Performance Management and Quantitative Modeling of IT Service Processes Using Mashup Patterns

Carlos Raniery P. dos Santos,
Lisandro Zambenedetti Granville
Institute of Informatics - UFRGS
Porto Alegre, RS, Brazil
Email: {crpsantos, granville}@inf.ufrgs.br

Winnie Cheng, David Loewenstern,
Larisa Shwartz, Nikos Anerousis
IBM Watson Research
Hawthorne, NY 10532, USA
Email: {wcheng, davidloe, lshwart, nikos}@us.ibm.com

***Abstract*—IT Service Management (ITSM) encompasses the practices for managing information technology systems. ITSM processes can be laden with segments where the human becomes a bottleneck and slows down the entire process. These inefficiencies are usually caused by insufficient design of the process itself, or defects in the tools being used. Our work provides a systematic framework for analyzing inefficiencies through a combined model to guide the use and estimate the value of improving orchestration of the process using mashup design patterns.**

I. Introduction

IT Service Management (ITSM) encompasses the practices for managing information technology systems. A significant body of work in this field addresses the issue of quality, *i.e.*, the frameworks, processes and metrics that measure effectiveness from the point of view of the receiver of such services. In this paper we study the service provider's perspective, particularly aspects of performance that have direct implications on the efficiency and cost of the operation from the provider's point of view.

In ITSM, a very large percentage of the work is performed by humans, rather than machines. Due to its unpredictable nature, human behavior and performance are much harder to model, and consequently, to optimize. Consider the example of a modern data network that receives packets at an entry point and needs to transfer them to a destination. The data packet in its path will be processed by a variety of system elements, each programmed to perform a specific task with a high amount of accuracy and predictability. The number of events (exceptions) that can interrupt a normal processing path can be large, but are always finite, and in many cases can be accounted for in the design itself through redundancy and error handling programs. By contrast, consider a service management operation organized according to the Information Technology Infrastructure Library (ITIL) standards. The presence of humans in the critical path for performing work introduces significant variability in the final outcome. Even if the nature of work is exactly the same, a human operator may execute it in a different way each time: she may use a different process; or different tools; or a different sequence of steps; or be interrupted a number of times by external factors such as a telephone call or email. Enforcing and obtaining tight performance bounds in a human-staffed organization is far more difficult than in a process executed by a machine.

The competitive nature of IT service provider organizations calls for a continuous improvement process: IT operators need to find ways to increase performance in terms of effectiveness, productivity and quality. We focus on productivity and define it as units of work performed per unit of time; but even with rigorous definitions, performance can be evaluated in many ways and at different levels of granularity. In this paper, we attempt to provide a comprehensive model for evaluating and optimizing productivity in human-centered ITSM processes. In our analysis we do not address exogenous events, such as answering telephone calls or other interruptions. Rather, we focus on individual steps in the process that can be measured through instrumentation or observation, and can be improved through design and automation. In particular, we study the request fulfillment process, one of the operational service management processes defined by ITIL. The analysis covers the steps that a human follows to execute the process, and identifies the areas where productivity improvement is possible. As we will see in the next section, ITSM processes can be laden with segments of the process where the human becomes a bottleneck and slows down the entire process. These inefficiencies are usually caused by insufficient design of the process itself, or defects in the tools being used.

Our work provides a systematic framework for analyzing inefficiencies, and addressing them through a set of design patterns that ultimately provide a significantly improved orchestration of the process. Again, our framework does not address the rather unpredictable and chaotic nature of external events that affect human behavior. After all, we cannot control when humans decide to interrupt or slow down the process – but we can measure and control the environment where productivity is limited due to lack of tools or inefficient orchestration.

The paper is organized as follows: in Section II, we discuss the technical background related to this paper. Section III defines inefficiencies in the ITSM context and proposes a model for measuring them. Section IV introduces the concept of mashup patterns as an effective approach for eliminating inefficiencies in an ITSM process. Section V demonstrates the application of the proposed methodology on the request fulfillment process and presents results of our case study. The paper concludes in Section VI with a brief summary of our findings and an outline of future work.

II. Background

A. Mashups

Mashups are Web applications created through the composition of pre-existing Web resources (*e.g.*, interactive maps, Web services, traditional HTML pages, or even Flash presentations) [1]. A key difference between mashups and traditional composition technologies like BPEL [2] and WSCI [3] is that mashups have the explicit goal of enabling users with limited or no programming skills to create their own tailored Web applications; traditional technologies, in turn, usually demand from developers a significant knowledge of programming languages, communication methods, and service description. Because mashups can be quickly created, they present the additional benefit of being appropriate for composing *situational applications*, *i.e.*, applications that tackle very particular, short-lived problems and so would be otherwise expensive to be coded by specialized personnel.

