Who Do You Call? Problem Resolution through Social Compute Units

Bikram Sengupta[1], Anshu Jain[1], Kamal Bhattacharya[1], Hong-Linh Truong[2], and Schahram Dustdar[2]

[1] IBM Research - India
{bsengupt,anshu.jain,kambhatt}@in.ibm.com
[2] Distributed Systems Group, Vienna University of Technology, Austria
{truong,dustdar}@infosys.tuwien.ac.at

Abstract. Service process orchestration using workflow technologies have led to significant improvements in generating predicable outcomes by automating tedious manual tasks but suffer from challenges related to the flexibility required in work especially when humans are involved. Recently emerging trends in enterprises to explore social computing concepts have realized value in more agile work process orchestrations but tend to be less predictable with respect to outcomes. In this paper we use IT services management, specifically, incident management for large scale systems, to investigate the interplay of workflow systems and social computing. We apply a recently introduced concept of Social Compute Units, and flexible teams sourced based on various parameters such as skills, availability, incident urgency, etc. in the context of resolution of incidents in an IT service provider organization. Results from simulation-based experiments indicate that the combination of SCUs and workflow based processes can lead to significant improvement in key service delivery outcomes, with average resolution time per incident and number of SLO violations being at times as low as 52.7% and 27.3% respectively of the corresponding values for pure workflow based incident management.

1 Introduction

Business process management (BPM) and workflow systems have had tremendous success in the past two decades with respect to both mindshare and deployment. We can safely consider service-oriented architecture (SOA) to be a business-as-usual design practice. On the other hand, we are observing enterprises embracing social computing as an alternative for executing more unstructured yet team-based collaborative, outcome-based strategies. Gartner [1] predicts that by 2015, we will observe a deeper penetration of social computing for the business as enterprises struggle to deal with the rigidity of business process techniques. Current workflows are suitable for automation of menial tasks but inflexible when it comes to supporting business users who must deal with complex decision making. However, a significant conceptual gap clearly exists between workflows on the one hand, and social computing as it is known today.

The goal of this paper is to introduce a framework that establishes the interactions of business processes and workflows with a concept called Social Compute Units (SCU) [8], recently introduced by some of the authors. A SCU is an abstraction of a team consisting of human resources that bring together the appropriate expertise to solve a given problem. The SCU abstraction treats the SCU as a programmable entity. The resources that make up a SCU are socially connected. The term *socially* implies connectedness of an individual beyond his or her organizational scope. The reason for connectedness could be prior ad-hoc collaborations but also collaboration within a given scope of responsibility where the scope is distributed across organizational verticals.

The context in which we propose the use of SCUs in this paper is the IT Service Management (ITSM) domain. More specifically, we are interested in the Incident Management process within ITSM. IT service vendors maintain large, complex IT infrastructure on behalf of their clients, and set up skill-based teams with the responsibility of maintaining different infrastructure components, such as applications, servers, databases and so on. When an interruption or degradation in service in some part of the IT infrastructure is detected, service requests are raised in the form of *tickets* that describe the incident. However, due to inherent dependencies between different system components, identifying the root cause of the problem is a complex, and often time-consuming, activity. The traditional approach to incident management is to have a human dispatcher intercept the ticket and review the incident description. Using his/her knowledge of the system and dependencies, the dispatcher then determines the most likely component that may be faulty and forwards the ticket to the relevant team, where it is assigned to a practitioner for investigation. The practitioner may determine the presence of a fault in the component and the incident may be resolved by taking corrective action to remove the fault. However, often the practitioner may discover that the fault does not lie in the component s/he is managing, and sends the ticket back to the dispatcher, who then needs to decide on the next team the ticket should be sent to, and this process continues till the right team receives the ticket and resolves the incident.

Such a process-driven approach may be reasonable when the problem description is detailed and clear. In reality, we find the end user reporting the incident to state the symptom at best. It is the human dispatcher's responsibility to guess the root-cause and identify the right person for the resolution job. This may be appropriate for simple and low severity incidents, but is risky in more complex situations. In those cases the overall resolution time may exceed the contractually agreed upon response time as manifested in Service Level Objectives (SLO). The consequences can be degradation of client satisfaction and/or monetary penalties. Our proposal is to demonstrate the benefits of bringing together appropriate resolution units, conceptualized as a SCUs, that possess the right skills composition to deal with the eventualities of the given context, as defined by the system where the incident occurs. The members of a SCU may be drawn from components where there is a higher likelihood of a fault, and component dependency information may be used to on/off-board members as investigation

proceeds. Such an agile way of managing incident resolution should help facilitate parallel investigations and thereby quicker resolution. However, SCUs may also incur a higher cost (since multiple practitioners are investigating a problem at once), hence its use has to be judiciously interleaved with the more standard workflow-driven sequential investigation of incidents, so that the right trade-off between quicker resolution and higher cost may be achieved.
The main contributions of the paper are as follows:

1. The development of a technical framework for SCU sourcing, invocation and evolution in the context of IT incident management spanning multiple teams and organizational verticals.
2. A simulation based approach to study the efficiencies that may be gained through SCUs over standard process management approaches, and the trade-offs involved.

