Activity-Based Scheduling of IT Changes♦

David Trastour, Maher Rahmouni, Claudio Bartolini
Trusted Systems Laboratory
HP Laboratories Bristol
HPL-2007-103
July 3, 2007*

ITIL, change management, scheduling

Change management is a disciplined process for introducing required changes onto the IT environment, with the underlying objective of minimizing disruptions to the business services as a result of performing IT changes. Currently, one of the most pressing problems in change management is the scheduling and planning of changes. Building on an earlier mathematical formulation of the change scheduling problem, in this paper we take the formulation of the problem one step further by breaking down the changes into the activities that compose them. We illustrate the theoretical viability of the approach, discuss the limit of its applicability to real life scenarios, describe heuristic techniques that promise to bridge the scalability gap and provide experimental validation for them.

* Internal Accession Date Only
♦ AIMS 2007, LNCS 4543, pp. 73-84, 2007

Approved for External Publication

Activity-Based Scheduling of IT Changes

David Trastour[1], Maher Rahmouni[1], and Claudio Bartolini[2]

[1] HP Labs Bristol, UK
[2] HP Labs Palo Alto, USA
{david.trastour,maher.rahmouni,claudio.bartolini}@hp.com

Abstract. Change management is a disciplined process for introducing required changes onto the IT environment, with the underlying objective of minimizing disruptions to the business services as a result of performing IT changes. Currently, one of the most pressing problems in change management is the scheduling and planning of changes. Building on an earlier mathematical formulation of the change scheduling problem, in this paper we take the formulation of the problem one step further by breaking down the changes into the activities that compose them. We illustrate the theoretical viability of the approach, discuss the limit of its applicability to real life scenarios, describe heuristic techniques that promise to bridge the scalability gap and provide experimental validation for them.

1 Introduction

As defined in the IT infrastructure library (ITIL, [1]), *change management* is a disciplined process for introducing required changes onto the IT environment. A good and effective change management process must minimize disruptions to the business services as a result of performing IT changes.

The main driver for IT organisations to adopt ITIL is the need to improve service quality [2]. Change management has a direct impact on service quality as it tries to understand and reduce risks. This makes change management a major ITIL process, often implemented early on when adopting ITIL, alongside incident management and configuration management.

Our research agenda in change management is driven by the results of a survey with IT change managers and practitioners in 2006 [3]. The survey highlighted that currently, the top three problems in change management are: 1) scheduling and planning of changes, 2) handling high number of urgent changes, and 3) dealing with ill-definition of request for changes. To respond to these challenges, we have projects underway on assessment of risk in change management, on assisted design of changes and on business-driven scheduling of changes. In this work, we formalize the change scheduling as an optimization problem and we develop methods to solve it to optimality. We build on our previous work by extending our conceptual model for change scheduling and breaking down the changes into the activities that compose them. As an example, we reuse the calculation of business impact defined in [5] and use it as the objective function of the optimization problem.

A.K. Bandara and M. Burgess (Eds.): AIMS 2007, LNCS 4543, pp. 73–84, 2007.

The problem with scheduling changes is that IT practitioners have little visibility into business risk and impact of changes onto customers. In order to make as much information as possible transparently available to all the stakeholders, ITIL recommends the creation of a change advisory board (CAB). The typical CAB is made up of decision-makers from IT operations, application teams, and business units—usually dozens of people—who meet weekly to review change requests, evaluate risks, identify impacts, accept or reject changes, and prioritize and schedule the ones they approve. However, CAB meetings are usually long and tedious and consume a great amount of time that could be made available to deal with change building, testing and deployment, with consequent benefit for the IT organization's efficiency. The problem is further complicated by the ever increasing number of changes and the constantly growing complexity of IT infrastructure. It is not uncommon for CABs to receive several hundreds of changes per week (such volume of change has been observed in HP outsourcing customers).

Besides the negative impact on efficiency imposed by CAB meetings, various other factors impact the effectiveness of the change management process, the effect of which could be mitigated by careful scheduling:

- because of the complexity of infrastructures and the number of possible stakeholders, CABs can't accurately identify "change collisions" that occur when two simultaneous changes impact the same resource or application;
- it is difficult to understand cross-organization schedules since large organizations have multiple CABs with no coordination between them.

