Applied Computing Review

Dec. 2013, Vol. 13, No. 4

Frontmatter

Invited Paper

Selected Research Articles

SIGAPP FY'13 Quarterly Report

October 2013 – December 2013
Sung Shin

Mission

To further the interests of the computing professionals engaged in the development of new computing applications and to transfer the capabilities of computing technology to new problem domains.

Officers

Chair	Sung Shin South Dakota State University, USA
Vice Chair	Jiman Hong Soongsil University, South Korea
Secretary	Michael Schumacher University of Applied Sciences Western Switzerland, Switzerland
Treasurer	Tei-Wei Kuo National Taiwan University, Taiwan
Webmaster	Hisham Haddad Kennesaw State University, USA
Program Coordinator	Irene Frawley ACM HQ, USA

Notice to Contributing Authors

By submitting your article for distribution in this Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library and in any Digital Library related services
- to allow users to make a personal copy of the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will refer requests for republication directly to you.

A Message from the Editor

I am happy to release the winter issue of *Applied Computing Review*. This issue includes one invited paper, two selected papers presented at the 2013 *ACM Symposium on Applied Computing (SAC)*, and three from the 2013 *ACM Research in Adaptive and Convergent Systems (RACS)*. *RACS'13* was held in Montreal, Canada, and it successfully provided a professional forum to share novel ideas within the areas of adaptive and convergent computing systems. The selected papers have been reviewed, revised, and expanded for inclusion in ACR, and I can proudly tell you that each and every one of them maintains high quality.

Our goal is to provide you with a platform for sharing innovative thoughts among professionals in various fields of applied computing. We have provided excellent service to various technical communities and to the scientific computing society in a productive manner. In addition, we are working with the ACM SIG Governing Board to further expand SIGAPP by increasing membership and developing a new journal on applied computing in the near future.

I would like to take this opportunity to announce that the fifth international conference on Future Energy Systems (ACM e-Energy) will be held in Cambridge, UK next year. More information about e-Energy can be found below. Finally, once again, I would like to remind you of the 29th *SAC* which will be held in Gyeongju, South Korea, from March 24th to 28th, 2014. As we did for the first time in the previous year, *SAC'14* will also offer students an opportunity through the *Student Research Competition (SRC)* program to meet with scientific researchers in the world. Please join us and make *SAC'14* a great success. Your kind support and cooperation would be highly appreciated. Merry Christmas and Happy New Year! Thank you.

Sincerely,

Sung Shin
Editor in Chief & Chair of ACM SIGAPP

ACM e-Energy

SIGCOMM is sponsoring a conference on the intersection of IT and energy called e-Energy. It focuses on the areas of computing and communication for smart energy systems as well as energy-efficient computing and communication systems. The conference will be held in Cambridge, UK, from June 11th to 13th, 2014. Details about the conference can be found at http://conferences.sigcomm.org/eenergy/2014/.

Next Issue

The planned release for the next issue of ACR is March 2014.

SAC 2014 Progress Highlights

The 29th annual edition of the ACM Symposium on Applied Computing (SAC) will be held in the historic capital city of Gyeongju, Korea, Monday March 24 to Friday March 28, 2014, on the campus of Dongguk University. The *Tutorials Program* is planned for Monday; the *Technical Program* for Tuesday through Friday; the *Student Research Competition (SRC) Program* for Tuesday (display session) and Wednesday (presentations session), respectively; and the *Posters Program* for Thursday.

SAC 2014 has received 913 submissions, from 50 countries. To date and after completing the review process, 223 papers were accepted, giving the conference an acceptance rate of 24.42%. In addition, approximately 100 posters were invited for participation in the Posters Program. These posters went through the review process as papers. The SRC Program received 20 submissions. After being reviewed by the respected track committees, it is anticipated that 14 student research abstracts will participate in the SRC Program. The accepted abstracts will compete for three cash prizes ($500, $300, and $200) and winners will be recognized during the banquet event. The first place winner can proceed to the National ACM SRC program. Furthermore, 6 tutorials were reviewed by the organizing committee and invited to participate in the Tutorials Programs. The details are posted on the conference website.

#	Track	Submissions	Accepted Papers
1	BHI	18	5
2	CC	17	5
3	CIVIA	11	3
4	CM	9	3
5	COSYS	8	2
6	CSP	7	0
7	DADS	16	5
8	DM	33	10
9	DS	14	4
10	DTTA	25	8
11	EC	9	2
12	EE	24	7
13	EMBS	19	5
14	HC	33	5
15	HCI	21	4
16	IAR	21	5
17	IIF	7	2
18	IILE	15	4
19	MCA	27	8
20	MMV	33	8
21	MP	43	5

#	Track	Submissions	Accepted Papers
22	NET	20	5
23	OS	47	10
24	PL	19	7
25	PSC	3	2
26	RE	19	5
27	RS	23	6
28	SATTA	12	4
29	SE	63	16
30	SEC	38	8
31	SEGC	9	4
32	SGST	40	5
33	SOAP	14	4
34	SONAMA	54	11
35	ST	20	0
36	SVT	33	8
37	SWA	18	6
38	TRECK	13	5
39	UIG	8	2
40	WCN	37	9
41	WT	14	5

The planning is underway. Hotel information (Hilton Gyeongju) is posted on the conference website along with the reservation form for special rates. The organizing committee recommends attendees to book their reservations at the designed hotel while the rooms are available. January 31, 2014 is the deadlines for the special rate. The conference will provide shuttle service between the conference venue and the Hilton hotel. Detailed shuttle schedule will be posted on the website. Other travel and transportation information is also posted.