The rest of the paper is organized as follows. In Section 2 we present a motivating example and introduce the concept of SCUs. In Section 3 we introduce the formal system model for the use of SCUs in incident management, and describe the life-cycle of a SCU from its invocation, evolution to dissolution. Section 4 presents a simulation based method to demonstrate the benefits that may be achieved through a combination of SCUs and workflow based approaches. After discussing related work in Section 5, we conclude with a discussion on future work and extensions of our framework in Section 6.

2 Motivating Example

Consider an IT service provider that manages applications on behalf of a client. Each application is a complex eco-system of its own, consisting of actual application code, the hosting middleware and operating system, servers, storage components, network and firewall configurations etc. An incident in any application may have its root-cause in any of its sub-systems. Resolving the incident for the application may henceforth require multiple skills, from programming skills to networking skills. IT service providers that manage hundreds or thousands of applications cannot scale by assigning individual teams to manage individual applications. Instead, it is more cost-efficient to form organizational units that are formed around skills and manage specific, similar system components.

Figure 1 shows an example of a component dependency graph for an application and its management context. The connectors between nodes indicate the nature of relationship or dependency between the components. For example, the straight line between *Application* and *Application Server* denotes a tight coupling as in a component being hosted on another (thus the dependency type is *isHostedOn*). Each dotted line, e.g. between *Web Server* and *Application Server* denotes a loose coupling between components as in one component being connected to the other through a web-service call or an HTTP request (*isConnectedTo* being the dependency type).

Each layer of an application is managed by an organizational entity as denoted on the left side. The *Application* component is managed by the Application Management team, which has the right set of coding skills and application knowledge to debug issues or extend functionality. The middleware layer (web server, application server and DBMS) is managed by the Application Hosting team that knows how to manage application servers and databases. In principle each management entity can be modeled as another node in the dependency graph but for simplicity we have depicted management entities as dotted line boxes around the components they are responsible for. For example, the dotted line box around the DBMS component could also be represented by a DBMS node connected to a Database Management node through a connector stereotyped as *isManagedBy*.

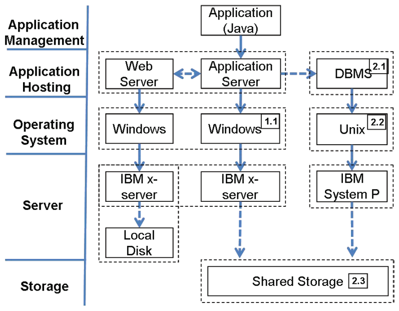


Fig. 1. Dependencies between application components and their management context

Let us now consider incident management - an important area in ITSM [3] - for the above example. An incident is an event which is not part of the standard operation of a service and causes, or may cause, an interruption to or a reduction in, the quality of that service. The goal of incident management is to restore the service functionality as quickly as required. The service restoration time is tied to the Service Level Objective (SLO) the client and provider have agreed upon and usually depends on the severity of the incident. An incident may be caused by a known problem for which there exists an action plan, or, may also require problem resolution as the root-cause is not known. Let us now examine incident resolution through two scenarios, with a focus on who is resolving an issue.

1. An incident states an issue with capacity exceeded on the server hosting the Application Server. The action plan (1.1 in Figure 1) is to delete all unnecessary temporary files using a pre-defined sequence of steps. The resource required is a member of the Operating System team and has Windows skills.
2. An incident indicates an issue with slow performance of the database management system (DBMS). The root cause needs to be identified. The IT Help Desk responsible for dispatching the ticket will route the ticket to the Application Hosting team and a resource will be assigned to look for standard causes, such as full log files (2.1 in Figure 1). The assigned resource determines that the cause is capacity exceeded on the server and hence passes

the incident on to the Server support to free up capacity (2.2 in Figure 1). The Server support team member executes some pre-defined steps to free up space, however notices that this is not solving the issue as the problem lies in the growth of the tables. This requires the ticket to be passed on to the storage team (2.3 in Figure 1). The storage team allocates space and takes necessary steps to resolve the issue and close the incident.

In the first scenario, the incident description has sufficient clarity for a dispatcher to quickly identify the relevant component and required skills. The ticket can be dispatched to the Windows Operating System team, where an available practitioner will execute a standard set of actions to resolve the issue. Thus such a ticket is amenable to a well-defined, repeatable incident management process. On the other hand, the second case involves a ticket where the symptom reported can be due to one of several possible causes. Using the standard sequential approach to incident management, we can try to proceed one component at a time, ruling out the presence of a fault in one component, before passing on the ticket to the next likely component. However, by the time the actual problem is discovered and fixed, significant time may have elapsed and a SLO may also have been breached. It may be noted that the time spent is not only due to the investigations that have to be carried out by each team, but also due to the delays that occur when a ticket has to be transferred across organizational boundaries, and the time that is wasted when a ticket is pending in a queue, waiting for a practitioner to become available and accept the ticket.