Mashups are composed through the use of *basic operators*. These operators hide from mashup end users how the original Web resources are orchestrated, so that users are exposed to the resulting mashup without being aware of the internal details of the composition. The coupling of such mashup operators can be guided in several ways [4]. Using metadata available in the operator's definition, the mashup system engine can select *default bindings* between the operators that are being used during the composition step. Compatibility rules and quality criteria can be used to suggest the most appropriate operators for a given one. In [5], we have defined the architectural components employed in this paper, and presented different categories of mashup basic operators, listed below.

- **Visual** operators deal with the visual presentation of relevant information through, for example, tables, graphs, and maps;
- **Control** operators relate to basic programming logics including loops and conditions;
- **Transform** operators manipulate data employing, for example, sorting and filtering;
- **Adaptation** operators translate original data from Web resources into formats more easily handled inside mashups;
- **Input** operators allow end users to feed mashups with their particular information, for example, through text fields in Web forms or by uploading files;
- **Execute** operators trigger the asynchronous background execution of actions without the explicit request of the end user, which is typical in background monitoring systems and similar applications;
- **Reuse** operators allow users to extend the available mashups to build more sophisticated compositions.

By using mashup basic operators, a user is able to specify mashups for a variety of different purposes. Where several problems share similar structure, however, it is often convenient to consider the employment of *mashup patterns* [6], where mashups with similar logic can be instantiated even more quickly from the same common pattern. In addition, because patterns enable previously proven mashups to be reused in new scenarios, mashups patterns provide an additional level of stability. Considering these benefits, in Section IV a set of patterns will be defined to tackle the inefficiencies discussed in Section III. In Section IV, we also present a quantitative evaluation of the potential impact of employing such patterns in ITSM using a combined model created based on the furthercoming ones.

B. Quantitative Modeling for Performance Assessment

Despite its importance, by design ITIL only provides high-level generic guidelines to IT organizations, without proposing, for example, concrete models and methods for capturing metrics and evaluating the quality of IT processes. Such evaluation is important for the IT service providers to quantify, measure, and most importantly to predict the deployment impact of IT solutions. The scientific community has worked to propose models, methodologies, and metrics to fill this gap. We present two of such models bellow, and then, in the next section, we propose a combined model to account for inefficiencies throughout the IT process.

1) Keystroke-Level Model: The Keystroke-Level Model (KLM) was proposed to predict the time an expert user takes to perform a given task on a given computer system [7] [8]. It is based on the sequence of keystroke-level actions the user must perform to accomplish a task. This sequence is taken from a set of gestures, presented in Table I, where the total task execution time is the sum of the time for each of the gestures in the sequence. The model also provides the average time for each gesture as presented below.

TABLE I
KEYSTROKE-LEVEL MODEL

	Gesture	Time
K	Keying	0.2 sec
B	Holding/Releasing key	0.1 sec
P	Pointing	1.1 sec
H	Homing	0.4 sec
M	Mentally Preparing	1.35 sec

As an example of the use of KLM to predict interaction time, we can consider the following scenario: a file deletion by a human operator. In this simple case, we consider that the procedure is to drag the file icon to the trash can icon. For this, the action sequence can be represented as follows:

1) Initiate the deletion (decide to do the task) **M**
2) Point to file icon **P**
3) Press and hold mouse button **B**
4) Drag file icon to trash can icon **P**

5) Release mouse button **B**

$$T_{total} = 2P + 2B + M = 2*1.1 + 2*0.1 + 1.35 = 3.75 \text{ sec} \quad (1)$$

2) Complexity Model: Brown and Hellerstein [9] introduced a methodology for quantitative benchmarking of configuration complexity of initial system setup, which has been extended in subsequent works. Brown *et al.* [10] proposed a model of configuration activity based on three concepts: configuration goals, procedures and actions. This model, along with the earlier methodology, allows an analyst to measure the complexity of different systems through proposed metrics classified into: execution, parameter, and memory complexities. Further, Diao *et al.* [11] extend these techniques to propose new complexity metrics and measure business-level performance indicators (*e.g.*, labor cost, productivity, quality). Finally, Diao and Keller [12] extended the metrics proposed in [10] to quantify the complexity of overall IT processes.

The approach used by Diao *et al.* can be summarized in three steps: assessing the complexity and timing a baseline scenario, construction of the regression model and evaluation of the model quality, and finally employing the model to predict labor costs, such as time. The relationship between time and complexity metrics are investigated using the multiple linear regression technique, with equation presented below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \quad (2)$$

In this equation, the $x_i$represent the IT management complexity metrics, and the least squares approach is employed to discover the β_i value. Further explanation of this methodology is beyond the scope of this paper and can be found in [11].

A simpler version of this model is presented in [13]. In this version, the complexity metrics are grouped by subtask, and the coefficients of Equation 2 are assumed to be proportional to the time spent on each subtask and equally weighted among metrics associated with the same subtask. This simplifies the model to the point that it can be applied with nearly the same ease and generality as the KLM model of the previous section, but addressing a much larger array of metrics. This work also extended the model to cover more complex processes which include forks, merges, and joins.