The registration system is now open for authors and attendees. Included in the registration fee, SAC will provide daily lunches, coffee breaks, a reception on Tuesday, and a banquet dinner on Thursday. The reception and

banquet dinner will be held at the Hilton hotel. In addition, the local committee is organizing a number of excursions. Details are posted on the conference website.

#	Country	Submissions
1	Afghanistan	1
2	Algeria	2
3	Australia	10
4	Austria	11
5	Belgium	11
6	Brazil	206
7	Canada	55
8	Chile	1
9	China	77
10	Colombia	2
11	Czech Republic	2
12	Denmark	3
13	Egypt	1
14	Estonia	2
15	Ethiopia	1
16	Finland	7
17	France	43
18	Germany	35
19	Hong Kong	3
20	India	50
21	Iran	1
22	Ireland	2
23	Italy	18
24	Japan	44
25	Jordan	2

#	Country	Submissions
26	Kenya	5
27	Korea	67
28	Lebanon	2
29	Luxembourg	4
30	Mexico	1
31	Netherlands	20
32	Pakistan	3
33	Peru	1
34	Poland	5
35	Portugal	23
36	Saudi Arabia	2
37	Senegal	1
38	Singapore	9
39	Slovenia	1
40	Spain	12
41	Sweden	9
42	Switzerland	6
43	Syria	1
44	Taiwan	29
45	Tunisia	7
46	Turkey	4
47	UK	23
48	USA	61
49	Uruguay	2
50	Viet Nam	3

The Steering and Organizing committees are pleased to have SAC 2014 in the historic city of Gyeongju. We invite you to join us this March, meet other attendees, enjoy the conference programs, and have a pleasant stay in Gyeongju and Korea. We hope to see you there.

On Behalf of SAC Steering Committee,

Hisham Haddad
Member of the Steering Committee
Member of SAC 2014 Organizing Committee

Migration-based Hybrid Cache Design for File Systems over Flash Storage Devices

Po-Chun Huang
Institute of Information Science, Academia Sinica
Taipei, Taiwan (R.O.C.)
pchuang.19840320@gmail.com

Yuan-Hao Chang
Institute of Information Science, Academia Sinica
Taipei, Taiwan (R.O.C.)
johnson@iis.sinica.edu.tw

Che-Wei Tsao
Dept. of Computer Science and Information Engineering, National Taiwan University
Taipei, Taiwan (R.O.C.)
bearman.sky@gmail.com

Ming-Chang Yang
Graduate Institute of Networking and Multimedia, National Taiwan University
Taipei, Taiwan (R.O.C.)
riddle216@gmail.com

Cheng-Kang Hsieh
Center for Embedded Networked Sensing (CENS), University of California,
Los Angeles, CA, USA
changun.tw@gmail.com

ABSTRACT

This work is motivated by the urgent need to enhance the reliability of file systems in battery-powered mobile computing systems and consumer electronics that utilize flash storage devices for data storage. In this paper, we use non-volatile RAM (NVRAM) in the cache design because of its non-volatility and random byte addressability properties. Specifically, we propose a migration-based caching strategy that exploits NVRAM to enhance the reliability of file systems and to improve the access performance by utilizing the localities of accesses and the characteristics of flash storage devices. Our experimental results demonstrate that the proposed strategy can significantly improve the performance and reliability of file systems in flash storage devices.[1]

Categories and Subject Descriptors

B.3.2 [**Design Styles**]: Cache memories; B.3.2 [**Design Styles**]: Mass storage (e.g., magnetic, optical, RAID)

General Terms

Design, Management, Performance, Reliability.

Keywords

Non-volatile RAM (NVRAM), flash memory, hybrid cache, file system.

1. INTRODUCTION

Flash storage devices (*e.g.*, SD flash cards, eMMC and solid-state drives (SSDs)) are widely used to store data in mobile computing systems and consumer electronics (*e.g.*, light-weight notebooks, smart phones, and digital cameras or camcorders). They have a number of desirable features, such as low power consumption, shock-resistance, and small size [1]. However, most file systems and cache systems are designed for traditional hard disk drives, and the characteristics of flash storage devices are not considered. For example, cache systems usually store data in the dynamic random access memory (DRAM) without considering the features of flash storage devices, so cached dirty data could be lost if the system crashes or power is lost. In addition, flash storage devices do not perform well in writing small random data. Leading semiconductor companies have announced the development of next-generation storage media called non-volatile RAM (NVRAM), e.g., phase-change RAM (PCRAM) and ferroelectric RAM (FeRAM) [18], which are byte-addressable; however, they are not cost-effective compared to DRAM and NAND flash memory. These observations motivate us to propose a cache design that enhances the performance and reliability of file systems in flash storage devices by exploiting NVRAM, while also considering the cost issue.

