Disruption-Driven Resource Rescheduling in Software Development Processes

Junchao Xiao[1,2], Leon J. Osterweil[2], Qing Wang[1], and Mingshu Li[1,3]

[1] Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] Department of Computer Science University of Massachusetts, Amherst, MA 01003-9264 USA
[3] Key Laboratory for Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

Abstract. Real world systems can be thought of as structures of activities that require resources in order to execute. Careful allocation of resources can improve system performance by enabling more efficient use of resources. Resource allocation decisions can be facilitated when process flow and estimates of time and resource requirements are statically determinable. But this information is difficult to be sure of in disruption prone systems, where unexpected events can necessitate process changes and make it difficult or impossible to be sure of time and resource requirements. This paper approaches the problems posed by such disruptions by using a Time Window based INcremental resource Scheduling method (TWINS). We show how to use TWINS to respond to disruptions by doing reactive rescheduling over a relatively small set of activities. This approach uses a genetic algorithm. It is evaluated by using it to schedule resources dynamically during the simulation of some example software development processes. Results indicate that this dynamic approach produces good results obtained at affordable costs.

Keywords: Incremental resource scheduling, time window, reactive rescheduling, proactive rescheduling.

1 Introduction

Complex systems are typically comprised of a group of activities, each of whose executions requires different resources that have various capabilities and availabilities. Careful resource scheduling can reduce resource contention, thereby reducing delays and inefficiencies, and in doing so, can increase the value of the system [1-3].

A lot of work has investigated different approaches to determining optimal schedules of assignment of resources to system activities. One approach is static resource scheduling [1, 2, 4-7], in which a complete schedule of resource assignment is computed in advance based on advance knowledge of the sequence of activities to be performed and the size and duration of all these activities [8, 9].

However, in disruption prone systems like those used in software development and healthcare, activity estimates may unreliable, and uncertainties such as the sudden

J. Münch, Y. Yang, and W. Schäfer (Eds.): ICSP 2010, LNCS 6195, pp. 234–247, 2010.

arrival of rush orders, unexpectedly slow activity performance, inaccurate estimates, and unexpected lack of resources [10] are numerous. These uncertainties dynamically change the execution environment and create the potential for consequent schedule disruptions [11, 12], thus making scheduling results unreliable and uncertain. In response to such uncertainties, researchers have studied different kinds of dynamic resource scheduling approaches [11], such as robust scheduling [13-16] and reactive scheduling [17, 18]. Though these methods seem effective in addressing some uncertainties, they have the following problems:

- Robust scheduling is most effective when there are limited and predictable disruptions in process executions. If exceptions exceed the scope that a robust approach can deal with, rescheduling is needed, and this becomes a reactive scheduling approach.
- Reactive scheduling addresses unexpected events and other uncertainties by responding with a rescheduling. But each rescheduling covers all possible future activities. Thus it can be expected that the rescheduling results will encompass a lot of uncertainties that will probably cause future disruptions that will necessitate further reschedulings.

These issues seem particularly troublesome in fast changing environments such as software development, where process execution plans may need to change continually. To tackle this sort of problem, we have suggested (in [19]) the use of planning incremental rescheduling, where rescheduling was expected to happen frequently, and over only a modest-sized scope. In [19] we assumed that accurate estimates for the execution times of tasks, and precise definitions of software process steps are available to guide such incremental rescheduling activities. In that work, reschedulings were undertaken proactively, and unexpected disruptions such as process delay and future activity changes, were not taken into account. But in actual practice, such events are common and should indeed be used to trigger reschedulings. Thus, in this work, we address the need to deal with such disruptive events.

In this work we present an approach that supports both proactive incremental rescheduling of resources, and reactive rescheduling of resources in response to the occurrence of unexpected disruptive events. This paper extends the dynamic resource rescheduling method presented in [19] and proposes a ***T**ime **W**indow based **IN**cremental resource **S**cheduling method* (**TWINS**). The method decomposes the overall resource scheduling problem into a series of dynamic reschedulings continuously happening either proactively or reactively and covering only selected subsets of activities. A genetic algorithm (GA) [20] is used as the basis for our scheduling approach. In addition to its speed, another advantage of GA is that it can readily incorporate many types of constraints into the definition and solution of the scheduling problem.