III. Identifying Inefficiencies in Service Management Processes

Inefficiencies are portions of a service management process characterized by suboptimal execution of activities. In this paper, we concentrate on inefficiencies characterized as segments of the process where suboptimal human productivity reduces overall throughput for the process. Inefficiencies can appear at fundamentally different levels of analysis. For the purpose of analyzing inefficiencies within ITSM we focus on two levels: higher level inefficiencies due to the complexity of the activity itself, and lower level inefficiencies due to the mechanical execution involved in performing the activity. As an example of a potential complexity inefficiency, in a process with many decision points, the operator needs to spend time determining the correct choice. In ITSM the lower level takes into consideration the interaction of human operators with the available software tools. For example, a web application created with poor usability can impose a significant amount of wasted time for the operator due to added mouse-clicks and keystrokes required to retrieve, create or update information.

In order to discover common inefficiencies, we initially collected descriptions of the tasks performed by a group of operators involved in a common activity, as will be discussed in more detail in Section V. Based on such descriptions, we recreated their processes in a form of sequence of tasks (*i.e.*, workflows), which were later validated by the operators. We then drilled down on these workflows to subtasks representing individual actions performed by the operators when interacting with the systems. With this detailed process, we could analyze both levels of inefficiencies. The non-exhaustive list below represents several types of inefficiencies we have found during our investigations:

Basic Inefficiencies

- **Context-Switching**: the operator needs to switch to different application from the one he/she is currently working on;
- **Locating Data**: after reaching the place where the information is available, the operator needs to search for the specific data across the screen;
- **Entering Data**: the operator has to input data manually in the screen he/she is working on;

Information Management Inefficiencies

- **Copy/Paste**: manual copying of data from one system to another;
- **Consistency checks**: the operator needs to guarantee that information is consistent in different places;
- **Information Lookups**: navigate between multiple screens to assemble information;

Skill-dependent Inefficiencies

- **Retaining Information**: remembering information for a subsequent step;
- **Combining Information**: all the data is in one screen and the operator needs to extract their meaning;
- **Data transformation**: the data require some simple manual processing (*e.g.*, reformatting dates to a local format) when transferring from one screen to another.

Synchronization Inefficiencies

- **Contacting a Person**: the operator needs to talk to someone by e-mail, instant messenger or in person;
- **Becoming aware**: the operator needs to access a tool repetitively to be aware of new service requests;

We classify the inefficiencies into four categories: basic, information management, skill-dependent, and synchronization. The first refers to the most simple and low-level inefficiencies, occurring independently from the others. Information-management inefficiencies are formed by the combination of several basic inefficiencies. Skill-dependent inefficiencies relate to the reasoning capabilities or training of the human operator. Finally, synchronization inefficiencies are those incurring delays due to factors such as waiting for an external input.

The identified inefficiencies can be mitigated by the use of a new generation of human-centric software tools aimed at decreasing both the actions required to complete a task and also the complexity encountered in carrying out the tasks.

A. Combined Model

We combine the models of Section II-B to take advantage of their relative merits. The KLM model is more widely used, is well corroborated by experiment (see for example [14]) and provides a wealth of detail at the lower level of human-computer interactions, while the Complexity model addresses both levels of inefficiencies. Thus, the presented models are the current best starting points for creating a new model to better evaluate the performance of IT processes from a time productivity perspective. To avoid double-counting, we discard all the complexity metrics except the memory and decision metrics, which capture higher level potential inefficiencies not addressed by KLM.

An analyst constructs the combined model in the following stages:

1) The analyst works with a domain expert to determine the tasks and subtasks of the process.
2) The analyst works with a domain expert to determine the complexity metrics for the Complexity model.
3) The analyst determines the KLM model through observation of user interactions.
4) The analyst measures the time to perform each subtask.
5) The analyst derives the Complexity model coefficients from the time measurements using the method of [13].
6) Since β_0 of Equation 2 represents the expected time for all factors not explained by the complexity model, the time predicted by the KLM model can be subtracted from it.

The value of the combined model is in allowing us to predict the expected change in time due to modifications in the process. If we create a standardized set of modification templates, we have the potential to search through these templates to find an optimally modified process, along with the expected time savings. We have identified a particular set of modification templates that apply specifically to subtasks involving interactions with a user interface in processing information. These *mashup patterns* form the building blocks for quantitatively motivated process improvement in human-computer interactions within ITSM.

IV. Mashup Patterns for ITSM Inefficiencies

As previously discussed, IT processes can present inefficiencies at various levels of analysis. Some of these inefficiencies are related to the interaction of users with the available tools, while other issues involve the complexity of performing a particular activity or providing mechanisms to decrease the failure risk of a specific action. We argue that these inefficiencies can be tackled by the adoption of mashups and, more specifically, by using mashup patterns. The mashup patterns described below are context-independent and can be used to address different ITSM scenarios. For each proposed mashup pattern, we discuss a relevant ITSM problem and the associated solution that employs that pattern.