We posit that tickets such as in the second case above, need a different approach to incident management. Instead of being investigated by one practitioner from one component team at a time, they may need simultaneous attention from multiple practitioners and teams. In this paper we will apply the concept of a Social Compute Unit (SCU) [8] to address the second scenario above. A SCU is a "loosely coupled, virtual and nimble" team of socially-connected experts with skills in domains relevant to the problem at hand. Socially connected in common terms is widely understood outside of the work context. We define connectedness with respect to a *social fabric*. We believe that in general a human resource is a connected entity represented as a point (or node) in the social fabric. The social fabric consists of a continuum of network expressions such as an organizational network or a collaboration network. The former is typically well defined with specific roles and responsibilities assigned to nodes in the network. The latter is an expression of an individual's ability to transcend organizational boundaries to act as a source or sink of information.

A SCU team member may not be a dedicated resource, but someone who can invest a certain amount of time towards solving a problem, when the requirement comes up. A SCU is created on request (e.g. from the problem domain's business owner), has a certain amount of computing power derived from the skills and availability of its members and from their collaboration, uses its computing power towards addressing the problem which triggered its creation, and is dissolved upon problem resolution. These characteristics make the SCU an attractive approach for incident resolution management in an IT service organization. For more details on the SCU concept, the interested reader is referred to [8]. In the rest of the paper, we will describe in detail how SCUs may be used within an IT service management environment for facilitating collaborative incident resolution spanning multiple teams/organizations.

3 Incident Management Using SCUs: A Technical Framework

We first describe our system model, and then outline the basic principles that guide a SCU through its life-cycle. There may be different ways of realizing the abstract framework presented in this section and a concrete instantiation will be described and evaluated in the following section.

3.1 System Model

We assume a component dependency model represented as a graph $G = (V, E)$, where V is a set of nodes, each representing a component of the overall IT infrastructure being managed, and E is a set of directed edges representing dependency relationships between the nodes. Each edge may be represented by a triple $< C_i, C_j, DT_k >$, where $C_i, C_j \in V$ and represent system components, and $DT_k \in DT$ is the dependency relationship that exists between C_i (source) and C_j (target), drawn from a set DT of possible relationships relevant to the domain. A component $C_i \in V$ has a set of possible *features* C_i^F through which it provides services to the end-user. A *ticket* T_i is raised when an interruption or degradation in IT performance is detected, and may be initially represented as $< TS_i, D_i, P_i >$, where TS_i is the time-stamp when the ticket was raised, D_i is a textual description of the problem, P_i is a priority level indicating the criticality of the problem that may be drawn from an ordered sequence of numbers $\{1, 2, ...p\}$, with 1 representing the highest priority and p the lowest. Based on its priority P_i, a ticket will also have a Service Level Objective $SLO(P_i)$ which is the window of time W_i within which the incident needs to be resolved, thereby setting a deadline $TS_i + W_i$ that the ITSM team has to strive to meet. Each ticket is also implicitly associated with a fault profile $FP = \{(C_i, f_j) | C_i \in V, f \in C_i^F\}$, which is a set of features in a set of system components that have developed a fault, and need to be investigated and fixed. In the majority of cases, we may expect a ticket to be associated with a single malfunctioning feature in one component. Of course, the fault profile is not known when a ticket arrives, but needs to be discovered in course of the incident management process.

When a ticket is raised, it is the dispatcher's responsibility to review the problem description, and try to identify the likely component(s) that are not functioning correctly, so that the ticket may be dispatched to the relevant team(s). The dispatcher is aided in this task by the component dependency graph, and in general, s/he may be expected to devise a dispatch strategy that contains an ordered sequence of components $\{C_{i_1}, C_{i_2}, ..., C_{i_{|V|}}\}$, which we call the *Likelihood*

Sequence (LS(t)) for a ticket t, going from the most to the least likely component that may be the cause for this incident. $LS(t)$ represents the order in which the ticket should be dispatched to the different components for investigation, till the root cause is identified and fixed. Note that we assume a fully ordered sequence only for simplicity; the likelihood of some of the elements may be deemed to be equal, in which case any of them may be picked as the next component to be investigated. $LS(t)$ would depend on the ticket description, which may contain some indicators of the potential source of the problem, while the component dependency graph would make certain related components candidates for investigation as well, based on the nature of the relationship they have with the sources. The Likelihood Sequence is a key construct in our incident management framework. It not only drives the dispatch strategy for the business-as-usual (BAU) way of routing tickets one team at a time, but is also the sequence our SCU-based approach would refer to, to help decide on the composition and dynamic reconfiguration of the SCU during an incident resolution activity. The Likelihood Sequence may be expected to evolve as investigation proceeds and better understanding is achieved regarding the nature of the problem.