We have implemented TWINS and evaluated its effectiveness using different time windows and different disruptions to generate different resource allocation schedules. The results obtained show that TWINS is effective in dealing with the needs for both short-range schedules that can be quite accurate because they have fewer uncertainties to deal with, and long-range schedules that must deal with greater uncertainties, yet still be highly accurate.

The paper is organized as follows. Section 2 presents the TWINS framework. Section 3 presents some details of the components and technologies used in our TWINS implementation. Section 4 describes a simulation of a software development process and reports on some case studies aimed at evaluating this approach. Section 5 presents conclusions and suggests future work.

2 Time-Window Based INcremental Scheduling (TWINS)

TWINS decomposes a large end-to-end resource scheduling problem into a series of dynamic reschedulings. The rescheduling in TWINS can be triggered either 1) reactively, when events occur that are beyond the scope of what we have been able to anticipate, or preferably, 2) proactively, at time points that may be dictated by recognition of upcoming uncertainties derived from analysis of precise and detailed system definitions. Each incremental rescheduling activity covers only the tasks that will occur within a specified time window.

The time window can be a given time interval or a specified number of activities. It is important to select the right size for this window. If the window is too small, more frequent reschedulings may be needed, but they may produce more accurate results. If the window is large, scheduling may cover more activities, but scheduling cost may be high, and more disruptions that require new rescheduling are likely to be encountered during execution of steps that have been included in the scheduling window. Indeed it may be the case that the size of the scheduling window may have to be estimated differently depending upon whether the rescheduling was undertaken proactive or reactively. The TWINS method shown in Fig 1 has also been presented in [19]. We summarize it here for completeness. The method consists of the following major components:

- **Scheduling activity set constructor.** This component selects activities from a process system and assembles the rescheduling problem, which consists principally of a specification of the activities that may possibly be executed in the near future (within the time window). The scheduling activity set constructor relies upon a specification of the system.
- **Scheduler** component, which uses the output of the scheduling activity set constructor and a Genetic Algorithm (GA) to identify the specific resources to be used to support the execution of each activity. A repository of specifications of available resources is also used to support this scheduling.
- **Rescheduling indicator** component, which determines when rescheduling should be done. Rescheduling is triggered when the rescheduling indicator determines that activities outside the window needed to be executed. This component could also be used to identify when certain types of unexpected events make rescheduling desirable or necessary.
- **System execution** component, which provides execution events needed to update the system execution state upon which the rescheduling indicator and the scheduler rely.

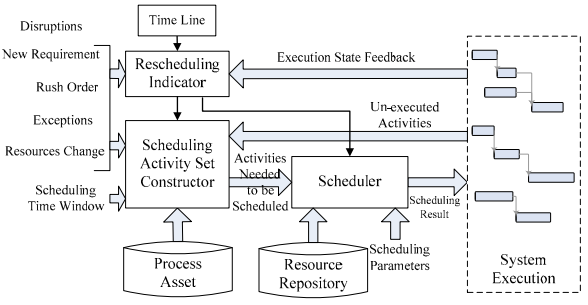

Fig. 1. The TWINS Method

3 TWINS Implementation

3.1 Process Activity Definition

A complete, precise, and detailed system definition can improve the quality of the resource scheduling need to support execution of the processes in that system. As in [19], we also choose Little-JIL [21, 22] to define the system for which we will do our scheduling. A Little-JIL process definition consists of a specification of three components, an artifact collection (not described here due to space constraints), an activity specification, and a resource repository. A Little-JIL activity specification is defined as a hierarchy of steps, each of which represents an activity to be performed by an assigned resource (referred to as its agent). Each step has a name and a set of badges to represent control flow among its sub-steps, its interface, the exceptions it handles, etc. A leaf step (one with no sub-steps) represents an activity to be performed by an agent, without any guidance from the process. Each step specification also contains a collection of resource requests. Each request in the collection is described by the following definition.

Definition 1. $Req = (ResName, Capability_1, SkillLevel_1, ..., Capability_r, SkillLevel_r)$ **, where,**

- *ResName* is the type of the resource being requested, (e.g. requirement analyst, coder, tester), which indicates the kinds of capabilities that this resource has.
- $Capability_i$ is a capability that is being requested.
- $SkillLevel_i$ is the minimum level of skill in $Capability_i$ that is being requested.