Alerter pattern

Problem – In ITSM, it is common to find scenarios where a user needs to be aware of events in the managed environment. The simplest method to support this involves periodically accessing the management system to manually look for new events. For example, in the service dispatching scenario, service tickets are created at no specific time, and a dispatcher responsible for assigning those tickets needs to constantly access the ticketing system to check for new requests. That can become a problem if the dispatcher does not access the system sufficiently often, or if the time spent in unnecessary repeated accesses degrades the dispatcher's productivity. It can be even worse when the amount of monitored information is very large, or when the dispatcher needs to promptly react to time-sensitive events.

Solution – Mashups are not restricted to constantly interacting with users to perform some action. A mashup alerter pattern periodically monitors a system of interest on behalf of the user and, based on previously established conditions, sends notifications only when events of interest take place. For example, alerts can take the form of visual elements on the user's console, e-mail messages, or SMS (text) messages. Another advantage of using an alerter mashup pattern relates to situations where multiple systems must be monitored at the same time, eventually overloading the human operator with too much information. In that case, correlated events from different systems can be summarized to decrease the number of notifications. Figure 1-A presents how an alerter pattern can be created through the combination of mashup basic operators.

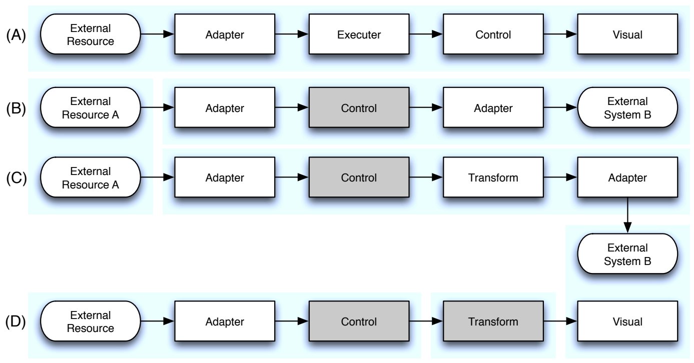


Fig. 1. Examples of mashup patterns. (A) Alerter. (B) Importer. (C) Transform. (D) Displayer.

Importer pattern

Problem – In ITSM, it is not uncommon to find scenarios where customers and service providers require the use of common data, although they use their own, particular database systems. To maintain data consistency across such systems, diverse methods can be used. For example, data adapters can grant one party access to the system of another's. When adapters are not available, screen scrapers can be used to access the Web interface of the remote system and retrieve the common data. Finally, users can access one another's system and manually copy and paste the common data into their own system's interface. In all these cases, maintaining data consistency is not transparent for the users because they need to consciously switch the integration method when accessing multiple systems.

Solution – If external resources natively expose an application programming interface (API), then leveraging their information is just a matter of basic software programming. However, it is often the case that the most valuable content is locked away in closed or proprietary formats. In these cases, an importer mashup pattern abstracts the different methods used to access the external data so that data consistency maintenance becomes transparent to the user. Figure 1-B presents how such an importer pattern can be implemented.

Transform pattern

Problem – While interacting with different systems, it is common to find cases where data needs to undergo some simple processing while being transferred from one screen to another. For example, while copying a field, a user needs to apply rules to filter out confidential information, or the data needs to be reformatted before it could be used by a different system (*e.g.*, US and UK date formats). These data transformations are usually manually performed because ITSM systems are often created without having integration in mind.

Solution – During the process of importing data, transform operators can be inserted into the mashup logic to enable the processing of certain types of data and thus both materializing the compatibility between systems and satisfying the requirements of the IT process. It is thus possible to reduce the number of manual interventions performed by the human through the automation of these adaptations. Figure 1-C highlights some elements to show that they are not required when transforming external data.

Displayer pattern

Problem – In order to make better decisions, humans involved in ITSM activities use information from multiple systems. This information is often memorized or recorded for future use during the decision making process. For example, the configuration database process can automatically generate a port number that needs to be remembered when installing another application. If the port number is forgotten or misremembered, errors in the process may occur.

Solution – By definition, mashups combine data from multiple sources and present the results of this combination in a Web page. However, this integration tends to occur only at the presentation level; it rarely occurs at the data level. This means that information from multiple systems can be presented alone in the same Web page as independent widgets. The employment of many displayer patterns in one page enforces the concept of a "single pane of glass". This concept reduces the risks of having a poorly executed process, which would generate errors and impose costs to the company. Figure 1-D presents the operators composing the displayer pattern.

A. *Quantitative Perspective*

The methodology presented in Section III-A allows us to estimate time savings for our mashup patterns. By doing this, we aim to help users to predict the performance improvements quantitatively before deploying mashups over their current ITSM processes. We will use the scenario described in the alerter pattern as an example.

The scenario described in the alerter pattern is a task composed of several subtasks. The first subtask is for the operator to notice that it is time to check for new events. We will label the time spent "becoming aware" of the need to start the task T_a. Once the operator decides to look for new requests, the next subtask is to interact with tools to examine the new events. We will label the time spent on this subtask T_k. This second subtask can be modeled by KLM, while the first subtask demands investigation beyond the scope of this paper. It is important to observe that this scenario does not include a subtask associated with memory complexity. This is because the human operator does not need to retain any information to look for new requests. We consider all requests to be processed independently of the others. The alerter pattern can decrease time spent on the task by reducing the awareness time T_a to zero. Once the requests arise, notifications are sent to the human operator automatically. In addition, a well-designed implementation of the alerter pattern could reduce T_k, for example by allowing the dispatcher to access the ticket associated with an alert with just one mouse click.

