task = print    |   | task     = bind   | ::
    |                  |     | duration = 2    |   | duration = 1      |
    |                  |     | role = eng,line |   | role     = binder |
```

The estimated duration of the print task is 2 time units and it needs a print engine (eng) and a line (line) to be performed. The role feature specifies the type of resources (both human and machine) required by a task. The description of a complex process is decomposed into many rules, woven together by matching feature structures. The declarative rule-based approach enables incremental modification and elaboration of the process model.

3.2 Dependencies among Tasks

Zippin offers several ways to specify dependencies between activities, using the powerful constraint programming paradigm [18] that allows both a declarative and a procedural reading. Direct dependencies, expressed by the process definition, result from the structure of the rules: when a goal is decomposed according to a rule, it is assumed it chronologically covers all the subgoals generated by the rule. A priori, the subgoals are performed in parallel. The rule, however, may explicitly a set of constraints between the different subgoals, creating further

temporal and causal dependencies. This allows to model deadlines, sequential tasks and overlapping tasks [2]. For instance, adding the timing constraint "`print triggers bind`" in rule `(1)`, the DM can express the fact that the `print` activity "triggers" the `bind` activity, i.e. binding can start any time after printing has started. The DM can even specify how much binding will overlap with printing. For example, another constraint could be "`bind starts_after(50) print`" meaning that binding could start only after 50% of the printing has been performed. Explicit causal dependencies among the activity features can also be defined. In the next section we will see for example how arithmetic constraints are used to establish iterative loops. Indirect dependencies result from the competition among tasks for bounded resources: if two tasks require a common unique resource, they cannot overlap in time. The DM may request Zippin to look for a solution for the allocation of the resources, respecting given resource bounds.

3.3 Control Structures

The rule-based language offered by Zippin allows to set up a variety of control structures and relationships among the activities and among the objects the activities create or modify. This allows the DM to model non-deterministic behaviors, like the conditional routing of activities. For example, the following three rules express the proof-cycle for preparing a job.

```
(2) | goal = doJob | --> | task     = prepare  |  | task     = proofRead |
    | iter = I     |     | duration = 2        |  | duration = 1         |
    |              |     | role     = layoutOp |  | role     = validator |
    |              |     | iter     = I        |  | iter     = I         |

                           | goal = proofReadEval |  ::
                           | iter = I             |

constraint = {prepare precedes proofRead, proofRead precedes proofReadEval}


(3a) | goal   = proofReadEval | --> | goal = routeJob |  ::
     | iter   = I             |     | iter = I        |
     | status = ok            |     |                 |


(3b) | goal   = proofReadEval | --> | goal = doJob |  ::
     | iter   = I             |     | iter = I1    |
     | status = ko            |     |              |

constraint = {I1 = I + 1}
```

First a layout operator prepares the job and a validator proof reads the result (rule `(2)`). Depending on the outcome of the proof reading the job is routed for printing or goes through another phase of preparation. If, the job has been correctly configured during the preparation (rule `(3a)`), it is routed for printing. Otherwise (rule `(3b)`), the preparation is re-iterated.

3.4 Simulating a Process with Zippin

The GPSG formalism underlying the Zippin system allows to define a space of possible processes that the DM may explore for simulation and scheduling purposes. The DM proposes an initial goal and the system decomposes it according to the rules of the grammar. When several alternatives occur, the system suspends its decomposition process and waits for input from the DM, who can resume the simulation by specifying new constraints on feature values of activities. The DM can dynamically modify constraints and examine the resulting behavior. Hence, the simulated process is represented as a net of tasks and goals, with dependencies among them. At any point of time, Zippin displays the current state of the simulation, which may be seen as a partial (grammatically correct) sentence, through several views, acting as lenses focusing on different aspects:

- the graphic template editor shows the graphical definition of the process;
- the rules template editor shows the process grammar rules;
- the task view shows the parse tree of the current (partial) sentence, i.e. how the initial process has been decomposed into subprocesses and terminal tasks;

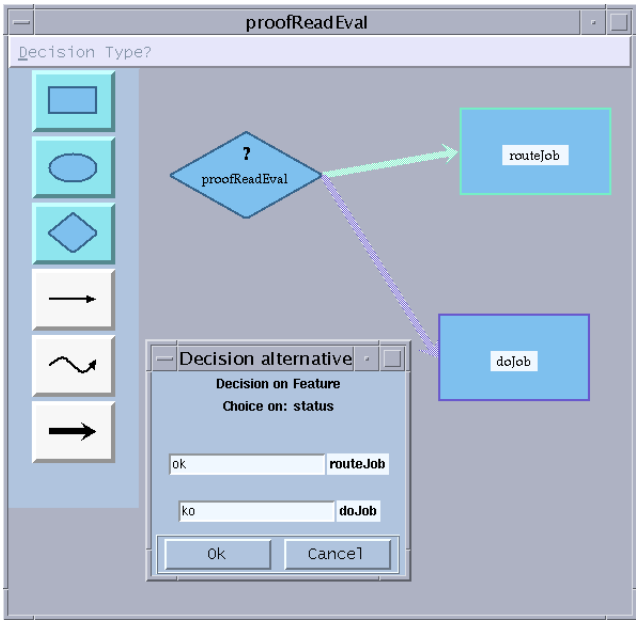

Figure 2: Proof reading decision point

- the time view shows the start and end times assigned to each activity, or, if these times are not assigned yet, the (finite) domains to which they are constrained;
- the resource view shows the amount of each resource required at any time;
- the session log shows the history of the DM interactions with the views, and allows to remove previous actions.