ITSM services will be provided by a set of practitioners (human resources) R. Each $R_i \in R$ has a skill profile $SP(R_i) = \{(C_p, f_q, l_r) | C_p \in V, f_q \in C_p^F, l_r \in L\}$, where L is an ordered sequence of skill levels $\{1, 2, ..., l\}$, with 1 being the lowest (most basic) and l being the highest skill level for any skill. A skill is the ability to investigate and correct a particular feature in a given component. Most practitioners will possess a set of skills, with varying skill levels, and the skills of a practitioner will usually be centered around different features in a single IT component, although there may be a few experienced practitioners with skills that span multiple components. Given this, each component is immediately associated with a *team* - a group of practitioners who have skills in one or more of the component features and may be called upon to investigate an incident that may have potentially been caused by the component. For a component C_i, this is given by $Team(C_i) = \{R_i | R_i \in R, \exists l \in L, \exists f \in C_i^F.(C_i, f, l) \in SP(R_i)\}$. For each combination (f, l) of a feature f and skill level l, we have an effort tuple $< E_{inv}(f, l), E_{res}(f, l) >$, which indicates representative (e.g. average) effort needed by a practitioner with skill level l in feature f to investigate if the feature is working correctly $(E_{inv}(f, l))$, and to restore the feature to working condition $(E_{res}(f, l))$ in case a fault is detected.

A Social Compute Unit $SCU(T_p, t)$ for a ticket T_p at a point in time t, may be represented by $< \mathcal{C}(t), \mathcal{R}(t), \mathcal{S}(t) >$, where $\mathcal{C}(t) \subseteq V$ is the set of components currently being investigated (i.e. at time t), $\mathcal{R}(t) \subseteq R$ is the set of practitioners that are part of the SCU at time t, and $\mathcal{S}(t)$ is an abstraction of the current *state* of the investigation, encompassing all components/features that have been verified to be functioning correctly till this point, all the faulty features that have been restored, and the all features that are currently under investigation or restoration. Thus a SCU is a dynamic entity whose composition (in terms of components and practitioners) as well as state (in terms of features investigated or restored), will continually evolve with time.

The management of an incident - from its creation to closure - will incur a cost, primarily due to effort spent by practitioners on various investigation tasks. We assume that when a practitioner joins an investigation effort, s/he will need to spend a certain amount of effort E_{init} to familiarize with the problem and the current investigation state. Subsequently, s/he will expend effort on feature investigation and restoration, commensurate with her skill levels. If s/he is part of a SCU, she will be expected to spend E_{collab} effort to collaborate with the larger team through the sharing of findings. This effort may be proportional to the duration of her stay in the SCU, as well as the size of the SCU during that period (in terms of the number of practitioners and components/features involved). If we wish to monetize the effort spent, we may assume a function $UnitSalary(R_i) : SP(R_i) -> \mathcal{R}$ (where $\mathcal{R}$ is the set of Real numbers), that returns the cost of unit effort spent by the practitioner, based on his/her skill profile. We also assume there is a certain amount of effort needed to set up and dissolve a SCU, given by SCU_{setup} and $SCU_{disolve}$ respectively. Also, in the process-driven sequential way of incident management, there will be a certain delay $D_{transfer}$ imposed each time a ticket is transferred from one team to the next. Finally, delays may be introduced due to unavailability of practitioners.