Fig 2 shows a Little-JIL activity definition that defines a software development process. Note that a different instance of this process is instantiated for every new project, and thus the work of a typical development organization is represented by the concurrent execution of several

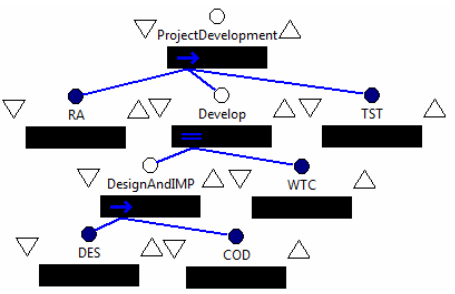

Fig. 2. Module development process

instances of this process. Each process needs the same types of resources, which must be provided by one central resource repository. This sets up resource contention. Each process instance is represented by the top step, "ProjectDevelopment". The right arrow in this step specifies that, in sequential order, a project is developed by requirement analysis (RA), development (Develop), and testing (TST). Development is further decomposed into design (DES), coding (COD), and writing test cases (WTC). WTC is executed in parallel with DES and COD (represented by "=" sign).

3.2 Resource Repository

The resource repository is the second key component of a Little-JIL process definition. It contains a set of resource instances that are available for assignment to the tasks specified in the Little-JIL activity diagram.

Thus, $ResourceRepositiory = \{Res_1, Res_2, ..., Res_l\}$, where each element of this set has certain capabilities and availabilities. A resource is defined as follows:

Definition 2. $Res = (ID, ResName, Attributes, SchedulableTimeTable, Capability_1, SkillLevel_1, Productivity_1, Capability_2, SkillLevel_2, Productivity_2, ...)$

where,

- *ID* is a prose identification of the resource.
- *ResName* is the type of the resource, which indicates the kinds of capabilities that this resource has, such as software analyst and designer.
- *Attributes* is a set of (*name*, *value*) pairs that describe the resource. Some example attribute names might be *Age*, *Experience_Level*, and *Salary_Rate*.
- *SchedulableTimeTable* is a representation of the times when a resource is available to be assigned to an activity. This consists of a set of schedulable time intervals, defined by a start time (*st*) and end time (*et*). The resource can be assigned to an activity during any of the specified time intervals. Thus,

$$SchedulabeTimeTable = \{[st_1, et_1], [st_2, et_2], ..., [st_s, et_s]\}$$

- $Capability_i$ (i = 1, 2 …) is the i^{th} kind of capability that the resource has to offer. For example, a coder has the capability of using JAVA for coding.
- $SkillLevel_i$ (i = 1, 2 …) is the level of quality at which the resource is able to perform $Capability_i$.
- $Productivity_i$ (i = 1, 2 …) is the average productivity that the resource is able to achieve in performing $Capability_i$.

Assume that an activity specifies that *S* is the quantity of $Capability_i$ required in order to complete the activity. Then $S/Productivity_i$, is the amount of time that resource *R* requires in order to complete the activity. Only if this amount of time is contained within R's *SchedulableTimeTable* attribute, can *R* be assigned that activity.

3.3 Scheduling Activity Set Constructor

The Scheduling Activity Set Constructor assembles all of the information needed in order to make scheduling decisions. This function determines which upcoming activities fall within the scheduling time window (whose size has been previously determined),

and assembles the activities into a graph called the Dynamic Flow Graph (DFG). The DFG is derived from an analysis of another graph called the resource utilization flow graph (RUFG). Space limitations prevent a full discussion of how these graphs are built and used, and so we summarize descriptions of them. Details describing the DFG and RUFG are provided in [23]

The RUFG is a directed graph, derived from a Little-JIL activity diagram, which represents all the possible execution sequences of the process. Each node in the RUFG represents a step in the Little-JIL process definition and is annotated with precise and detailed information about the resource needs of the step. Thus, the RUFG is a representation of the static structure of the process.

Our rescheduling approach, however, uses dynamic process state information to take dynamic constraints and uncertainties into account. Thus when a rescheduling is needed we use the static RUFG and dynamic state information to generate a dynamic flow graph (DFG) that is then used as the basis for the rescheduling. The DFG incorporates both the RUFG's static information and runtime state information such as which activities are currently executing in parallel, which resources are allocated to them, and the priority level assigned to each activity.