The scenarios where the importer and transformer patterns can be applied present both kinds of time inefficiencies we focus on this paper: mechanical execution (T_k) and task complexity (T_c). Since both operations can be completely automated by employing mashup patterns, the time reduction in those scenarios is 100%. The same applies to the displayer pattern. Since the necessary information to process a request is provided in one single screen to the human operator, the time spent looking for the information is decreased to zero and the time associated with complexity is significantly decreased due to eliminating the need to remember one specific piece of information. Table II summarizes the time reduction estimation for each of the presented patterns.

TABLE II
TIME SAVINGS

Pattern	Current	New
Alerter	$T_a + T_k$	T'_k
Importer	$N_{fields} \cdot (T_k + T_c)$	0 sec
Transformer	$T_k + T_c$	0 sec
Displayer	$T_k + T_c$	T'_c

The next section will provide a case study of the application of the quantitative methodology presented here to an existing ITSM activity.

V. Case Study

Amalgamation of multiple systems in a single pane of glass is critical in the area of IT Service Management, in particular in the area of IT Operations. Our case study considers the Request Fulfillment process, which is one of the operational processes in IT Management. Request Fulfillment is the process that deals with service requests, and it is defined in ITIL terminology as "management of customer or user requests that are not generated as an incident from an unexpected service delay or disruption." [15].

Request Fulfillment interfaces primarily with Service Desk and Incident Management, and supports two functions: it provides a point of communication for users and serves as a point of coordination between several groups and activities. In our study we focus on the latter function of Request Fulfillment. The process for this case study breaks coordination into two main activities: support the requests made by the customers, and solve those requests. Requests are solved by system administrators (SAs) with technical knowledge to resolve specific requests. Requests are supported by human operators, called dispatchers, with responsibilities that include: monitoring for new requests, dispatching the requests to the appropriate SA, and monitoring compliance with Service Level Agreements (SLAs).

A. *Dispatch Process for Service Management*

We study the case where the dispatcher has knowledge of standard fulfillment procedures and responsibility for generating requests and assigning them to a system administrator (SA). The Service Desk receives requests and creates a ticket, which may be any of the following types: incidents, problems and changes. Tickets are routed to a dispatcher, who is responsible for analyzing the request and determining the appropriate SA to assign it to for a resolution. The SA has the required skills and knowledge to solve specific requests and is responsible for taking the appropriate actions and closing the ticket.

Usually, each dispatcher is responsible for a team of system administrators in a specialized technical background. This setting has some advantages, providing high-quality and agile support, and allowing system administrators to work more efficiently.

Customers create new requests (*i.e.*, tickets) in Service Desk systems, and include all information used by system administrators to solve the request. Once the dispatcher receives the ticket and determines that his team can resolve the ticket, he would use his knowledge of his team's schedules and workloads as well as the expertise of each system administrator to finally make the assignment. Figure 2 presents the elements of this scenario.

Several time-consuming issues can arise in the dispatching process, making it infeasible to resolve tickets within the times established in SLAs and therefore resulting in financial loss to service providers. For example, it is common for customers and service providers to use their own ticketing systems, making it necessary to import data and maintain consistency between the systems. Dispatchers need to deal with data consistency and redundancy without violating any customer policies, such as data compliance for dealing with confidential information. In addition, the dispatcher and his team of SAs may be responsible for multiple customers, where each customer has a different ticketing system. This adds additional overhead in switching between multiple systems. Finally, information required for finding the most appropriate SA for one specific ticket could reside in various locations and require different tools to access it. For example, schedules tend to be managed by calendar-based systems, while the SA's actual workloads would be most accurately represented in Request Fulfillment systems and finally it is common to have SA skills associated with their user profile in the service provider's directory.

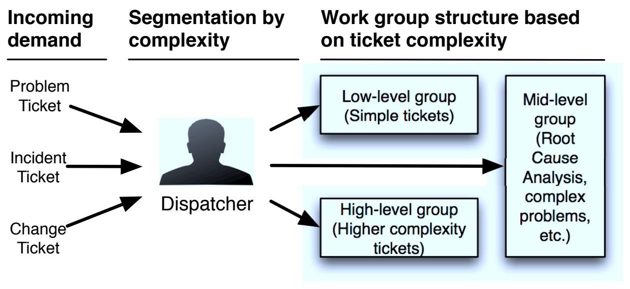

Incoming demand
Segmentation by complexity
Work group structure based on ticket complexity
Problem Ticket
Incident Ticket
Change Ticket
Dispatcher
Low-level group (Simple tickets)
High-level group (Higher complexity tickets)
Mid-level group (Root Cause Analysis, complex problems, etc.)