The DM interacts with Zippin through any of these views by adding and relaxing constraints. For example, the DM can further constrain (or assign) the start time of a task, add a task or a precedence constraint between two subprocesses in the process definition. Zippin manages a constraint store that maintains a consistent representation of the process definition, the state of the views, and the set of DM actions, represented as constraints. The DM can interact with his/her favorite view, and the system will update the other views. Also, the action of adding a constraint may trigger navigation in the process space. Typically, an action may wake a goal, which was suspended, because it can now be uniquely expanded. Zippin performs the expansion on the parse tree (thus reducing the process space). Ultimately, when all goals have been expanded, the overall process is fully described and the simulation/exploration ends. However, for navigation in the process space to be flexible, it must be non-monotonic: in Zippin the DM has the possibility to undo some of his/her actions, i.e. to relax constraints.
We include specialized scheduling strategies as independent tools which the DM may, at any time, invoke to test the current feasibility of a given process state w.r.t. resource bounds. For example, in the middle of a simulation, a DM may test "what would happen if I bounded such and such resources." A specialized tool would then attempt to solve the resource allocation problem with the given constraints, returning some new constraints which the DM may choose to keep or not for the rest of the simulation. Finally, from the Zippin system point of view, the grammar driving the exploration process is itself seen as a set of constraints that can be added or relaxed.

3.5 Simulating the Print Process

The first step when using Zippin is to create a description of the process. DMs design the process model using the graphic and/or textual process editor. The textual process editor gives the DM direct access to the underlying process modeling language (grammar rules). The graphic editor, on the other hand, allows novice DMs an easier entry point into modeling processes. The graphic description of a process contains a set of windows, which correspond to the goals in the process. The graphical editor automatically generates the set of rules used by Zippin to simulate the process. For example, Figure 2 shows the window for the proof reading decision point (proofReadEval). After specifying the process, the DM can initiate the exploration of the process. Zippin expands the grammar rules as far as it can without non-deterministic choices: further expansion requires input from the DM, e.g. to resolve decision points (should the job be performed locally or be outsourced?). The DM can then see the simulated process using the different views.

3.6 Synthetic Views

The task view (Figure 3) shows the current decomposition of the initial goal in a hierarchical structure (parse tree). Each node in the tree represents an activity. A task is represented as a circle labeled with the name of the task. A goal is represented as a square containing a question mark if it marks a decision point or by a flag if it marks an instantiated decision point. Otherwise, a goal is represented with overlapping circles. A goal marks a decision point if its decomposition is suspended because several alternative decompositions exist. The features of an activity are displayed when the corresponding node in the tree is selected. The DM can insert constraints on the current state of the activity. Figure 3 is an example of interaction through the task view. If the selected goal is a decision point (as in the example) there is at least one decision feature among its features, i.e. a feature whose instantiation determines which rule will apply to decompose the goal. For example, in Figure 3 the `proofReadEval` decision point has been selected, and the `status` decision feature has been instantiated to `ko`, triggering the application of rule `(3a)`. On the other hand, selecting the `ok` value would have triggered rule `(3b)`.
The time view (Figure 4) visually depicts the current constraints on the start dates and end dates of the tasks in the process. Each task is represented as a box the length of which is proportional to its duration. The left side of the box is the earliest possible start time of the task. The DM can enforce or relax timing constraints by moving the box for the task along the horizontal axis. It is also possible to return to previous configurations in the time view using backtrack points. Backtrack points are displayed as flags and save the previous earliest start values for a task.