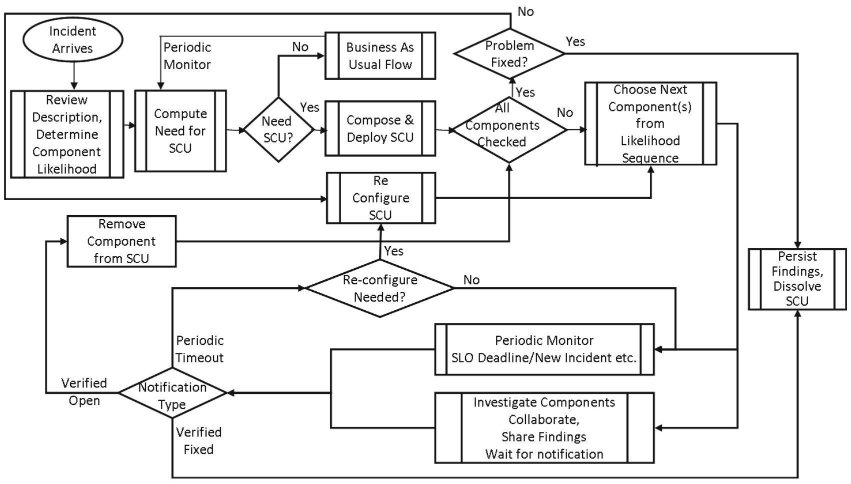

Incident Arrives
Review Description, Determine Component Likelihood
Periodic Monitor
Compute Need for SCU
Need SCU?
No
Yes
Business As Usual Flow
Compose & Deploy SCU
Problem Fixed?
No
Yes
All Components Checked
Yes
No
Choose Next Component(s) from Likelihood Sequence
Re Configure SCU
Remove Component from SCU
Yes
Re-configure Needed?
No
Persist Findings, Dissolve SCU
Periodic Timeout
Periodic Monitor SLO Deadline/New Incident etc.
Verified Open
Notification Type
Investigate Components Collaborate, Share Findings Wait for notification
Verified Fixed

Fig. 2. Incident Management Using SCUs

3.2 SCU Based Incident Management

Figure 2 depicts the overall incident management process that we propose. It is important to note that our SCU approach complements, rather than replaces, sequential process-driven incident management, represented in Figure 2 as the Business As Usual (BAU) flow. We do not expect a SCU to be required for every ticket, rather we base this decision upon the specific context of a ticket at a given point in time. When an incident arrives, the problem description should be reviewed, and the relative likelihood of different components being the source of the problem, has to be evaluated. This may be done by a human agent (e.g. a dispatcher) who uses a combination of ticket description and knowledge of component dependencies to identify potential faulty components. Supervised learning techniques such as Support Vector Machine (SVM) [15] may also be used to suggest for new tickets the likelihood (represented by a probability distribution) of each component being the source of the problem [2]. A combination involving a human dispatcher being assisted by an automated agent is also possible. It may be noted that such a likelihood evaluation is anyway done (even if implicitly) as part of a standard incident management process, since the dispatcher has to decide each time s/he receives a ticket, which component team needs to be contacted next to investigate the problem. Also, it is not necessary for the entire likelihood sequence to be generated as soon as an incident arrives. Instead, this may also be done incrementally, by considering at any point in time which are the most likely components that may be the cause for the incident (taking into account components that have already been investigated), and involve only those teams in the next phase of investigation.

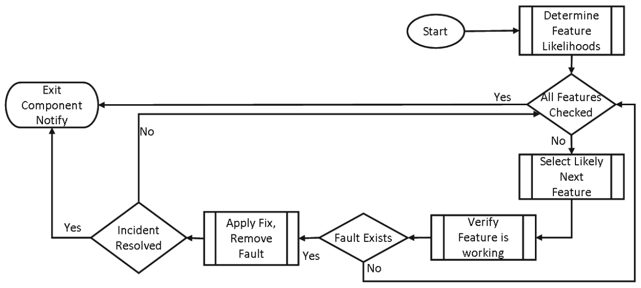


Fig. 3. Investigation Within A Component

Once this initial analysis has been done, we need to decide whether or not to invoke a SCU. In case the ticket deadline is sufficiently far away and/or there is a very strong likelihood of one particular component being the source of the problem, then the system may decide to follow the BAU mode, in which the ticket is dispatched to the most likely faulty component, where a practitioner will have to investigate it (this is explained in more detail later in the context of Figure 3). However, there will still be a need to monitor the situation so that in case the deadline is approaching without the root cause been detected, then a decision may be taken to set up a SCU to accelerate the investigation.

In case the BAU mode is not deemed appropriate in view of an impending deadline or lack of clarity in the problem description, a SCU may be invoked. Here, a few of the more likely faulty components are identified, and a set of practitioners who have the necessary skills in these components are on-boarded to the SCU. The decision on how many such components to consider, how many practitioners to on-board, what their skill levels should be etc. may be taken based on availability and the urgency of the situation. For example, if there is a crisis situation with the deadline of a high priority ticket being near at hand, then highly skilled practitioners from most/all of the component teams may have to be involved. In less urgent cases, one practitioner per component, for a small number of components at a time (say, 2-3) may be sufficient. Once on-boarded, the practitioners representing a component would try to determine if the incident has been caused by a fault in one of its features, using the process depicted in Figure 3. Here, a practitioner would proceed through the feature list of a component in the order of their relative likelihood of having a fault (as inferred by him/her based on the ticket description and understanding of features), and for each feature, investigate if it is correctly functioning, and if not, apply a fix to restore the feature. If the fix resolves the incident as well, then any ongoing investigation will be stopped across all components. Suitable findings from the investigation process will be harvested for later reference, and the SCU will be dissolved (Figure 2). Otherwise, the practitioner may move on to the next feature. If the current practitioner(s) representing a component do not have all the skills needed to cover the complete feature set, then on completion of their investigation, they may be replaced by other suitable practitioners. Note that this basic approach towards investigating a component remains the same whether a BAU or SCU mode is used. In case multiple practitioners are investigating the same component together in a SCU, they may partition the feature list amongst themselves to ensure there is no redundancy of effort. Also, within a SCU, a practitioner will be expected to collaborate with others e.g. through the sharing of findings, as shown in Figure 2.