Fig. 2. E-Ticketing system to Service Desk

Automated dispatching solutions may be complex due to the variability of the environment and therefore it is not always feasible or the best alternative for this scenario. For example, in some situations a dispatcher may want to train a new administrator, and so may intentionally assign a request to a less skilled SA than is available. In this context, mashups are an interesting technology allowing the creation of dispatching systems to focus on the process of each dispatcher and helping him improve the efficiency of the assignment.

B. *Performance Footprint*

In order to discover bottlenecks in the dispatch process, we performed a series of time measurements among four dispatchers in a service delivery center. This information also allowed us to predict improvements obtained by the usage of mashups in the existing process. Using a stopwatch, we took 10 measurements for each assignment process and its individual tasks. This process is represented in Figure 3 as a workflow, which was obtained following the methodology in Section III-A.

The results of the measurements showed a significant time variation according to the ticket's complexity and the dispatcher's familiarity with the reported issue. For simple tickets (TS), usually repetitive tasks that the dispatcher is accustomed to assigning, the time average was 159 seconds (90% confidence level, 23.65 standard deviation), while for high complexity tickets (TC) this time was 357 seconds (90% confidence interval, 41.58 standard deviation). This difference can be justified by the need to spend more time reasoning about all the information related to the ticket, and also by the need to gather and provide detailed information to the system administrators.

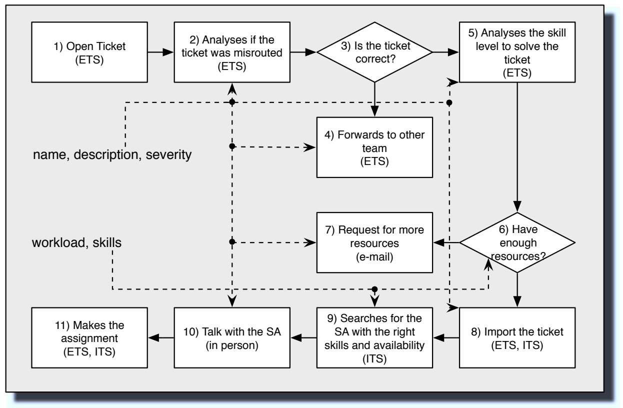

1) Open Ticket (ETS)
2) Analyses if the ticket was misrouted (ETS)
3) Is the ticket correct?
5) Analyses the skill level to solve the ticket (ETS)
name, description, severity
4) Forwards to other team (ETS)
workload, skills
7) Request for more resources (e-mail)
6) Have enough resources?
11) Makes the assignment (ETS, ITS)
10) Talk with the SA (in person)
9) Searches for the SA with the right skills and availability (ITS)
8) Import the ticket (ETS, ITS)

Fig. 3. Workflow of activities performed by dispatchers in Service Desk

Looking at the individual tasks, 35 (TS) and 58 (TC) seconds of the time was spend analyzing if the ticket was misrouted or not. To accomplish this task, dispatchers need to look for the right information (*e.g.*, keywords on tickets description) and decide if their teams have the right knowledge (*e.g.*, database, Unix) to solve the ticket. We also observed that most of the time was spend importing manually the tickets. This task consumed 41% and 50% of time respectively for simple and complex tickets. Finally, 58 (TS) and 94 (TC) seconds of the time were spent making the assignment, activity which involves updating the SA assignment information in both internal (ITS) and external ticketing systems (ETS).

C. Applying Mashup Patterns

All the proposed patterns can be used to create a mashup-based solution for the above-mentioned dispatching scenario. By using them, we aim to improve dispatchers assignment performance by automating some tasks, and by implementing the single pane of glass concept. This concept states that all the necessary information a human operator may need to achieve a goal should be presented in a single screen with the data already filtered and transformed.

Figure 4 shows how the proposed patterns relate with the dispatching tasks. Considering that name, description, and severity are the basic information from a ticket that a dispatcher uses most frequently to make the assignments, the *displayer pattern* can be used to show all the needed information in a single screen. This pattern can also be used to display the system administrator's workload and skills. With this pattern, it is possible to eliminate both *information lookup*, and *retaining information* inefficiencies of this task.

Since the dispatchers need to constantly monitor for new tickets, the *alerter pattern* can be used to notify them about new tickets as soon as they are created, and eliminate the *becoming aware* inefficiency. The *importer pattern* can be used to automate the task of importing tickets to the internal database, and the *transformer pattern* can be applied when the dispatchers need to modify (*e.g.,* augment, exclude) some information, for example, filtering confidential data (*e.g.*, phone numbers) on the ticket's description.