The resource view shows the resource usage of the process activities. For each resource (human as well as material), the view shows the upper bound specified by the DM (by default 0) and its usage at each instant. Various colors (shades of gray on the black-and-white figure) shows the number of unit of each resource required at each instant as a percentage of the declared upper bound on that resource. Figure 5 shows an example of the resource view. Resource bounds can be tuned and the resource view is automatically updated.
Interactions with the views are logged and displayed to the DM. The DM can backtrack an interaction directly from a view but (s)he can also select from the log a group of interactions and remove them, thus backtracking to previous configurations of the simulation. Also, the DM can insert a checkpoint in the log to mark (or save) an interesting configuration of the simulation and come back to it later, clicking on the checkpoint. Simulations can be saved and reloaded later for further refinement or for re-use.

3.7 Resource Allocation: the Scheduler

Changing the planning of the tasks with the help of the views, the DM can manually schedule the allocation of resources to tasks, but (s)he can also ask the Zippin scheduler to search for a solution for the allocation of the resources to the tasks in the current state of the simulated process. We use a technique similar to the micro-opportunistic scheduling of [14]. Once given a set of resource bounds, the scheduler searches for an allocation of the resources, respecting the given bounds and the other constraints, while trying to minimize the execution time for the whole process. Each time a solution is found, it is displayed on the different views together with the information about the global execution time for the solution. The DM may either accept the solution currently displayed, or request further search for another solution, or decide to refine the simulation further. See [1] for more details on the scheduler.

4 Decision-making using Zippin

The flexibility of the modeling framework underlying Zippin and its simulation functionality can play an important role in our scenario. The DM needs flexibility for modeling the way jobs should be performed and Zippin allows for example to dynamically model non-deterministic behaviors, e.g. conditional branching in the decision whether to perform a job locally or to outsource it. In several occasions along the process of accomplishing the print requests, the DM has to make decisions on the allocation of the jobs. Zippin allows him/her to simulate job schedules with alternative configurations for jobs and resources and evaluate benefits and drawbacks.
As soon as a print request arrives, the DM can model the job processes associated to it, and decide to accept or reject it. If the job is accepted, the DM must then find a way of executing it, changing the current schedule if necessary. In some cases, the DM may want to see if, when and how a print request can be accepted. So (s)he may want to simulate the job process for the request, possibly generating several quotations, which can be used in the negotiation with the customer, thus offering several possibilities in terms of cost and time. Moreover, the DM can also use simulation results for decision-making during negotiation with other printshops. A printshop **Pa** may act as a subcontracting printshop candidate, among others, for the acquisition of a job outsourced by another printshop **Pc**. Then, **Pa**'s DM may simulate several ways of performing the job outsourced by **Pc** and possibly make several proposals in terms of time and costs. On the other hand, for accomplishing a print request, **Pa**'s DM

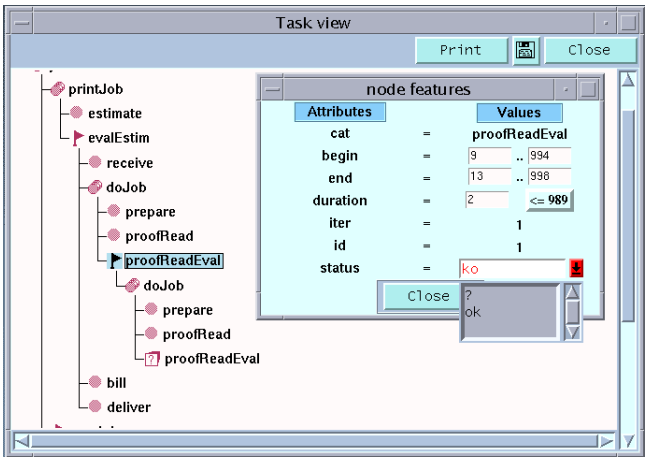

Figure 3: The task structure of the proof reading goal

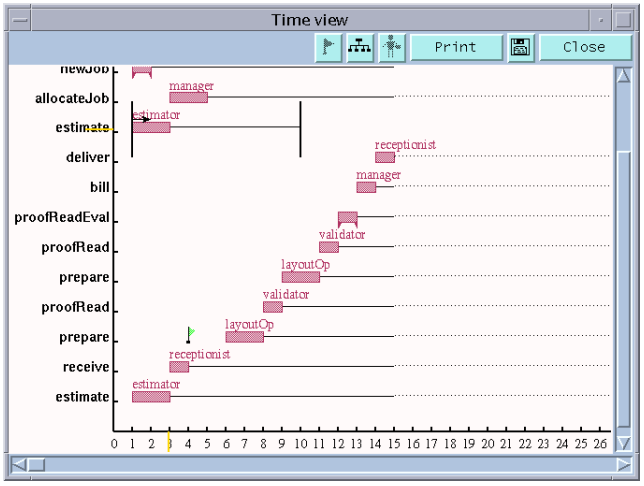

Figure 4: The time constraints of the tasks

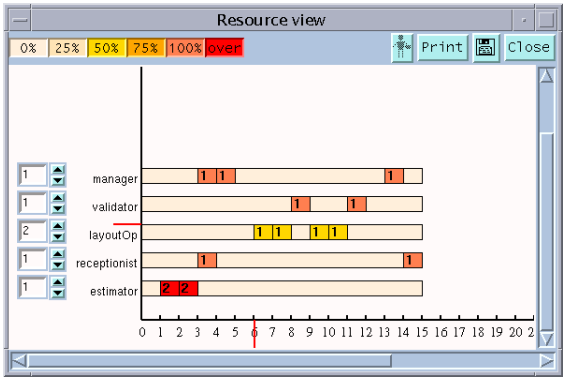

Figure 5: The resource usage of the tasks