A flash storage device might be comprised of multiple NAND flash chips with a multi-channel architecture. A NAND flash chip consists of one or more sub-chips, each of which contains many blocks. A block has a fixed number of pages, and is the unit for erase operations, while a page is the unit for read-write operations. Because of the "write-once property," a page cannot be overwritten unless its residing block is erased. To enhance the access performance, the "out-place update" strategy is used to write data to free pages. Address translation is thus needed to map the logical addresses of the data to their corresponding physical addresses. If there is insufficient free space, the garbage collection function is activated to reclaim pages containing old data by erasing their residing blocks. Before a block is erased, all the pages with up-to-date data should be copied to free pages. This process is called "live-page copying." The block erasure and live-page copying operations cause serious performance degradation. As each block can only endure a limited number of erase cycles, "wear-leveling" is performed to distribute block erasures evenly so as to prevent wearing out some flash blocks prematurely.

Due to cost considerations, various address translation schemes have been proposed to reduce the main-memory space required by the translation information; however, most of the schemes do not perform well on small random writes

(compared to sequential writes) [5, 12, 21, 32]. Some native file systems for raw flash-memory media have been developed to better utilize the characteristics of the flash memory [7, 22, 24, 31]. A number of researchers have also considered how to improve the performance of NAND flash memory with a RAM cache, *e.g.*, [20, 23, 27], and how to exploit energy-aware technologies and data compression to improve flash-memory storage systems, *e.g.*, [4, 10, 13, 15]. Due to the significant performance impacts of flash memory management on file accesses, there are also proposals on the cooperative layered designs of flash memory management and file management, such as in [16]. Recently, NVRAM has been adopted as the cache to enhance the performance and reliability of flash storage devices [17] and native flash file systems [8, 19]. It has also been suggested that NVRAM could be used as the write cache for the metadata (*i.e.*, the system information and file attributes) of general-purpose file systems [9]. On the other hand, hybrid storage architectures with NVRAM and hard disk/flash memory are suggested to greatly enhance the access performance by redirecting access requests of different patterns to NVRAM and hard disk/flash memory [28, 30]. However, there has been little work devoted on exploiting NVRAM to enhance the reliability of file systems in order to protect both metadata and userdata (*i.e.*, the content of files), or to improve the performance of file systems by considering the characteristics of flash storage devices.

This work is motivated by the urgent need to enhance the reliability of file systems in battery-powered mobile computing systems and consumer electronics, especially when flash storage devices are used as secondary storage units. We propose a migration-based caching strategy that utilizes both DRAM and NVRAM in the cache system to enhance the performance and reliability of file systems by utilizing the access patterns and the characteristics of flash storage devices. Under the strategy, unmodified *clean data* and modified *dirty data* are cached in the DRAM and NVRAM respectively, and an enhanced caching architecture with a *dirty cache tree* design is employed to manage and search data efficiently. In addition, a page-based management scheme with a two-phase selection policy is introduced to manage the NVRAM space and allocate free NVRAM space appropriately. To evaluate the performance of the proposed strategy, we conducted a series of experiments on traces generated by the well-known benchmark "Iometer" tool [26] using a 64GB flash storage device simulated with a modified DiskSim simulator [1, 3]. The results demonstrate that the proposed strategy significantly improves the performance of file systems in consumer electronics that utilize flash storage devices for data storage.

The remainder of this paper is organized as follows. In Section 2, we present the system architecture and explain the motivation for this work. In Section 3, we describe the proposed migration-based caching strategy; and in Section 4, we discuss the performance evaluation. Section 6 contains some concluding remarks.

2. SYSTEM ARCHITECTURE AND MOTIVATION

Flash storage devices (*e.g.*, eMMC and SSDs) are usually regarded as secondary storage devices for battery-powered mobile computing systems and consumer electronics. They are accessed by the host through an I/O interface, as shown in Figure 1(a). Normally, a flash storage device uses a multi-channel architecture to access multiple flash chips simultaneously; hence, the device's read and write performance could be improved significantly if it contains multiple flash chips. The chips are usually managed by the flash translation layer (FTL), which emulates the underlying flash chips as a block device (or a disk drive) and is responsible for the address translation, garbage collection, and wear leveling functions. On the host side, the file systems usually implement a cache system in DRAM to enhance the read and write performance of the secondary storage. A file system usually contains metadata and userdata. The metadata is general information about the file system, such as system information, journaling information, and the attributes of files/directories, while the userdata is the content of the files and directories. Because of space allocation and caching performance considerations, a file system usually partitions the logical block address (LBA) space into consecutive clusters. All the clusters are the same size (*e.g.*, 4096B) and have the same number of consecutive LBAs. Thus, the cache system can store both metadata and userdata in a cluster unit.