In this paper, we discuss how our approach to activity-based scheduling of IT changes allows us to tackle these problems. The remainder of this paper is structured as follows. In section 2 we introduce concepts and design relevant data structures that are the bases for the formalization of the activity-based change scheduling problem (presented in section 3). In section 4 we provide experimental validation of the approach. We discuss related work in section 5 and draw our conclusions in section 6.

2 Related Work

Our work belongs to the research domain in IT service management, and in particular of business-driven IT management (BDIM). For a comprehensive review of business-IT management, see [9]. The research in Business-driven IT management covers automation and decision support for IT management processes, driven by the objectives of the business.

The novelty of the work presented here, (as well as for [5] that preceded it), is that our approach targets the dimensions of *people* and *processes* in IT management rather than the *technology* dimension of it as the most notable early efforts in business-driven IT management do, in particular the ones that were applied to (see [9,10,11,12,13,18] for service level management, [12,14,15] for capacity management, and [19] for security management on the service delivery side of ITIL [1]).

More relevant to our line of research are BDIM works that touch on IT support processes, such as incident management, problem management, change management itself and configuration management. The management by business objectives (MBO) methodology that we described in [16] - and that we applied there to incident management – is also the driver for this work. However, the focus of this paper is on the solution of the scheduling problem itself, whereas in our previous paper we did lead to the formulation of (mixed integer programming) incident prioritization problem, but we touched on it just as an example of putting the MBO methodology to work. Besides, the scheduling problem considered here reaches a far deeper level of complexity than the incident prioritization problem.

Coming to change management, Keller's CHAMPS [17] (CHAnge Management with Planning and Scheduling) is the seminal work. At a first level of analysis, the formulation of the scheduling problem that we present here can look very similar to the scheduling optimization problem that CHAMPS solves. While this provide mutual validation of both approaches, it has to be noted that CHAMPS addresses the automation aspects of the change management process and deals in particular with software deployment, whereas in this work we look at scheduling as a decision problem, offering support for negotiation of the forward schedule of change in CAB (change advisory board) meetings. In particular, CHAMPS assigns activities to servers, whereas in our formulation activities are assigned to technicians and affect configuration items. Another significant difference in the two approaches is that this work takes into account the IT service model: hardware components, applications and services and their dependencies. This allows us to model and avoid conflicts between changes.

With respect to our previous work, in [4] we introduced a mathematical formulation of the business impact of performing IT changes. In [5], we presented a conceptual model of change scheduling and evaluated the business impact of a change schedule. While the algorithm presented in [5] was only dealing with assigning changes to change windows, here we take the scheduling problem to the next level of detail, by actually scheduling down to the level of the single change activities composing the change, and producing detailed schedules for maintenance windows. [5] also concentrated on providing a plausible business-oriented utility function to maximize, whereas here we are here agnostic as far as the objective function is concerned.

Finally, scheduling is a field which has received a lot of attention. A great variety of scheduling problems [20] have been studied and many solution methods have been used. Staff scheduling problems in particular have been well studied [Ernst]. Our problem can be seen as a generation of a generalized resource constraint scheduling problem [21]. Our problem has the additional difficulty that one need to avoid conflicting change activities on IT components.

3 Change Scheduling

As seen in the introduction, CAB members need to have up-to-date change information to be able to make good decisions. Such information includes the detailed designs of changes, the topology of the underlying IT infrastructure and services, the

calendars of change implementers. We now briefly recall the sections of the conceptual model presented in [5] that are relevant to our more detailed problem description. We extend the model to include the notion of change activities. We then move on to presenting the mathematical formalization of the activity-based scheduling problem.

We first need a model of the IT services that are under change control. ITIL calls configuration item any component of the IT infrastructure (hardware or software) that is required to deliver a service. The configuration management database (CMDB) holds the collection of configuration items, along with their dependencies. We model the CMDB as a directed graph where the nodes are configuration items and where edges represent direct dependencies between configuration items. Such dependencies can be containment dependencies (i.e. a web server instance runs a given server) or logical dependencies (i.e. a J2EE application depends on a database server).

A *request for change (RFC)* represents a formal proposal for a change to be made. The RFC contains a high-level textual description of the change. It also specifies an implementation deadline, by which the change must be implemented. Penalties may apply if not.