may have to evaluate the benefits of adopting one of the alternatives among locally performing the job, outsourcing the job to another printshop **Pb** (e.g. because (s)he knows that **Pb** is making a special offer), splitting the job (e.g. because (s)he wants to keep free some time slots of some resources in **Pa**). Zippin can help the DM in modeling and simulating these alternatives.
Also, the DM may have to face a situation like the arrival of a job request from a major customer with a strong deadline. In that case, the DM may have to change the way jobs already being processed are performed, for example splitting a print job, being processed on a given machine, to allow the higher priority job, requiring the same machine, to be performed in time. Again, Zippin can help the DM simulate future executions before choosing among multiple possible evolutions of a running job process or before changing the job process description. In [1] we gave an example of how a DM could model and simulate jobs slitting in Zippin: print requests can be either simple requests (`simplePrint`), e.g. printing one piece of data, or complex requests (`formPrint`), e.g. printing a number of data sets (which share a large part in common: a form). Zippin could help the DM make decisions not only to determine when to schedule the requests, but also whether or not, and in the affirmative how, to split the complex requests, simulating several splitting (see [1] for more details). In our scenario, first rule (1) would be refined replacing the print task by a decision goal, say `typedPrint`, with a `type` decision feature denoting the type (simple or complex) of the print task. Then alternative decomposition rules (according to different values of the `type` feature) would allow to specify if a request is simple or complex.

5 Related Work

The use of constraints to flexibly describe work processes has also been explored in the Ariadne system [4] but this is done in the context of parsing a given sequence of actions (i.e. checking the process instance for "grammatical" correctness), without focussing on generating the space of potential work actions towards a goal. Constraints are also used to describe plans, processes and activities in systems like O-Plan [17] based on the I-N-OVA Constraint model of activity [16]. In particular, the O-Plan system focuses on mixed-initiative planning for realistic problems, especially for military logistics.
The ability to express and visually manipulate complex interdependencies among business objects and activities differentiates Zippin from commercial process modeling and simulation tools currently available on the market [5]. Commercial process modeling and simulation tools like Extend ([8], Imagine That!), ServiceModel (ProModel Corporation), iThink, and Enterprise Modeller (Business Integration Technologies Limited) perform a number of functions in common with Zippin, namely process modeling and visualization. These tools have the ability to provide user-customizable analysis mechanisms, reports and graphical plots of numerical simulation results. On the whole, when positioned in the domain of work processes, such tools are intended for business process re-engineering. This accounts for their focus on optimization of processes vs. Zippin's focus on exploring processes and process alternatives.
A number of workflow systems have incorporated several forms of flexibility in process modeling [13]. For example TeamWARE Flow, originated from the Regatta research prototype [15], allows independent subplans to be elaborated on the fly. The goal-based approach to task management implemented in Polymer was likewise developed to permit flexible enactment of work processes [10]. Also, some workflow systems provide simulation capabilities to study workflow design efficiency and to study workflow system performance and reliability [12, 11].
Systems and environments are being developed to provide some kind of support for interactive decision-making and scheduling processes, mainly letting decision-makers choose among several heuristics approaches [9, 7, 3].