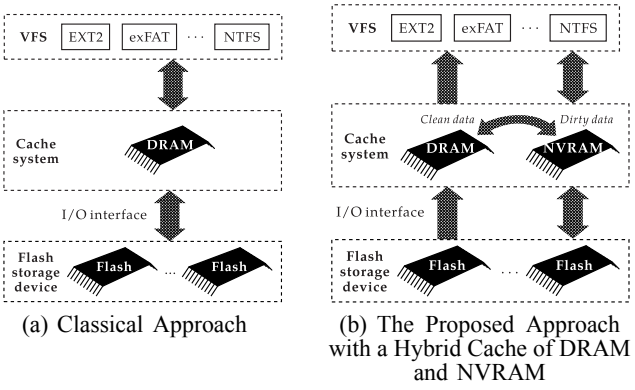


Figure 1. System Architecture

Table 1. Performance Comparison of DRAM and Different NVRAMs

Type	Access Time (nsec./2B)		Non-volatility
	Read	Write	
DRAM	20	20	×
MRAM	10–50	10–50	✓
PRAM	20–80	20–80	✓
FeRAM	30–100	30–100	✓

Although the cache system can enhance the read and write performance of file systems, the volatility of DRAM could cause the loss of cached dirty data (*i.e.*, modified data that have not been written back to the storage device) if the system crashes or a power failure occurs. As a result, the consistency/reliability of the data and the file-system's integrity could not be preserved in the event of system failure or a power outage. In contrast, emerging NVRAMs, such as MRAM, FeRAM, and PRAM, are not volatile and they can

preserve data after a power failure or system crash. However, their unit prices are much higher than those of DRAM and their read/write performance is worse than that of DRAM, as shown in Table 1 [17]. Moreover, as shown in Table 2 [25, 29], flash storage devices do not perform well on small read and write operations, especially small random writes. To improve the overall performance of data access between the host and the flash storage device, the number of small random writes should be minimized. In addition, most existing cache systems of file systems (*e.g.*, EXT3 and FAT32) are designed for mechanical hard disk drives and do not consider the characteristics of flash storage devices.

Table 2. Performance Comparison of Flash Storage Devices

Device	Sequential		Random (4KB)	
	Read	Write	Read	Write
Sony SD Card	2.83MBps	3.06MBps	2.39MBps	25KBps
STEC Zeus SSD	200MBps	100MBps	52KBps	11KBps

As mentioned earlier, this work is motivated by the need to enhance the performance and reliability of file systems with caching support when flash storage devices are used as secondary storage units. Our goal is to support both the DRAM and the NVRAM in the cache system by utilizing the cost simultaneously. Technically, we need to determine how to identify and manage dirty data in file systems efficiently, given the characteristics of flash storage devices and the limited size of NVRAM. To resolve the problem, we propose an efficient caching strategy, which we describe in the next section.

3. MIGRATION-BASED CACHING STRATEGY

3.1 Overview

To improve the reliability and read/write performance of file systems in battery-powered consumer electronics, we propose a migration-based caching strategy that exploits the characteristics of flash storage devices. As shown in Figure 1(b), the proposed strategy uses DRAM and NVRAM in the cache system to maintain clean data (*i.e.*, unmodified data) and dirty data (*i.e.*, modified data) respectively. If any clean data are modified, they are migrated to the NVRAM. In the cache system, the storage device's metadata is maintained by a device cache tree; the userdata in each file are maintained by a file cache tree; and all the cached dirty data are maintained by a dirty cache tree. All the trees are implemented as modified radix trees, which are space-efficient trees that are effective for looking up sequential data. (We discuss this aspect further in Section 3.2.) In NVRAM, the unit for caching data in file systems is a cluster. Our objective is to keep frequently updated data in the NVRAM as long as possible (based on observation of the temporal locality) and extend the average length of each write request (to improve the performance of flash storage devices). When the NVRAM is full, we initiate a two-phase procedure to select the cached data that must be deleted to allow space reclamation (see Section 3.3). In the first phase, we select sequential data from among the dirty data that has been updated the least recently, and then delete it in the second phase.