During the planning phase of the change management process, the high-level description of the change contained in the RFC is refined into a concrete *implementation plan*. The implementation plan describes the collection of *activities* and *resources* (people, technology, processes) that are required to implement the change. The plan also specifies dependency constraints between activities. As commonly done in project management [6], the dependency constraints are expressed in the form of a lag time and a dependency type, *finish-before-start*, *start-before-finish*, *finish-before-finish* or *start-before-start* constraints.

A change activity represents an elementary action that must be performed in order to complete a step of the change implementation. An activity has an associated expected duration and requires a set of implementation resources. As seen previously, it might also depend on other activities. Finally, a change activity may affect one or more configuration items.

An implementation resource is any technical resource that is required to perform a change activity, such as a change implementer or a software agent. Our model attaches an hourly cost to each implementation resource.

Finally, change windows are pre-agreed periods of time during which maintenance can be performed for an IT service. Such windows are usually found in service level agreements (SLA) or operating level agreements (OLA).

With this conceptual model in mind, we can define the activity-based scheduling problem. Our solution to the problem consists of two phases. In the first phase, changes are assigned to pre-defined change windows. This is modeled in figure 1 with the *change window assignment* association. In the second phase, activities are assigned to implementation resources within each change windows, and this results in an *assignments* being created.

If we look at the activity-based scheduling problem as an optimization problem, several objective functions can be considered: minimizing the total cost of implementing changes, maximizing the number of changes to implement or minimizing the downtime of certain applications. We thoroughly discussed alternative objective functions definition in [5]

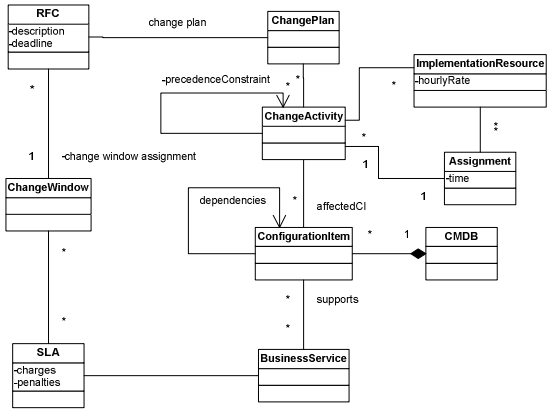

Fig. 1. Change scheduling conceptual model

and will not go into nearly as much detail in this paper. However, the objective function does play a role in the mathematical formulation of the problem, and we will cover it from this point of view in the following section

4 Mathematical Formulation of the Activity-Based Change Scheduling Problem

Let $C = \{c_i : 1 \le i \le N\}$ be the set of changes that have been designed, built and tested and are ready to be scheduled. Each change c_i is composed of a set of activities $A_i = \{a_{i,j} : 1 \le j \le |A_i|\}$, where each activity $a_{i,j}$ has an estimated duration $\delta_{i,j}$.

The scheduling of changes is done over a given time horizon. Let W be the number of predefined change windows $w : 1 \le w \le W$ that are pre-allocated within this time horizon. We refer to time within each change window through the index $t : 0 \le t < \Delta_w$.

Let $\{r_k : 1 \le k \le R\}$ be the set of implementation resources that are necessary to implement changes. Let $\kappa_{k,w,t}$ be the capacity of resource r_k at time t in window w. This capacity allows us to model both the availability of a resource (when $\kappa_{k,w,t} = 0$ the resource is unavailable) and the degree of parallelism of a given resource (a resource can perform up to $\kappa_{k,w,t}$ activities in parallel). Let $\rho_{i,j}$ be the set of resources that are necessary to implement activity $a_{i,j}$.

To represent conflicts between changes, we also need a model of the service hierarchy and of the configuration items that are being changed. Let $\{i_l : 1 \le l \le I\}$ be

the set of configuration items. Let $\tilde{A}_l$ be the set of activities that directly impact configuration item i_l. Let D_l be the set of configuration items that depend on i_l (i.e. the transitive closure of its direct dependants).