The size and shape of the DFG is determined by a specification of the time window, which dictates how many of the future execution possibilities are to be considered in the rescheduling. At present we define the size of a time window by an integer that represents the length of the longest sequence of possible future activities that make resource requests. Thus, specifically, if L is the integer used to define time window W and $CURRACT$ is the set of activities that are currently being performed,

$$CURRACT = \{activity_1, activity_2, ..., activity_n\},$$

then node $NODE$ is included in W if and only if $NODE$ requires the allocation of resources, and, for some i, $1 \le i \le n$, there is a path, P, in the RUFG

$$P = (activity_i, n_1, n_2, ..., n_k, NODE)$$

such that the number of nodes n_j, $1 \le j \le k$ that make resource requests is less than L-1.

Further details about the definition of the RUFG and DFG are omitted due to space constraints. In addition, note that other approaches may be used to define the scheduling window, and the one just defined is used here as an example.

3.4 Using a GA to Do Resource Scheduling

By being careful to keep time windows small we can assure that our scheduling problems remain relatively tractable. We nevertheless expect that our windows may still be large enough to contain quantities of activities and resources that are sufficiently large to require considerable scheduling computation. Thus in this work we have extended the GA designed in [19] as the basis for each rescheduling.

3.4.1 Encoding and Decoding

We used the binary representation of integers to help encode the rescheduling problem as a chromosome. Note that because the set of DFG nodes waiting to be scheduled changes during process execution, new chromosomes must be built for each

rescheduling. For each DFG node, N, in a time window (i.e. each node waiting for the assignment of resources), we establish a set of resource genes and a set of priority genes. Suppose N requires m resources. Then each of the m resource requests that have B_m candidate resources is encoded as a set of binary genes (i.e. genes whose value can be only 0 or 1), where the size of the set is the smallest integer greater than or equal to $\log_2 B_m$. The binary values of these $\log_2 B_m$ genes are used to represent the decimal number of a candidate resource, namely one that could be selected to satisfy the corresponding request. Similarly, the priority level of the node is represented by a set of g binary priority genes. When two or more DFG nodes contend for the same resource, the node whose priority genes specify the highest priority is assigned to the resource. The structure of such a chromosome is shown in Fig 3.

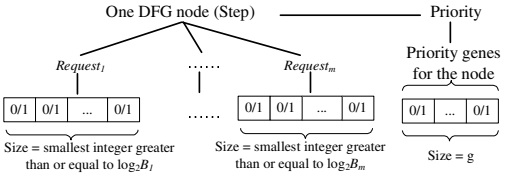

Fig. 3. Chromosome structure

Each chromosome encoded by the above method can, subject to the application of constraints, be decoded as a scheduling scheme, namely the assignment of a specific resource to each of the requests made by each of the activities in the time window. Decoding is done by reversing the encoding process.

3.4.2 Fitness Function

The role of the fitness function is to evaluate the relative desirability of each of the chromosomes as a solution to the resource rescheduling problem. The fitness function to be used must be designed to meet the needs of each individual scheduling problem. The fitness function used in evaluation in this paper was first presented in [19]. It was created to address resource allocation in the domain of software development. As this is the domain of evaluation in this paper too, we reuse this fitness function in this paper too. Note that this fitness function does not attempt to increase the value for all steps of all projects, only the value for the steps that are involved in the time window.

3.5 Rescheduling Indicator

The rescheduling indicator determines the circumstances under which a rescheduling should be carried out based on such runtime state information as the activities currently being executed, the resources being used, resource capacity that is available, and different kinds of uncertainty-causing events. Research is needed to determine the rescheduling criteria that should be used for each of the many different kinds of resource allocation problems. And, indeed, different criteria may trigger reschedulings based upon time windows of different sizes, and rescheduling decisions may be made differently under different execution circumstances. What follows are examples of events that could be used in dynamically determining whether a rescheduling should be performed:

- **Delay of currently executing activity:** if the currently executing activity is delayed, and it causes other activities that use the same resources to be unable to obtain the resources that they need at their starting time, a rescheduling should be carried out.