As investigation proceeds, it is necessary to periodically monitor the situation and take appropriate action (Figure 2). For example, if the incident deadline is approaching, then there would be a need to re-configure the SCU by on-boarding more practitioners to cover other components. In case a high priority ticket arrives that needs the attention of a practitioner who is currently part of another (relatively less urgent) ticket's SCU, then the practitioner may have to leave the SCU and be replaced by another suitably skilled colleague who is available. Again, if all the features of a component have been verified to be functioning correctly, then the component may be removed from the SCU and the corresponding practitioners off-boarded. New SCU members may then be added from the next likely set of component(s). Finally, in the unusual case when all components have been investigated without the defect being identified, the SCU may be re-constituted and re-run, with higher skilled practitioners if needed.

4 Experiments

To experimentally evaluate our proposed SCU-based approach for incident management, we have designed an event driven simulator that mimics the flow of tickets through an IT service delivery environment. We first describe the experimental set-up, and then present the results.

4.1 Experimental Set-Up

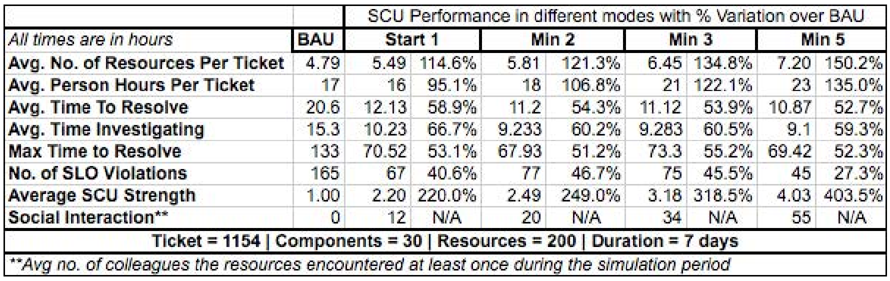
The simulation framework is built on java and has 4 major components: Events Generator that generates standard events related to incidence creation and management; Ticket Lifecycle Simulator, which manages various timers and notifications related to a ticket; Delivery Model, which includes the basic models of all the system entities (tickets, components, features, resources etc.) and relationships, and whose generated runtime is directly used within the simulation framework; and SCU Runtime Manager, designed as a library for implementing a SCU model in a service delivery environment.

For our experiments, we have considered an IT system with 30 components, with each component having between 0 to 5 dependencies generated as a random graph. For generating ticket data, we used a power law probability distribution of tickets across components, which is suitable for generating Pareto-like long tailed distributions. Based on our experience from working with large ITSM organizations, we have set this closer to a 70:30 distribution, which means that only 30% of the components cause 70% of the tickets. Overall, we generated 1154 tickets to cover a 1 week period of study, and maintained a resource pool of 200 practitioners to ensure a reasonable service rate. The ticket arrival rate is modeled as a stochastic process using a Poisson distribution initialized by average hourly arrival rates of tickets as we have seen in several actual service delivery environments. We assumed 4 different priority levels for tickets, with SLOs of 4 hours (highest priority), 8 hours, 24 hours and 48 hours (lowest priority) respectively. The relative distribution of the priority levels, were 2% (highest priority), 8%, 20% and 70% (lowest probability). All these values were selected based on our review of multiple ticket sets and SLOs. We assumed each practitioner to have skills in all the features of one component (which is often the case since practitioners in such environments are usually specialists in a particular technical domain). The staffing levels of each component were determined based on their relative workload (in terms of number of tickets received, the number of features, and the effort needed to investigate and fix each feature). Each ticket was assigned a fault profile of a single feature in one component. The likelihood sequence of each ticket was generated carefully by assuming the faulty component to be amongst the most likely ones in a high percentage of cases, but we also generated tickets with unclear root causes, where the faulty component occurred later in the sequence (with a probability that decreased progressively as the likelihood decreased). Moreover, we adjusted each sequence to ensure that the position of a likely component correlated well with that of some of its neighbours in the dependency graph, so that these neighbours were likely candidates as well.

We studied two modes of incident management - a fully process driven BAU mode, and a heterogeneous mode of BAU and SCUs. In the former, a ticket is investigated by one practitioner from one component at a time, and whenever a ticket has to cross organization boundaries, we assumed a delay of 30 minutes to account for the process-related overheads. This is actually a conservative estimate, since in real life service environments we have found tickets to be stuck for hours or days together in transfer between the components, and this was a key motivation for the SCU. In the heterogeneous mode, SCUs were automatically assembled for every highest priority ticket. For the rest of the tickets (including those initially dispatched in BAU mode), the decision to compose a SCU was based on the urgency of the situation at a given point in time. We used 4 levels of urgency, based on distance from the SLO deadline, and gradually increased the span of a SCU to cover more components (while having a single practitioner per SCU component) as the ticket moved from one urgency level to the next higher one.