Possible solutions to the scheduling problem are characterized by the binary variables $u_{i,w}$ and $x_{i,j,k,w,t}$. The variables have the following meaning: $u_{i,w}$ is equal to 1 if change c_i is scheduled in change window w, and is equal to 0 otherwise. $x_{i,j,k,w,t}$ is equal to 1 if the implementation of activity $a_{i,j}$ by the resource r_k **starts** in change window w at time t and is equal to 0 otherwise. Finally the variables $l_{l,w,t}$ will be used to represent resource locking in order to avoid conflict; more specifically $l_{l,w,t}$ is equal to 1 when the configuration item i_l is locked by a change activity at time t in change window w.

We now model the constraints of the problem. When omitted, the ranges for each index are as follows: $i : 1 \le i \le N$, $j : 1 \le j \le |A_i|$, $k : 1 \le k \le R$, $w : 1 \le w \le W$, $t : 0 \le i < \Delta_w$, and $l : 1 \le i \le I$.

$$\sum_{l=1}^{W} u_{i,k} \le 1 \quad \forall i \tag{1}$$

$$\sum_{j=1}^{|A_i|} \sum_{k=1}^{R} \sum_{t=1}^{T_w - 1} x_{i,j,k,w,t} = u_{i,k} \cdot \sum_{j=1}^{|A_i|} |\rho_{i,j}| \quad \forall i, \forall w \tag{2}$$

$$x_{i,j,k,w,t} = 0 \quad \begin{cases} \forall i, \forall j, \forall k, \forall w \\ \forall t : \Delta_w - \delta_{i,j} + 1 \le t < \Delta_w \end{cases} \tag{3}$$

Equation (1) ensures that each change is executed at most once. In equation (2) we make sure that if a change is scheduled to be executed in a change window, then all the activities that it comprises of are implemented within that change window. This also ensures that a change cannot span several change windows, which is undesirable as this situation would leave the infrastructure in an unknown state and would likely result in service unavailability.

Equation (3) ensures that any activity that is started in a change window is completed within the bounds of the change window.

$$\sum_{t=0}^{T_w - 1} x_{i,j,k,w,t} = u_{i,w} \quad \begin{cases} \forall i, \forall j, \forall w \\ \forall k : k \in \rho_{i,j} \end{cases} \tag{4}$$

$$x_{i,j,k,w,t} = 0 \quad \begin{cases} \forall i, \forall j, \forall w, \forall t \\ \forall k : k \notin \rho_{i,j} \end{cases} \tag{5}$$

Equations (4) and (5) guarantee that the appropriate resources are used in the implementation of each activity. In particular, (4) states that if the change is scheduled for a given window, then sometime during that window all the necessary resources are scheduled to start working on it. Conversely, (5) prevents this from happening if the for the resources that are not required.

As far as capacity constraints are concerned, their expression in terms of the $u_{i,w}$ and $x_{i,j,k,w,t}$ variables does not come naturally. However, we observe that they can naturally be expressed via a binary variable signaling when an activity is being executed (recall that $x_{i,j,k,w,t}$ only specifies when the activity starts). To this end, we introduce the auxiliary variable $z_{i,j,k,w,t}$, whose value is 1 at all time during activity execution and 0 otherwise. $z_{i,j,k,w,t}$ is in turn best calculated through the introduction of two more auxiliary variables: $s_{i,j,k,w,t}$, that is indefinitely equal to 1 after the activity started and 0 otherwise; and $f_{i,j,k,w,t}$, that is indefinitely equal to 1 after the activity finished and 0 otherwise.

$$s_{i,j,k,w,t} = \sum_{\tau=0}^{t} x_{i,j,k,w,\tau} \quad \forall i, \forall j, \forall k, \forall w, \forall t \tag{6}$$

$$f_{i,j,k,w,t} = \sum_{\tau=0}^{t-\delta_{i,j}} x_{i,j,k,w,\tau} \quad \forall i, \forall j, \forall k, \forall w, \forall t \tag{7}$$

$$z_{i,j,k,w,t} = s_{i,j,k,w,t} - f_{i,j,k,w,t} \quad \forall i, \forall j, \forall k, \forall w, \forall t \tag{8}$$

The interpretation of these auxiliary variables is best understood graphically, as shown in the table below.