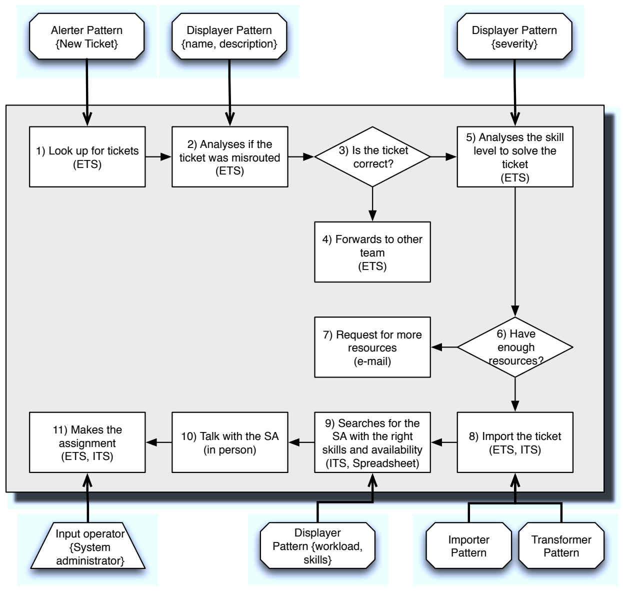
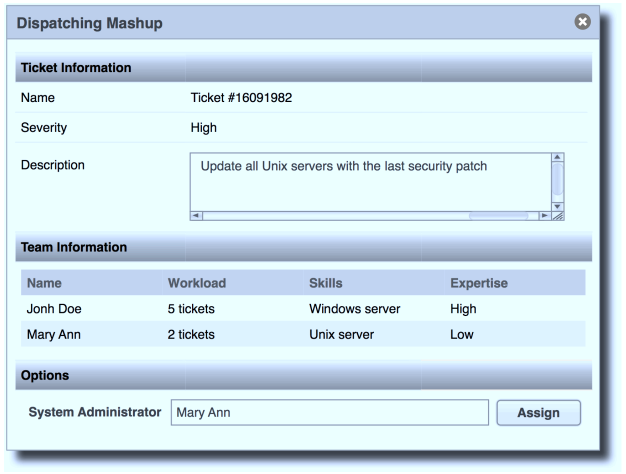

Alerter Pattern {New Ticket}
Displayer Pattern {name, description}
Displayer Pattern {severity}
1) Look up for tickets (ETS)
2) Analyses if the ticket was misrouted (ETS)
3) Is the ticket correct?
5) Analyses the skill level to solve the ticket (ETS)
4) Forwards to other team (ETS)
7) Request for more resources (e-mail)
6) Have enough resources?
11) Makes the assignment (ETS, ITS)
10) Talk with the SA (in person)
9) Searches for the SA with the right skills and availability (ITS, Spreadsheet)
8) Import the ticket (ETS, ITS)
Input operator {System administrator}
Displayer Pattern {workload, skills}
Importer Pattern
Transformer Pattern

Fig. 4. Relating mashup patterns with dispatching tasks

The final Graphical User Interface (GUI) for the dispatching mashup, constructed based on mashup patterns, is presented in Figure 5. It is important to observe that to create complete solutions, the patterns need to be composed of basic operators. As presented, the **input operator** was used to allow the dispatcher to specify the system administrator who will be responsible for solving the ticket.


Dispatching Mashup

Ticket Information

Name	Ticket #16091982
Severity	High
Description	Update all Unix servers with the last security patch

Team Information

Name	Workload	Skills	Expertise
Jonh Doe	5 tickets	Windows server	High
Mary Ann	2 tickets	Unix server	Low

Options

System Administrator: Mary Ann

Fig. 5. Graphical User Interface (GUI) of the mashup for the dispatching scenario

D. Predictions on Mashups Usage

Analyzing the tasks of the dispatching scenario and using the methodology described in Section 3.C, we estimated the times presented in Figure 6. The bars indicate the labor cost (*i.e.*, time) for each task of the dispatching process. The different colors indicate the obtained and estimated (with and without mashups) times for the process. As can be observed, the real measurements and the predicted times present a close fit, with a R^2 error of 0.82, which means the model can explain 82% of the variability in the time data, and a Root Mean Square Error (RMSE) of 13.34. This shows the accuracy of our model.

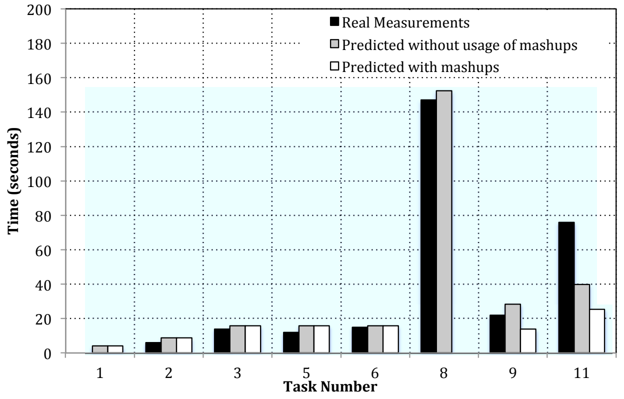


Fig. 6. Quantitative model validation with predictions of mashup cost

We do not present tasks 4 and 7 in Figure 6 because in our scenario, the tickets were not misrouted to the wrong dispatcher and we also considered that he/she always had enough resources (*i.e.*, System Administrators) to solve them. Task 10 is not showed because it was not observed during our evaluations. It is important to observe that this graph does not represent the awareness time since the KLM or the Complexity Model cannot predict it.