3.2 Enhanced Caching Architecture and Dirty Cache Tree

As mentioned in the previous section, the unit used to cache data is a cluster. To manage the cached clusters, operating systems usually adopt tree structures to maintain the cached data because trees have fast lookup and insertion properties. For example, Linux, which is used in Android phones, utilizes a (special) radix tree to manage the cached data in the DRAM-based cache system [2, 6, 11]. The radix tree is space-efficient, and its height can be adjusted dynamically based on the largest key value. As shown in Figure 2, each node contains a fixed number of slots that are indexed by a portion of the integer key of the cached data/cluster. Each slot in the root node and the internal nodes points to a node; each slot in the leaf nodes points to a page, where a page is used to store the data of one cluster of file systems. Note that dirty (resp. clean) pages are pages that cache dirty (resp. clean) data. Suppose each node in Figure 2 contains 2^n slots, where n is 6. If the largest key value of the cached data is less than 2^n, the entire tree can be represented by a single node. (Note that the height of a tree with a single node is 0.) When the largest key value of the cached data is between 2^n and $2^{2n}-1$, a new root node is created and the original root node becomes an internal node indicated by the first slot of the new root node. Therefore, if the largest key value of the cached data is between 2^{dn} and $2^{(d+1)n}-1$, the height of the needed radix tree is d. In a radix tree of height d, the least significant $(d+1)n$ bits of the key value are used for lookup operations. Then, each level of the tree can be used to index n bits of the key from the most significant bits of the $(d+1)n$ bits. The least significant n bits indicate the slot that points to the page where the corresponding cluster is cached. *Hereafter, we refer "page" as a page pointed by the cache trees in DRAM or NVRAM by default.*

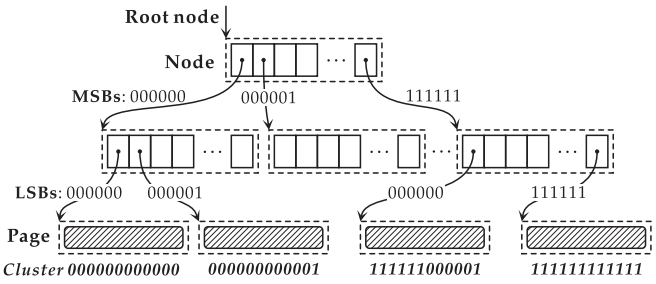


Figure 2. The DRAM-resident Radix Tree Used in the Cache System of Linux

Modern operating systems such as Linux usually adopt a tree structure (referred to as a *cache tree*) to maintain the frequently-used file system metadata and the file contents/userdata. Typically, the tree structure is saved over volatile DRAM, along with the metadata and the userdata. For example, in Linux, the system maintains a *device cache tree* to manage the cached system metadata. In addition, each accessed file usually has its own *file cache tree* (instead of using one tree for all accessed files) to maintain the cached content/userdata of the file. This enhances the lookup per-

formance for the cached content/userdata of each file. As shown in Figure 3, the device cache tree uses the cluster number (i.e., its sequence number in the device) as the key to manage the cached metadata. Each file cache tree uses the cluster index in the corresponding file to manage the cached userdata or content in that file, where the cluster index is the sequence number of the cluster in the file. Note that each cluster in a file is physically stored in a cluster of the device. As the cache trees and the cached metadata/userdata are saved over volatile DRAM, the file system might lose the cached metadata/userdata on accidental power failures, and the data consistency may be harmed. Due to this reason, NVRAM often becomes the alternative for the maintenance of the frequently-used metadata and userdata of file systems, in order to simultaneously enhance the performance and reliability of the file system.

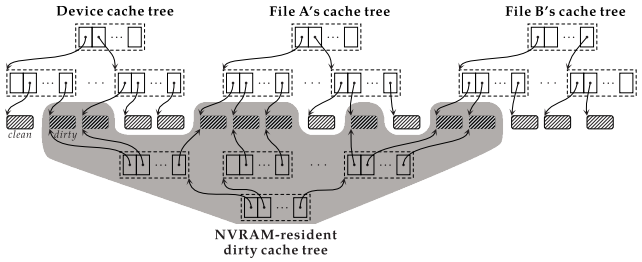


Figure 3. Enhanced Caching Architecture with Dirty Cache Tree

To ensure the reliability of file systems with a limited amount of NVRAM, we propose an enhanced caching architecture that maintains the clean data in the DRAM and the dirty data in the NVRAM. That is, clean pages and dirty pages are allocated to the DRAM and NVRAM respectively. To manage the dirty pages efficiently, we designed a *dirty cache tree* that is stored with the dirty pages in the NVRAM, where it functions as a radix tree with the cluster number (in the device) as the key. As shown in Figure 3, both clean and dirty pages are indicated by the device cache tree or one of the file cache trees; and the dirty pages are also indicated by the dirty cache tree. In this way, the file system can access all the cached data through the device cache tree or the file cache trees. Moreover, when there is insufficient NVRAM space to cache new dirty pages, the cache system can easily find old dirty pages to replace through the dirty cache tree. Therefore, the cache system only needs to reclaim clean pages when there is insufficient DRAM space, since all the dirty pages are cached in the NVRAM.

"Data migration" from the DRAM to the NVRAM occurs if written data were originally cached on a clean page. During data migration, a page (called NVRAM page) is allocated in the NVRAM and the written data is written to the (NVRAM) page. Then, the space for the original clean page is released so that the dirty data can be preserved if the system crashes. Note that a dirty page should maintain two pointers (see Figure 3). One points to its corresponding slot in the device/file cache tree, and the other points to its corresponding slot in the dirty cache tree. This is because, when a dirty page is reclaimed, the corresponding slot of the device/file cache tree should be notified, and when a dirty page is accessed by the file system, the least-recently-used (LRU) list maintained in the dirty cache tree should be updated accordingly (see Section 3.3.2).