4.2 Results and Discussion

The table in Figure 4 presents the results obtained from our simulation-based experiments. The column BAU stands for the mode where only process-driven sequential investigation was carried out for each ticket, while the rest of the columns involve situations where the BAU mode was complemented by SCUs. We experimented with different variations of this latter mode. We started with a conservative policy of initializing each SCU with a single component (Start1), but still investing in the SCU set-up cost (e.g. for getting the collaboration platform ready) in anticipation of the need to onboard more practitioners. In the other variations, we initialized each SCU with 2, 3 and 5 components. Once set up, a SCU was, of course, allowed to expand in size by onboarding more components, as the ticket progressed towards its deadline.



		SCU Performance in different modes with % Variation over BAU							
All times are in hours	**BAU**	**Start 1**		**Min 2**		**Min 3**		**Min 5**	
Avg. No. of Resources Per Ticket	4.79	5.49	114.6%	5.81	121.3%	6.45	134.8%	7.20	150.2%
Avg. Person Hours Per Ticket	17	16	95.1%	18	106.8%	21	122.1%	23	135.0%
Avg. Time To Resolve	20.6	12.13	58.9%	11.2	54.3%	11.12	53.9%	10.87	52.7%
Avg. Time Investigating	15.3	10.23	66.7%	9.233	60.2%	9.283	60.5%	9.1	59.3%
Max Time to Resolve	133	70.52	53.1%	67.93	51.2%	73.3	55.2%	69.42	52.3%
No. of SLO Violations	165	67	40.6%	77	46.7%	75	45.5%	45	27.3%
Average SCU Strength	1.00	2.20	220.0%	2.49	249.0%	3.18	318.5%	4.03	403.5%
Social Interaction**	0	12	N/A	20	N/A	34	N/A	55	N/A
Ticket = 1154 | Components = 30 | Resources = 200 | Duration = 7 days									
***Avg no. of colleagues the resources encountered at least once during the simulation period*									

Fig. 4. Experimental Results

We compare the performance of BAU and SCU modes along two main dimensions: effort and time to resolve. In terms of effort spent, the BAU mode is, in general, more efficient than the SCU mode. This is because in the former, only a single practitioner is being assigned at a time to conduct an investigation (on the most likely component at that point), while in the latter, multiple practitioners are assigned, and the aggregate effort invested is likely to be higher. Thus both the metrics Average Number of Resources Per Ticket and Average Person Hours Effort Per Ticket (aggregated across all resources who worked on a ticket) shows an increase as we go from BAU to SCU mode, and across the different variants of SCU modes. The only exception to this is when we start a SCU with a single component, in which case the Average Person Hours Effort Per Ticket decreases by about 5% relative to the BAU mode. This is an interesting case, and one probable reason for this may be that the Start1 mode, being at the boundary between full BAU and SCU modes, is able to effectively leverage the advantages of both, combining the power of quicker resolution with the low initial effort to bring down the total effort per ticket.

While the overall effort spent in SCU mode is, in general, higher, the collaborative investigation power of a SCU also significantly reduces the time to resolution, as seen from the values of the metrics Average Time To Resolve, Average Time Investigating and Max Time to Resolve. In all of these, the performance of the BAU mode lags far behind that of the SCU modes. From the business impact perspective, the most compelling case for the SCU comes from the dramatically improved SLO performance that results from its faster resolution of tickets, with Number of SLO Violations ranging between only 27.3% to 46.7% of the corresponding number for BAU. With the stringent penalties that IT vendors have to pay for poor SLA performance, the financial implications of this are far reaching. It may also be noted that while the SCU approach may consume more aggregate effort from practitioners, this does not necessarily translate to higher costs for the vendor. This is because, vendors typically maintain a dedicated team to provide production support to customer systems, and the effort available from these practitioners, if not utilized, may go waste and result in under-utilization, even though the vendor would still have to bear the same costs in terms of employee salary.

While a SCU has the flexibility to scale up as needed, we find that the average SCU size at any point in time (or its "strength") ranges from 2.2 to 4.03. While this may also partly be due to resource unavailability that prevents it from growing very large (since there will be many other tickets that keep practitioners engaged), the size is small enough for easy governance. Finally, we see that by virtue of being in a SCU, a practitioner is able to increase his/her sphere of interaction substantially, and the average number of colleagues they collaborate with during this brief period of study ranges from 12 to as high as 55. There are several long-term benefits an organization can derive from this that we intend to study in detail going forward, as mentioned in Section 6.