The obtained results show the significant time reduction of 64.42%, which mostly was due the automation of step 8 by using the importer pattern. The *displayer pattern* allows the reduction of the tasks 9 and 11, since the dispatcher does not need to look up any information in a different system, thus eliminating all the possible keystrokes, and also does not need to remember any information from a previous task, helping to decrease the memory complexity.

VI. Conclusions and Future Work

We believe that this paper represents a significant contribution: the use of a combined model to guide the use and estimate the value of improving IT processes using mashup design patterns. We have introduced a methodology for analyzing activities performed by human operators involved in IT Service Management. This methodology combines the Keystroke-Level and Complexity models to account for inefficiencies through the IT process. The combined model allowed us to tackle lower level inefficiencies of human-computer interactions, and higher level inefficiencies of performing IT tasks and subtasks, both from a quantitative perspective. As a solution for these inefficiencies, we have considered the employment of mashups and mashup patterns. The use of mashup patterns enabled us to make quantitative predictions of performance improvements due to the use of mashups. The improvements in productivity, usability and agility provided by the application of mashups demonstrate the viability of mashup technology as a means for improving IT Service Management.

Analyzis of the tasks performed by a group of human operators allowed us to find a set of common inefficiencies in their implementation of the ITIL Request Fulfillment process. Since these inefficiencies can be found in many different ITSM scenarios, mashup patterns can be applied as proven and reusable solutions. The analysis of one mashup developed from mashup patterns for a case study allowed us to observe a good fit between the times predicted by our combined model with real measurements. This indicates the feasibility of our methodology in predicting quantitatively labor costs savings due to the use of mashup technology. Finally, this paper demonstrates the significant improvement of 64.42% in the performance of the dispatching process in the case study.

As future research, we will expand our exploration of models of mental activities in business processes. By doing this, we hope to enhance the scope and accuracy of our model in predicting labor costs in the ITSM process. We also plan to extend the analysis of the ITSM process beyond inefficiencies, considering other aspects such as reliability. In this context, a plan to resolve failures in problematic tasks could be identified and instantiated for the final mashup solution. Finally, we will explore the automation of mashup development through the use of the combined model to guide selection of mashup patterns.

References

[1] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding mashup development," *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.

[2] OASIS, "Business process execution language, version 2.0," Organization for the Advancement of Structured Information Standards, May 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/

[3] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, "Web service choreography interface (WSCI) 1.0," World Wide Web Consortium, Note NOTE-wsci10-20020808, Aug. 2002.

[4] C. Cappiello, M. Matera, M. Picozzi, G. Sprega, D. Barbagallo, and C. Francalanci, "Dashmash: A mashup environment for end user development," in *ICWE*, ser. Lecture Notes in Computer Science, vol. 6757. Springer, 2011, pp. 152–166.

[5] C. R. P. dos Santos, R. S. Bezerra, J. a. M. Ceron, L. Z. Granville, and L. M. R. Tarouco, "On using mashups for composing network management applications," *Comm. Mag.*, vol. 48, pp. 112–122, December 2010.

[6] M. Ogrinz, *Mashup Patterns: Designs and Examples for the Modern Enterprise*, 1st ed. Addison-Wesley Professional, 2009.

[7] S. K. Card, A. Newell, and T. P. Moran, *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2000.

[8] D. Kieras, "Using the keystroke-level model to estimate execution times," *University of Michigan*, 2001.

[9] A. B. Brown and J. L. Hellerstein, "An approach to benchmarking configuration complexity," in *In Proceedings of the 11th ACM SIGOPS European Workshop*, 2004.

[10] A. B. Brown, A. Keller, and J. L. Hellerstein, "A model of configuration complexity and its application to a change management system," in *In Proceedings of the 9th IFIP/IEEE Symposium on Integrated Management*, 2005, pp. 631–644.

[11] Y. Diao, A. Keller, S. S. Parekh, and V. V. Marinov, “Predicting labor cost through it management complexity metrics,” in *In Proceedings of the 10th IFIP/IEEE Symposium on Integrated Management*, 2007, pp. 274–283.

[12] Y. Diao and A. Keller, “Quantifying the complexity of it service management processes,” in *DSOM*, 2006, pp. 61–73.

[13] L. Shwartz, Y. Diao, and G. Grabarnik, “Multi-tenant solution for it service management: A quantitative study of benefits,” in *Integrated Network Management*, 2009, pp. 721–731.

[14] W. D. Gray, B. E. John, and M. E. Atwood, “Project ernestine: Validating a goms analysis for predicting and explaining real-world task performance,” in *Human-Computer Interaction*, 1993.

[15] OGC, “Information technology infrastructure library v3 (itil v3),” Office of Government Commerce, May 2008. [Online]. Available: http://www.itil-officialsite.com/