3.3 Page-based Management Scheme with a Two-phase Selection Policy

3.3.1 Page-based Management Scheme

The proposed page-based scheme is responsible for managing the dirty pages and the dirty cache tree in the NVRAM space. As shown in Figure 4, the scheme partitions the NVRAM space into fixed-sized pages, each of which is used to store one dirty page or n_r ($= \lfloor s_b/s_n \rfloor$) nodes of the dirty cache tree, where s_b is the size of a NVRAM page and s_n is the size of a node in the dirty cache tree. Note that the first NVRAM page (referred to as the *metapage* in Figure 4) is reserved to maintain pointers that indicate the root nodes of the (cache) trees and the heads and tails of lists in the NVRAM. For example, in Figure 4, the first pointer *free-pg-head* indicates the first free NVRAM page, and each free NVRAM page indicates the next free NVRAM page to construct the *free NVRAM page list*. The second pointer *root-node* indicates the root node of the dirty cache tree, and the root node points to the other nodes in the tree.

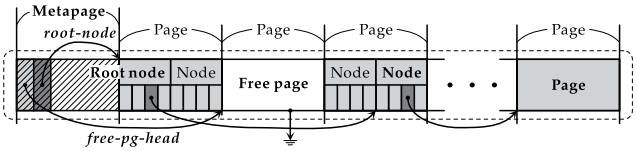


Figure 4. The Layout of NVRAM with the Dirty Cache Tree and the Cached Pages

When a new dirty page is created, it is allocated to the first free NVRAM page on the free NVRAM page list. If the data on a dirty page is flushed back to the storage device, the page is removed from all the cache trees that point to it, and its corresponding NVRAM page is returned to the free NVRAM page list. The page-based management scheme also maintains a *free tree node list*. When a new tree node is created and there is no free tree node on the free tree node list, it is assigned to a free NVRAM page. Next, the NVRAM page is partitioned into n_r free tree nodes, which are added to the above list and one of them is selected to store the new tree node. When a tree node is removed from the dirty cache tree due to the merging of two tree nodes or the deletion of the last page indicated by a leaf node, the removed node is returned to the free tree node list for future allocation. If the number of free tree nodes on the list is higher than a predetermined threshold, the NVRAM page with some free tree nodes or the largest number of free tree nodes is reclaimed by moving the used tree nodes in that NVRAM page to other free tree nodes. This reduces the number of free tree nodes and improves the utilization of the NVRAM space. The process is repeated until the number of free tree nodes is lower than the predetermined threshold. Although the process might degrade the caching performance, we can reduce its frequency by proper threshold selection. Of course, this is a trade-off between the performance and space utilization.

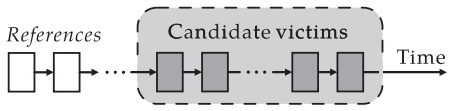
In practice, operating systems usually flush all dirty data

back to the storage device during shutdown; that is, the operation empties the dirty pages and the dirty cache tree in the NVRAM. However, if the system crashes or power is lost, the dirty cache tree would not be empty when the system is rebooted. Thus, all the dirty pages cached in the NVRAM must be flushed back to the flash storage device in order of the cluster numbers before any data in the file system can be migrated to the cache system. The operation ensures that all the data in the file system are preserved and synchronized. Then, the system can be rebooted as if it had been shut down properly. Although this might increase the time required to reboot the system, it minimizes the modifications required by the cache system and reduces the management complexity. The procedure used to flush dirty pages from the NVRAM to the flash storage device is very efficient because flash storage devices perform well on large sequential writes, and the dirty cache tree is very efficient in looking up dirty pages in order of the cluster numbers. In other words, with the dirty cache tree, it is possible to maximize the write performance of flash storage devices through large sequential writes. Note that if the shutdown performance is more important, the dirty pages could stay in the NVRAM without being flushed back to flash storage devices during shutdown; however, when a cluster is to be cached in the cache system after the system is rebooted, the dirty cache tree should be scanned to check whether the to-be-cached cluster is already cached in the NVRAM. This is because the device cache tree and the file cache trees are lost after the system is powered off.