- **Identifying changes to a future activity:** a future activity is an activity that has not yet been executed. If during execution of one activity it is determined that changes will cause a future activity to have insufficient resources allocated to it, then a re-scheduling should be carried out.
- **Arrival of new requirement:** as soon as the need to meet a new requirement is realized, a rescheduling to allocate resources to this new requirement must be carried out.
- **Un-assigned allocation of successive activities:** once an activity has completed, if any of its successor activities has not been allocated resources, a rescheduling should be carried out.

Note that the first two of these items characterize unexpected runtime situations that are necessitate relatively more disruptive reactive scheduling. The last two items characterize predictable situations that trigger proactive rescheduling. This proactive rescheduling is more desirable as its results should be expected to be less disruptive and more accurate.

4 Evaluation

To support the analysis of the effectiveness of our approach, we used it to allocate resources during the running of simulations of processes that define how software project developments perform their activities and utilize their resources.

4.1 The Simulation Setting

4.1.1 Process, Projects, and Resources

The process that was used as the principal basis for the case study presented here is the Little-JIL process shown in Fig 2. Note that non-leaf steps are used essentially to create scopes, and "real work" is done only by leaf steps. Thus, the resource requests are shown in Table 1 only for each leaf step in this process. And each step only needs one resource.

We assume that there nine projects named P1 to P9 that are executing simultaneously in this case study and each project is executing a different instance of the same process. Assume the scale of these projects are all 8KLOC, the cost constraint for each project is $50,000, the benefit of finishing ahead of deadline is $400/day, and the deadline delay penalty is $800/day. Projects P1 to P3 can start on 2009-01-01, and must finish by 2009-06-30. Projects P4 to P6 can start on 2009-04-01, and must finish by 2009-09-30. Projects P7 to P9 can start on 2009-07-01 and must finish by 2009-12-30.

Table 1. Resource requests of leaf step

Step	Request (ResName, Capability, SkillLevel)
RA	(Analyst, OOA, 3)
DEC	(Designer, OOD, 3)
COD	(Coder, JAVA, 3)
WTC	(Tester, WTC, 3)
TST	(Tester, TST, 3)

The different kinds of resources available in the case study include analyst, designer, coder, and tester. They are described in Table 2. We use dates for *SchedulableTimeTable* entries, and in each weekday, the available workload is eight person-hours, while on Saturday and Sunday, the available workload is zero.

Table 2. Available resource descriptions

ID	Name	Human Name	Schedulable Time Table	(Capability, Skill Level, Productivity)	Salary-Rate ($/hour)
1	Analyst	HR1	[2009-01-01, 2011-12-31]	(OOA, 4, 0.06)	70
2	Analyst	HR2	[2009-01-01, 2011-12-31]	(OOA, 4, 0.04)	45
3	Designer	HR3	[2009-01-01, 2011-12-31]	(OOD, 4, 0.06)	60
4	Designer	HR4	[2009-01-01, 2011-12-31]	(OOD, 4, 0.05)	60
5	Coder	HR5	[2009-01-01, 2011-12-31]	(JAVA, 4, 0.04)	45
6	Coder	HR6	[2009-01-01, 2011-12-31]	(JAVA, 3, 0.03)	35
7	Tester	HR7	[2009-01-01, 2011-12-31]	(WTC, 4, 0.05), (TST, 4, 0.05)	45
8	Tester	HR8	[2009-01-01, 2011-12-31]	(WTC, 4, 0.03) , (TST, 4, 0.04)	40

4.1.2 Resource Contention and Different Kinds of Events

For our case study, we assumed that all projects could be pursued simultaneously, but that only limited resources were available and each resource could perform only one step at the same time, thereby creating significant resource contention.

TWINS constructed its scheduling problems by assembling in its time window all activities for all the projects that fall within the time window specification. For this case study we hypothesized three "*arrival of new requirement*" events, thus causing three predictable reschedulings. But we did not hypothesize any "*un-assigned allocation of successive activities*" events that would have triggered predictable reschedulings. On the other hand, we hypothesized the following unpredictable events in the simulation, using as an example the disruption caused by a 20% increase in execution time.