Table 1. Illustration of problem variables for an activity of duration 5

t	0	1	2	3	4	5	6	7	8	9	10
$x_{i,j,k,w,t}$	0	0	0	1	0	0	0	0	0	0	0
$s_{i,j,k,w,t}$	0	0	0	1	1	1	1	1	1	1	1
$f_{i,j,k,w,t}$	0	0	0	0	0	0	0	0	1	1	1
$z_{i,j,k,w,t}$	0	0	0	1	1	1	1	1	0	0	0

Capacity constraints can now be quite naturally expressed:

$$\sum_{i=1}^{N} \sum_{j=1}^{|A_i|} z_{i,j,k,w,t} \leq \kappa_{k,w,t} \quad \forall k, \forall w, \forall t \tag{9}$$

The auxiliary variable f and s come very useful also when specifying precedence constraints between change activities. For example, we can naturally express a *finish-before-start* precedence constraint between two activities a_{i,j_1} and a_{i,j_2} with equation (10). λ_{i,j_1,j_2} represent an additional *lag* that can be modeled if needed.

$$\sum_{w=1}^{W}\sum_{t=0}^{T_w-1} f_{i,j_1,k,w,t} - s_{i,j_2,k,w,t} \le \lambda_{i,j_1,j_2} \quad \forall k, \forall w, \forall t \tag{10}$$

Start-before-finish, *start-before-start* and *finish-before-finish* constraints are expressed through similar linear compositions of $s_{i,j,k,w,t}$ and $f_{i,j,k,w,t}$.

The following constraints deal with the possibility of conflicting change activities on the infrastructure.

$$\sum_{a_{i,j}\in\tilde{A}_l} \frac{1}{|\rho_{i,j}|}\cdot\sum_{k=1}^{R} z_{i,j,k,w,t} = l_{l,w,t} \quad \forall l, \forall w, \forall t \tag{11}$$

$$l_{l,w,t} \le l_{l',w,t} \quad \forall l, l' : l' \in D_l, \forall w, \forall t \tag{12}$$

Equation (11) ensures that the lock $l_{l,w,t}$ is set and that, among the activities that have an effect on the configuration item i_l, only one activity is active at a time (possibly using several resources $\rho_{i,j}$). Equation (12) states that all dependent configuration items D_l are affected when the configuration item i_l is being worked on.

Other additional constraints can be imposed to make the model work in a practical setting. For example, one could require to have a minimum number of changes scheduled (i.e. 90% of changes must be scheduled). Or change managers and supervisors may want to restrict some changes to take place within certain given change windows, or to restrict the permissible schedules of some activities in other ways. The expression of these additional constraints lends itself quite usefully to the case in which only a marginal re-scheduling is necessary due to the incumbency of some changes. In this case, the user may want to prevent re-scheduling of changes whose implementation date is approaching. All these constraints can be naturally expressed through linear combinations of the $u_{i,w}$ and $x_{i,j,k,w,t}$.

In order for the problem formulation to be complete, we now express its *objective function*. Depending on the requirements of the change managers and supervisors, different instances of objective function could be used. As an example, when we minimize the total cost of implementing changes, including the estimated business impact [4], the objective function becomes:

$$\text{minimize} \sum_{w=1}^{W}\sum_{i=1}^{N} \phi_{i,w}.u_{i,w} \tag{13}$$

This completes the theoretical development of the activity-based change scheduling problem. In the next section we will discuss experimental validation of the method described here.

5 Experimental Validation

We have implemented the mathematical programming formulation presented in this paper using CPLEX [7]. Due to the complexity of the problem definition, it turns out that in the worst case scenario our formulation does not scale up to a number of changes in the order of the hundreds. This formulation has however been a valuable instrument to better understand user requirements, as it allowed us to quickly capture and test additional user constraints, and to compare alternative objective functions such as the minimization of the makespan or of the number of conflicts.

For practical applications of the algorithm, we therefore need to develop heuristic solutions, while we will still use the complete formulation to validate the accuracy of the heuristics for low-dimension cases. We therefore developed a priority-based list scheduler [8] where the business impact plays the role of the priority function.

To compare the performance of the two implementations and to gauge the quality of the solutions produced by the priority-based list scheduler, we have developed a random generator of changes and resources. The generator takes as input: the number of change requests submitted per day, the average number of activities per change, the number of managed services, the number of configuration items and the number of available implementers. The changes and resources generator produces the following:

- service model along with the dependencies between configuration items;
- service level agreement penalties;
- for each change, its type (emergency, routine and normal) and its implementation deadline. For example, the deadline of an emergency change is set to 2 to 4 days from its submission date on average;
- for each change, its reference random plan, modeled as a dependency graph between activities;
- for each activity, its duration, its required locks on configuration item and its required resources.