Comparing Zippin to project management tools, e.g. Microsoft Project'98, we have found that they do not offer a customizable definition of the activities, the objects they produce and the relationships that can exist between them. Zippin provides a more flexible environment for work process objects representation, be they activities or deliverables, and the dependencies among them.

6 Conclusions

In this document we have shortly discussed how our experimental tool Zippin can provide decision-makers with support for interactive simulation, scheduling and decision making processes in a document production management context.
Current work under development includes the extension of the Zippin technology (with negotiation and tracking capabilities) to provide support for simulation, scheduling and decision-making in alliances of independent organization. For more information concerning the Zippin project, visit our Web-site at

`http://www.xrce.xerox.com/research/ct/prototypes/zippin/`

References

[1] J-M. Andreoli, S. Castellani, U. Borghoff, R. Pareschi, and G. Teege. Agent-based decision support for managing print tasks. In *Proc. of PAAM'98*, London, U.K., 1998.

[2] J-M. Andreoli, S. Castellani, and N. Glance. Zippin: towards a graphical decision support tool. In *Proc. of CE'98*, Tokyo, Japan, 1998.

[3] J. Davis and J. Kanet. Production scheduling: An interactive graphical approach. *Journal of Systems Software*, 38:155–163, 1997.

[4] G. Florijn, T. Besamusca, and D. Greefhorst. Ariadne and hopla: Flexible coordination of collaborative processes. In *Proc. of the First Int. Conf. Coordination'96*, pages 197–214, Cesena, Italy, 1996.

[5] N. Glance, S. Castellani, and D. Pagani. Simulation for optimization vs. role-playing: A case study of a document authoring process. In *Proc. of PAKM'96, Workshop on Adaptive Workflow*, Basel, Switzerland, 1996.

[6] N. Glance, D. Pagani, and R. Pareschi. Generalized process structure grammars (GPSG) for flexible representations of work. In *Proc. of the 7th Europ. Conf. on Computer-Supported Cooperative Work*, Boston, Ma., U.S.A., 1996.

[7] R. Goodwin, S. Murthy, J. Rachlin, and R. Akkiraju. Interactive decision support: Advantages of an incomplete utility model. In *Proc. of the AAAI Spring Symposium on Interactive and Mixed-Initiative Decision-Theoretic Systems*, Stanford, Ca, U.S.A., 1998.

[8] G. Hansen. *Automating Business Process Reengineering: Breaking the TQM Barrier*. Prentice Hall, 1994.

[9] P. Higgins and A. Wirth. Interactive scheduling. In *Proc. of APORS'97*, Melbourne, Australia, 1997.

[10] D. Mahling, N. Craven, and W. Croft. From office automation to intelligent workflow systems. *IEEE Intelligent Systems*, 10(3), 1995.

[11] J. Miller, A. P. Sheth, and K. Kochut. Perspectives in modeling: Simulation, database and workflow. In *Proc. of Winter Simulation Conference*, pages 612–619, Arlington, Va, U.S.A., 1995.

[12] J. Miller, A. P. Sheth, K. Kochut, X. Wang, and A. Murugan. Simulation modeling within workflow technology. In *Proc. of ER'97 Symposium*, Los Angeles, Ca, U.S.A., 1997.

[13] C. Mohan. Tutorial: State of the art in workflow management research and products. In *Proc. of ACM SIGMOD International Conference on Management of Data*, Montreal, Qu, Canada, 1996.

[14] N. Sadeh. Micro-opportunistic scheduling: The micro-boss factory scheduler. In M. Zweben and M. Fox, editors, *Intelligent scheduling*. Morgan Kauffmann Publishers, San Francisco, Ca, U.S.A., 1994.

[15] K.D. Swenson, R.J. Maxwell, T. Matsumoto, B. Saghari, and K. Irwin. A business process environment supporting collaborative planning. *Collaborative Computing*, 1(1):15–34, 1994.

[16] A. Tate. Representing plans as a set of constraints - the I-N-OVA model. In *Proc. of the Third Int. Conf. on Artificial Intelligence Planning Systems*, Edinburgh, U.K., 1996.

[17] A. Tate, B. Drabble, and R. Kirby. O-plan2: An open architecture for command, planning, and control. In M. Zweben and M. Fox, editors, *Intelligent scheduling*. Morgan Kauffmann Publishers, San Francisco, Ca, U.S.A., 1994.

[18] P. van Hentenryck and V. Saraswat. Strategic directions in constraint programming. *ACM Computing Surveys*, 28(4):701–726, 1996.