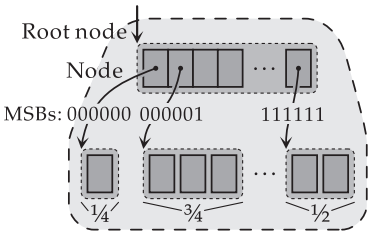
3.3.2 Two-phase Selection Strategy

When there is not enough NVRAM space, some dirty pages should be selected and be flushed back to the flash storage device, where the selected dirty pages are called *victim dirty pages*. In this work, a two-phase selection strategy is proposed to select victim dirty pages by utilizing the characteristics of flash storage devices to reduce the amount of data written back to the flash storage device and to increase the average length of data sequences in write requests. In general, data accesses have both temporal and spatial localities. Moreover, frequently written data sequences are usually small, so they are not suitable for flash storage devices. Thus, in the first phase, the selection strategy adopts an *LRU-based procedure* to identify the dirty pages that are unlikely to be referenced in the near future (Figure 5(a)). In the second phase, a *probability-based procedure* lets small least-recently-used dirty data sequences have a higher probability of staying in the NVRAM (Figure 5(b)). Consequently, the data will have more chances being aggregated into larger data sequences before being flushed back to the flash storage device.

As shown in Figure 5(a), in the first phase, the strategy uses a double-linked list to maintain an LRU list based on the times dirty pages were referenced. The list is implemented in the slots of leaf nodes in the dirty cache tree. That is, each slot of a leaf node should have two corresponding pointers to form the LRU list. As a result, when a cached dirty page is referenced, it can be moved to the tail of the LRU list in constant time without searching the whole list. As Figure 5(b) shows, in the second phase, a *victim pool* is maintained to manage a victim tree, which is a radix tree with the cluster



(a) Phase 1: The LRU List of the Candidate Victims



(b) Phase 2: The Candidate Victim Tree for Victim Selection

Figure 5. The Two-phase Selection Policy

number as the key to keep the least-recently-used n_v dirty pages (in order of cluster number); n_v is a predetermined threshold that is based on the size of the NVRAM and the access patterns of the adopted applications. In this phase, to find consecutive clusters of the storage device, we implement a probability-based policy to run a circular scan to the dirty pages indicated by the victim tree in order of cluster number. If the number of consecutive clusters is large (resp. small), the data on the dirty pages in the clusters have a higher (resp. lower) probability of being written back to the underlying flash storage device, and the NVRAM pages occupied by the dirty pages will be returned to the free NVRAM page list. For example, the probability that the data of a dirty cached cluster is written to flash memory is just the number of cached clusters indexed by the same node of the candidate victim tree divided by the maximal number n_r of entries in each node of the candidate victim tree. This increases the average length of write requests.

In practice, we can maintain a counter in each slot of the leaf nodes in the victim tree to record how many times a dirty page have been scanned without being flushed back to the flash storage device. If the counter of a dirty page reaches a predetermined total number k of times (*i.e.*, k times), the data of the cluster in the dirty page is forcibly flushed back to the flash storage device together with other dirty pages that contain consecutive clusters of the cluster stored in this dirty page. This is called the "multiple-chance policy" (or "k-chance policy") and it prevents any dirty pages remaining in the victim tree for a long time. Even short data sequences should be flushed back to the flash storage device if they have not been referenced for a long time so as to improve the utilization of the NVRAM space.

3.4 Implementation Remarks

3.4.1 File-level Concerns for Cache Management

When file information is available to the cache system, the proposed cache strategy could be extended to further enhance the space utilization and performance of the hybrid cache. Essentially, this extension is based on an observation that the access behaviors of data/clusters are often related

to those of different files in many application scenarios. This is due to that files are inherent separations for data accesses of many applications. In particular, when some cached data are frequently accessed in the past but have not been accessed for a long time, it is a sign that the file with the cached data might be no longer used by the applications. In this case, all other clusters of the same file could be evicted from the NVRAM and written back to flash memory, so as to enhance the space utilization of the NVRAM. Another advantage of this aggressive eviction strategy is that simultaneously writing a large chunk of data to flash memory is typically much more efficient (in terms of write throughput) than writing only one cluster at a time, due to the multi-channel architecture of many modern flash-memory storage systems.

3.4.2 Reduction of Mounting/Un-mounting Time of Storage Device

As explained in Section 3.3.1, the operating system used to flush all data in the dirty cache tree to the storage device during shutdown to ensure the data consistency. However, since the NVRAM is non-volatile, it is actually not needed to flush the dirty cache tree; only the related information of the dirty cache tree, such as the root node of the dirty cache tree, needs to be written to a specific area of NVRAM (such as the beginning pages of the NVRAM) when the storage device is un-mounted. During the next time the storage device is mounted, this information would be loaded back from NVRAM to DRAM, so as to restore the state of the dirty cache tree on NVRAM. Here, please note that there should still be a bit field in each tree node to indicate whether the node is the root node of the dirty cache tree or not. This field is necessary to prevent the loss of the NVRAM state information on accidental power failure. With this implementation technique, the time to mount or un-mount the storage device could be considerably reduced, since the whole dirty cache tree no longer needs to be written to or loaded from the storage device when the storage device is mounted or un-mounted.