We have run series of experiments comparing both implementations with different loads of changes and resources. We have fixed the number of services to 20, the number of configuration items to 100, the average number of activities per change to 5 and we varied the number of changes and the number of resources as shown in Table 2.

Table 2. Experiments with varying load

	Activities per Change	Changes	CIs	Services	Resources
Example 1	5	30	100	20	5
Example 2	5	90	100	20	10
Example 3	5	300	100	20	38
Example 4	5	600	100	20	70

The results of our experiments are shown in Table 3. Both algorithms were run on an HP Workstation XW8200 with a 3.2 GHz Xeon processor and 2GB RAM. For each implementation, Table 3 shows the processing time needed to schedule the examples defined in Table 2 as well as the estimated overall business impact. The business impact of assigning a change to a change window is calculated by summing up the following three components:

1. Cost of implementing the change: each resource has an hourly rate
2. Potential revenue loss: estimated loss in revenue due to the degradation of services impacted by the change.
3. Penalties incurred from the violation of service level agreements including penalties for missing deadlines.

Table 3. Comparison between PLS and CPLEX implementations

	Priority list scheduler		CPLEX scheduler	
	Processing Time	Overall Impact	Processing Time	Overall Impact
Example 1	0.5 sec	$24 K	40 sec	$18K
Example 2	8 sec	$155K	4 hours	$70K
Example 3	97 sec	$376K	**	**
Example 4	531 sec	$948K	**	**

For low-dimension examples (less than a hundred changes), the CPLEX scheduler produces the optimal solution within an acceptable time. As the number of changes gets bigger, the processing time grows exponentially, making it impossible to apply it to real IT environments (thousands of changes per month). In examples 3 and 4, the scheduler ran over 12 hours without producing a result while the list scheduler took less than 10 minutes.

Through analyzing the results produced by both implementations for small examples, there are some improvements that could be made to the list scheduler for producing better results. One improvement is to try to fit more changes into each change window by scheduling activities with smaller *mobility* (distance between its earliest possible assignment and its latest possible assignment) first, while giving priority to the highest impacted changes. Another improvement would be to sort the changes not according to their impact over one change window but over two or more change windows. As an example, let's take two changes c1 and c2 and two change windows cw1 and cw2 and let's assume that the impact of assigning:

- c1 to cw1 is $10K
- c1 to cw2 is $15K
- c2 to cw1 is $8K
- c2 to cw2 is $24K

If we assign c1 to cw1 and c2 and cw2, the overall impact is $34K, but if we assign c2 to cw1 and c1 to cw2, the overall impact is $23K.

6 Conclusions

Building on an earlier mathematical formulation of the change scheduling problem, in this paper we presented a methodology and a tool which pushes the formalization of the problem to the next level of detail, by breaking down the changes into the activities that compose them. We illustrated the theoretical viability of the approach, discuss the limit of its applicability to real life scenarios, describe heuristic techniques that promise to bridge the scalability gap and provide experimental validation for them.

In conducting our experiments and showing the prototype to domain experts, it emerged that end users would found it difficult to deal with schedules that are automatically generated. The tool we have produced assumes that the knowledge regarding change activities is complete and accurate. This is not necessarily the case in a production environment and may lead to problematic schedules. Rather than having a fully automated procedure, domain experts expressed the need to incrementally schedule sets of changes and to preserve pre-existing assignments as much as possible. They also recommended that all constraints should not be treated with the same importance and that some constraints should be relaxed based on preferences and user feedback. Our immediate next steps are to address these issues.

Further along our research path we plan to take into account the fact that changes may fail during the course of their implementation, thereby possibly invalidating current schedules. We will do so by accommodating for back-out change plans in our schedule. The challenge ahead of us is that to indiscriminately account for each and every change failure in our models will most likely be overkill. Techniques assessing the likelihood of a change to fail given past history and present conditions look like a promising avenue to assess risk of failure and only scheduling for possible back-out if the change has a non-negligible likelihood of failing.