- **Delay of currently executing activity**:
  - When COD of P1, P2, and P3 is executing, the time for executing it increases by 20% over the initial plan;
  - When TST of P4, P5, and P6 is executing, the time for executing it increases by 20% over the initial plan;
  - When DEC of P7, P8, and P9 is executing, the time for executing it increases by 20% over the initial plan;

- **Identifying changes to a future activity (only workload increases for future activities are used in this simulation, changes may also include the need to deal with new events such as the insertion of new requirements)**:
  - RA of P1, P2, and P3 causes DEC of the same project to increase by 20%;
  - RA of P4, P5, and P6 causes COD of the same project to increase by 20%;
  - RA of P7, P8, and P9 causes TST of the same project to increase by 20%;

The trigger times of these events are the completion dates of RA. If the corresponding changed activities have allocated resources at the trigger time, a new rescheduling needs to be carried out.

The trigger times for these events are the planned end times of the activities.

4.2 Simulation Experiment

GA-based scheduling uses a population of chromosomes to define and solve the scheduling problem. A chromosome evolution process that includes fitness computation for each chromosome, selection, crossover, and mutation can typically be expected to provide near optimal scheduling results. For TWINS, we set the chromosome population to 100, the crossover rate to 0.8, the mutation rate to 0.02, and generation number to 500.

TWINS uses an appropriate incremental scheduling time window to reduce uncertainties in scheduling results. We used the following simulation scenarios to evaluate the effects of using TWINS:

- Change the time window size and analyze the numbers of reschedulings caused by different kinds of events. Use these results to demonstrate the stability, accuracy, and uncertainty of scheduling results generated from different time window sizes.
- Change the number of events and compare the rescheduling number results obtained from running simulations by using different window sizes. Use these results to demonstrate the circumstances when TWINS is relatively more effective.

4.2.1 Scheduling Cost Variation with Changing Window Size

One key issue in using TWINS is how to determine the size of the window to be used in a rescheduling. The window size affects the number of different kinds of reschedulings that will be necessitated, as well as the costs of the reschedulings. Thus one major goal of our experimentation is to provide gain insight into the question of what window size should be selected in order to provide a good compromise between the accuracy and scheduling stability that is obtained from more frequent reschedulings using smaller windows vs. the reduced costs of less frequent reschedulings over larger window sizes that nevertheless lead to greater uncertainty.

Fig 4 shows the influence of different window sizes on the number of reschedulings and the total time of a simulation run. Note that when the size of the time window increases, more events are included in the window, and the following results are obtained:

- The number of "*arrival of new requirement*" events is always 3 because all projects arrive at times when those three events are scheduled to occur.
- The number of "*delay of currently executing activity*" reschedulings seems to increase as the time window sizes increases. If the time window size increases, more activities will be included in the window and will affect each other, thus increasing the probability that "delay of current executing activity" events will affect the scheduled result of other activities. That seems to explain why reschedulings caused by this event tend to increase.
- The number of "*identifying changes to a future activity*" reschedulings also seems to increase with increases in window size. If time window size increases, the currently

executing activities and the future activities that are changed by these events are more likely to be in the same window. Thus reschedulings caused by "identify the changes of a future activity" increase as time window size increases.

- The number of "*un-assigned allocation of successive activities*" reschedulings seems to decrease with increases in time window size. If the time window size increases, before the start of execution of activities for which resources are not yet allocated, there is a higher probability that new reschedulings caused by other kinds of events may occur causing these future activities to lose their allocated resources.

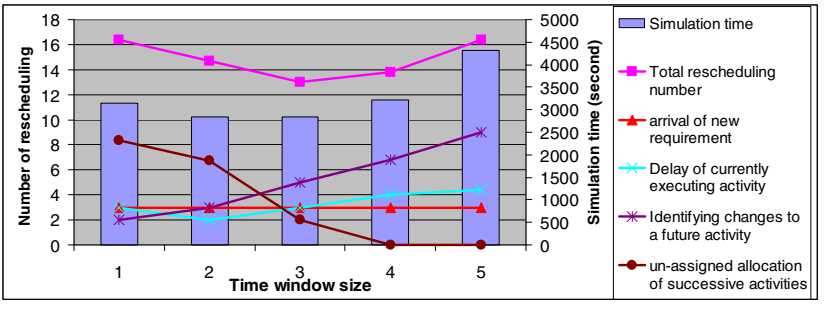

Fig. 4. Number of different kinds of reschedulings and the total rescheduling time (results are averages of 20 different simulations)

The above analysis shows that a small time window causes a lower number of reactive reschedulings, but a larger number of proactive reschedulings. An appropriately chosen time window size (3 in this example) causes a minimum number of reschedulings. Our research is still quite preliminary, but it suggests that this optimal window size may be context dependent and that more research is needed to understand better what features and state information should be used (and how) to help predict optimal window size.