5 Related Work

We believe the novelty of our work is in the usage of SCU teams in problem resolution of otherwise sequential services processes. For example, the authors of [5] developed SYMIAN, a simulation framework for optimizing incident management via a queueing model based approach, which identifies bottlenecks in the sequential execution of ticket resolution. SYMIAN is based on the current implementation of incident management in the IT service provider organizations. Our approach is different from [5] as it takes into account optimization of resolution time through parallelization of work effort in the context of otherwise sequential execution of work.

A number of researchers have looked at the problem of mapping tickets to teams based on the problem description. For example, [14] develops a Markov model by mining ticket resolution sequences of historical tickets that were transferred from one team to another before being resolved. In [10] supervised learning techniques are used to determine the likelihood that a service request corresponds to a specific problem symptom; prior performances of teams are then analyzed to identify groups that have the necessary skills to resolve these problems. In [12] an auction-based pull model for ticket allocation is proposed, along with incentive mechanisms for practitioners to be truthful about expected resolution time when bidding for tickets. Unlike our approach, however, none of these works consider dynamic team formation to facilitate faster resolution of tickets through collaborative problem solving. The use of component dependency graphs in the incident management process has also been explored [11,9]. However, these have mainly been used to correlate problems and to search possible solutions rather than to automatically establish a suitable team for solving problems.

Human-based tasks, e.g., in BPEL4People [4], can be used to specify human capabilities or certain management tasks, e.g., by utilizing human-specific patterns [13]. However, this model relies on specific, pre-defined management processes which are not suitable for complex problem resolution, as we have discussed in this paper. Crowdsourcing [6,7] has been employed for solving complex problems, but while it also offers parallel computation power, our approach is distinct in its use of *social collaboration* to harness complementary skills within an organization and drive towards a common goal.

6 Conclusions and Future Work

This paper is a starting point into a broader mission to investigate the interplay of service-oriented and social computing concepts. So far we have introduced the fundamental concept of Social Compute Units and in this paper demonstrated the cost-benefit aspects of SCU's for a typical enterprise process. Whereas we believe the initial results from our simulations based on real-world experiences from the service delivery business of a large IT Service provider are very promising, future work will address the following:

1. Our current model assumes the SCU to be an organizationally implemented work model, i.e. skill and availability of resources will drive SCU formation. Social computing has a richer set of mechanisms, such as incentive and rewards, that are not yet part of our framework
2. The culture of collaboration that an SCU will nurture should have several long-term benefits in terms of knowledge management and enhancement in skill profiles. We will incorporate these in our framework going forward.
3. An important next step is to realize this approach in a real service delivery environment.

References

1. http://www.gartner.com/it/page.jsp?id=1470115
2. IBM SPSS, http://spss.co.in/
3. IT infrastructure library. ITIL service support, version 2.3. Office of Government Commerce (June 2000)
4. WS-BPEL Extension for People (BPEL4People) Specification Version 1.1 (November 2009), http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cd-06.pdf
5. Bartolini, C., Stefanelli, C., Tortonesi, M.: SYMIAN: A Simulation Tool for the Optimization of the IT Incident Management Process. In: De Turck, F., Kellerer, W., Kormentzas, G. (eds.) DSOM 2008. LNCS, vol. 5273, pp. 83–94. Springer, Heidelberg (2008)
6. Brew, A., Greene, D., Cunningham, P.: Using crowdsourcing and active learning to track sentiment in online media. In: Proceeding of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence, pp. 145–150. IOS Press, Amsterdam (2010)
7. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM 54(4), 86–96 (2011)
8. Dustdar, S., Bhattacharya, K.: The social compute unit. IEEE Internet Computing 15(3), 64–69 (2011)
9. Gupta, R., Prasad, K.H., Mohania, M.: Automating ITSM incident management process. In: International Conference on Autonomic Computing (2008)
10. Khan, A., Jamjoom, H., Sun, J.: AIM-HI: a framework for request routing in large-scale IT global service delivery. IBM Journal of Research and Development 53(6) (2009)
11. Marcu, P., Grabarnik, G., Luan, L., Rosu, D., Shwartz, L., Ward, C.: Towards an optimized model of incident ticket correlation. In: Integrated Network Management (IM), pp. 569–576. IEEE Press, Piscataway (2009)
12. Deshpande, P.M., Garg, D., Suri, N.R.: Auction based model for ticket allocation in IT service delivery industry. In: IEEE SCC (2008)
13. Russell, N., van der Aalst, W.M.P.: Work Distribution and Resource Management in BPEL4People: Capabilities and Opportunities. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 94–108. Springer, Heidelberg (2008)
14. Shao, Q., Chen, Y., Tao, S., et al.: Efficient ticket routing by resolution sequence mining. In: 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2008)
15. van Gestel, T., Suykens, J.A.K., Baesens, B., et al.: Benchmarking least squares support vector machine classifiers. Machine Learning 54(1), 5–32 (2004)