References

1. IT Infrastructure Library, ITIL Service Delivery and ITIL Service Support, Office of Government Commerce, UK (2003)
2. ITIL Change Management Maturity Benchmark Study, White Paper, Evergreen, http://www.evergreensys.com/whitepapers_tools/whitepapers/cmsurveyresults.pdf.
3. The Bottom Line Project. IT Change Management Challenges – Results of 2006 Web Survey, Technical Report DSC005-06, Computing Systems Department, Federal University of Campina Grande, Brazil (2006)
4. Sauvé, J., Rebouças, R., Moura, A., Bartolini, C., Boulmakoul, A., Trastour, D.: Business-driven support for change management: planning and scheduling of changes. In: State, R., van der Meer, S., O'Sullivan, D., Pfeifer, T. (eds.) DSOM 2006. LNCS, vol. 4269, pp. 23–25. Springer, Heidelberg (2006)
5. Rebouças, R., Sauvé, J., Moura, A., Bartolini, C., Boulmakoul, A., Trastour, D.: A decision support tool to optimize scheduling of IT changes. In: Proc. 10th IFIP/IEEE Symposium on Integrated Management (IM2007), Munich (May 2007)
6. Elmaghraby, E., Kamburowski, J.: The Analysis of Activity Networks under Generalized Precedence Relations (GPRs). Salah Management Science 38(9), 1245–1263 (1992)
7. ILOG Inc, ILOG CPLEX 10.1 user's manual and reference manual

8. Coffman, G.: Computer and Job-shop Scheduling Theory.Wiley and Sons Inc.(February 1976)
9. Sauvé, J.P., Moura, J.A.B., Sampaio, M.C., Jornada, J., Radziuk, E.: An Introductory Overview and Survey of Business–Driven IT Management. In: Proceedings of the 1st IEEE/IFIP International Workshop On Business-Driven IT Management (BDIM06), pp. 1–10
10. Liu, Z., Squillante, M.S., Wolf, J.L.: On maximizing service-level agreement profits. In: Proceedings of the ACM Conference on Electronic Commerce (2001)
11. Buco, M.J. et al.: Managing eBusiness on Demand SLA Contracts in Business Terms Using the Cross-SLA Execution Manager SAM, International Symposium on Autonomous Decentralized Systems (April 2002)
12. Sauvé, J., Marques, F., Moura, A., Sampaio, M., Jornada, J., Radziuk, E.: SLA Design from a Business Perspective. In: Schönwälder, J., Serrat, J. (eds.) DSOM 2005. LNCS, vol. 3775, Springer, Heidelberg (2005)
13. Aib, I., Sallé, M., Bartolini, C., Boulmakoul, A.: A Business Driven Management Framework for Utility Computing Environments, HP Labs Bristol Tech. Report 2004-171
14. Aiber, S., Gilat, D., Landau, A., Razinkov, N., Sela, A., Wasserkrug, S.: Autonomic Self–Optimization According to Business Objectives. In: Proceedings of the International Conference on Autonomic Computing (2004)
15. Menascé, D., Almeida, V.A.F., Fonseca, R., Mendes, M.A.: Business-Oriented Resource Management Policies for e-Commerce Servers, Performance Evaluation 42, Elsevier Science, 2000, pp. 223–239. Elsevier, North-Holland, Amsterdam (2000)
16. Bartolini, C., Sallé, M., Trastour, D.: IT Service Management driven by Business Objectives – An Application to Incident Management. In: Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2006) (April 2006)
17. Keller, A., Hellerstein, J., Wolf, J.L., Wu, K., Krishnan, V.: The CHAMPS System: Change Management with Planning and Scheduling. In: Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), April 2004, IEEE Press, New York (2004)
18. Bartolini, C., Sallé, M.: Business Driven Prioritization of Service Incidents. In: Sahai, A., Wu, F. (eds.) DSOM 2004. LNCS, vol. 3278, Springer, Heidelberg (2004)
19. Wei, H., Frinke, D., Carter, O., et al.: Cost–Benefit Analysis for Network Intrusion Detection Systems, In: Proceedings of the 28th Annual Computer Security Conference (October 2001)
20. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice Hall
21. Demeulemeester, E.L., Herroelen, W.S: A Branch-And-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem. Operations Research 45(2), 201–212 (1997)