4.2.2 Scheduling Cost Variation with Different Number of Disruptions

Another of the advantages that we believe are derived from the TWINS approach is that it can be particularly useful in increasing the stability of environments that must deal with a lot of disruptions.

To evaluate this hypothesis, we have compared the results obtained from running simulations by using different window sizes and total number of "*delay of currently executing activity*", "*identifying changes to a future activity*", and "*un-assigned allocation of successive activities*" disruptions. We set window size as 2 and 5 respectively. We then compared the results obtained when have to deal with all these disruptions; disruptions to only 6 projects: P1, P2, P4, P5, P7, and P8; having to deal with disruptions to only 3 projects: P1, P4, and P7; and having to deal with no disruptions at all.

Fig 5 shows that if there are a large number of disruptions, a small window size can reduce the number of reschedulings needed. But as the number of disruptive events decreases, the system becomes more stable and larger window sizes can be used, which reduces the number of needed reschedulings. This case study shows that

relatively small time windows are likely to be most effective when there are a large number of disruptions in systems, while large time windows are effective when a system is less prone to disruptions.

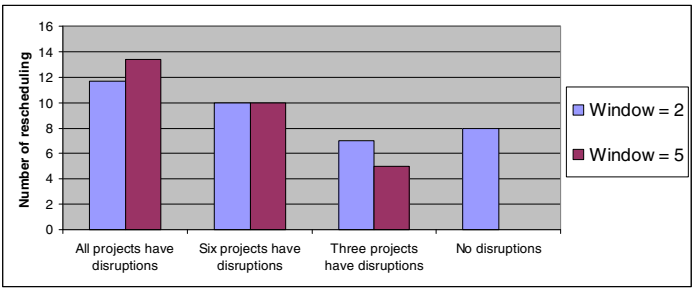

Fig. 5. Comparisons of the effects of different number of disruptions (results are averages of 20 different simulations)

5 Summary and Future Work

This paper has presented a Time Window based INcremental resource Scheduling method (TWINS) that uses a genetic algorithm. We used this method to develop a scheduling tool that was integrated with an existing discrete event simulation system in order to study the effectiveness of the approach in creating good resource allocation schedules in affordable time. We used this system to support a variety of simulations of software development processes. These initial case studies suggest that this approach can be effective. In order to gain more confidence in the approach further research is needed, however.

Future work

Determining optimal window size: As noted in the previous section research is needed in order to determine what window size should be used in rescheduling. In addition, we will also explore how to dynamically change time window size based on consideration of the appropriate runtime state and parameter information.

Analysis of different processes and domains: This paper mainly focuses on changing window size when attempting to address how to deal with disruptions that are caused by four different kinds of events in the software development process domain. Different processes from different domains should be studied as well.

More kinds of events: There are additional kinds of disruptive events in software development that could also be studied. For example, the execution of one activity may introduce the need for adding more detailed step specifications to another activity or may even create the need for an entirely new activity. Future work should address the issues of how to deal with these disruptions.

Evaluation in real software organizations: This paper only uses simulation to evaluate TWINS. Real applications and experiences with the method in actual software development are needed to further evaluate the TWINS approach.

Complex factors related to human resources: The scheduling of human resources in software development should ideally take into account human motivations, learning ability, productivity changes, and the effects of collaboration. Future work should take these complex factors into consideration to improve our models and methods.

Acknowledgments

We would like to thank Bin Chen and Heather Conboy for their help with the transformation from Little-JIL to RUFG, and Prof. Lori A. Clarke, Dr. M. S. Raunak, and Sandy Wise for their valuable feedback about this work. This paper is supported by the National Natural Science Foundation of China under grant Nos. 90718042, 60903051, the Hi-Tech Research and Development Program (863 Program) of China under grant No. 2007AA010303, as well as the National Basic Research Program (973 program) under grant No. 2007CB310802. This work was also supported by the National Science Foundation under Awards No. CCR-0205575, CCR-0427071, CCF-0820198, and IIS-0705772. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Alba, E., Chicano, J.F.: Software Project Management with GAs. Journal of Information Sciences 177, 2380–2401 (2007)
[2] Xiao, J., Wang, Q., Li, M., Yang, Q., Xie, L., Liu, D.: Value-based Multiple Software Projects Scheduling with Genetic Algorithm. In: Wang, Q., Garousi, V., Madachy, R., Pfahl, D. (eds.) ICSP 2009. LNCS, vol. 5543, pp. 50–62. Springer, Heidelberg (2009)
[3] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher, P.: Value-Based Software Engineering. Springer, Heidelberg (2005)
[4] Barreto, A., Barros, M.d.O., Werner, C.M.L.: Staffing a software project: A constraint satisfaction and optimization-based approach. Computer & Operations Research 35, 3073–3089 (2008)
[5] Pham, D.-N., Klinkert, A.: Surgical case scheduling as a generalized job shop scheduling problem. European Journal of Operational Research 185, 1011–1025 (2008)
[6] Goncalves, J.F., Mendes, J.J.M., Resende, M.G.C.: A Genetic Algorithm for the Resource Constrained Multi-project Scheduling Problem. European Journal of Operational Research 189, 1171–1190 (2008)
[7] Peteghem, V.V., Vanhoucke, M.: A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research (2009)
[8] Fowler, J.W., Monch, L., Rose, O.: Scheduling and Simulation. In: Herrmann, J.W. (ed.) Handbook of Production Scheduling, pp. 109–133. Springer, US (2006)
[9] Pfeiffer, A.s., Kadar, B., Monostori, L.s.: Stability-oriented evaluation of rescheduling strategies, by using simulation. Computers in Industry 58, 630–643 (2007)

[10] Herrmann, J.W.: Rescheduling Startegies, Policies, and Methods. In: Handbook of Production Scheduling, pp. 135–148
[11] Herroelen, W., Leus, R.: Project Scheduling under Uncertainty: Survey and Research Potentials. European Journal of Operational Research 165, 289–306 (2005)
[12] Antoniol, G., Penta, M.D., Harman, M.: A Robust Search–Based Approach to Project Management in the Presence of Abandonment, Rework, Error and Uncertainty. In: Proceedings of the 10th International Symposium on Software Metrics, pp. 172–183 (2004)
[13] Li, Z., Ierapetritou, M.G.: Robust Optimization for Process Scheduling Under Uncertainty. Industrial and Engineering Chemistry Research 47, 4148–4157 (2008)
[14] Al-Fawzan, M.A., Haouari, M.: A bi-objective model for robust resource-constrained project scheduling. International Journal of Production Economics 96, 175–187 (2005)
[15] Wang, J.: A fuzzy robust scheduling approach for product development projects. European Journal of Operational Research 152, 180–194 (2004)
[16] Ghezail, F., Pierreval, H., Hajri-Gabouj, S.: Analysis of robustness in proactive scheduling: A graphical approach. Computers & Industrial Engineering (2009)
[17] Rangsaritratsamee, R., Ferrell Jr., W.G., Kurz, M.B.: Dynamic rescheduling that simultaneously considers efficiency and stability. Computers & Industrial Engineering 46, 1–15 (2004)
[18] Yang, B.: Single Machine Rescheduling with New Jobs Arrivals and Processing Time Compression. International Journal of Advanced Manufacturing Technology 34, 378–384 (2007)
[19] Xiao, J., Osterweil, L.J., Wang, Q., Li, M.: Dynamic Resource Scheduling in Disruption-Prone Software Development Environments. Department of Computer Science, University of Massachusetts, Amherst, MA 01003 UM-CS-2009-050 (2009)
[20] Holland, J.H.: Adaptation in natural and artificial systems. MIT Press, Cambridge (1992)
[21] Cass, A.G., Lerner, B.S., McCall, E.K., Osterweil, L.J., Stanley, J., Sutton, M., Wise, A.: Little-JIL/Juliette: A Process Definition Language and Interpreter. In: Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, pp. 754–757 (2000)
[22] Wise, A.: Little-JIL 1.5 Language Report, Department of Computer Science, University of Massachusetts, Amherst UM-CS-2006-51 (2006)
[23] Xiao, J., Osterweil, L.J., Wang, Q., Li, M.: Dynamic Scheduling in Systems with Complex Resource Allocation Requirements. Department of Computer Science at the University of Massachusetts Amherst. Technical report: UM-CS-2009-049 (2009)