4. PERFORMANCE EVALUATION

4.1 Experimental Setup

In this section, we evaluate the performance of the proposed migration-based caching strategy on different access patterns and NVRAM configurations. The access patterns are based on various average-size requests and different percentages of random requests; while the NVRAM configurations are based on different-sized NVRAMs and different-sized victim pools. We compared our strategy's performance with that of a cache system that only caches read data and write data back to the flash storage device directly (referred to as the cache system without NVRAM). The comparison is fair because both approaches can maintain the sanity and data consistency of file systems if the system crashes or power is lost.

In this experiment, the probability-based policy of the proposed strategy gave sequential data (*i.e.*, data in the victim pool) that was not larger than 4KB and not smaller than 256KB a 5% chance and a 100% chance of being selected respectively. For sequential data between 4KB and 256KB, the probabilities of being selected were derived by interpolating the probabilities of 5% and 100% linearly based on the length of the data sequence. In addition, each NVRAM page could store 10 tree nodes, and the maximal number of free tree nodes in the NVRAM was set at 20. As shown in Table 3, a 64GB flash storage device was simulated by a modified version of the DiskSim simulator executed on the Linux operating system. The modified simulator could support simulations of multi-channel flash storage devices with multiple chips [1, 3]. NVRAM was of 64MB with working frequency 400MHz when the NVRAM configuration was not clearly stated. The trace was collected with the "Iometer" benchmark tool [26] which ran on the EXT2 file system of Linux (where each cluster of EXT2 was 4KB). The total amount of written data was 1GB, and 50% of the collected access traces were random requests. The Iometer is an I/O subsystem measurement and characterization tool for single and clustered systems. It is widely regarded as the benchmark for evaluating the performance of storage systems on Linux.

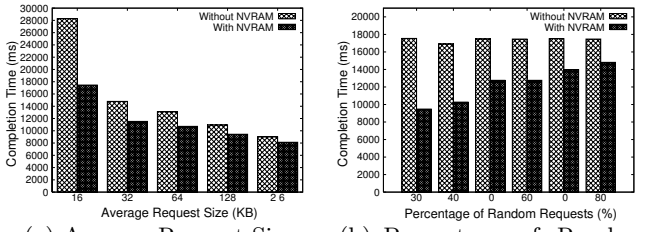
Table 3. Properties of the Simulated Flash Storage Device

Parameter	Value
Number of Channels	16
Number of Chips	16 (per Channel)
Chip Size	16384 Blocks (4GB)
Block Size	64 Pages (256KB)
Page Size	4KB + 128B
Read Time	25$\mu sec.$ Per Page
Write/Program Time	200$\mu sec.$ Per Page
Erase Time	1.5$msec.$ Per Block

4.2 Access Performance

4.2.1 Different Access Patterns

Figure 6(a) shows the performance of the proposed strategy on different average-size requests, where the x-axis denotes the size of the collected access pattern, and the y-axis denotes the completion time. The proposed strategy outperformed the cache system without NVRAM as the average request size was smaller. This is because the proposed strategy tries to flush more sequential data when the NVRAM cache is full. For example, when the average request size is 16KB, the proposed strategy outperforms the cached system without NVRAM by more than 38.2%.



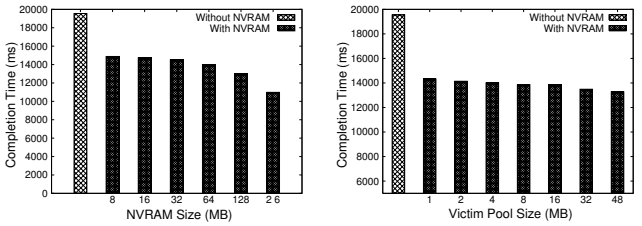
(a) Average Request Size (b) Percentage of Random Requests

Figure 6. Different Access Patterns

If a request is random, it does not access LBAs next to the LBAs (or clusters next to the clusters) accessed by the previous request. As shown in Figure 6(b), the proposed strategy outperformed the compared system by at least 15.2% in terms of the completion time. The completion time of the proposed strategy increased as the percentage of random requests increased. This is because it becomes harder to find longer data sequences in the victim pool when the percentage of random requests increases.

4.2.2 Different NVRAM Configurations

Figure 7 shows the performance of the proposed strategy on different NVRAM configurations. The strategy requires a shorter completion time as the size of the NVRAM increases (Figure 7(a)) because the NVRAM can cache more recently-used data; hence, the proposed strategy has a higher probability of finding a longer sequence of data to flush back to the flash storage device. For example, when the size of the NVRAM is 64MB, our strategy requires up to 28.34% less time than the cache system without NVRAM support.



(a) NVRAM Size (b) Victim Pool Size

Figure 7. Different NVRAM Configurations

As shown in Figure 7(b), when the size of the victim pool increases, the proposed strategy's completion time is shorter. This is because the proposed two-phase selection strategy can reclaim cached dirty data that will be referenced in the near future. Thus, a larger victim pool could help increase the average size of data sequences flushed to the flash storage device for each request.

5. ACKNOWLEDGMENT

This work is based on an earlier work "Performance and Reliability Enhancement for File Systems with Non-Volatile RAM over Solid-State Drives" in *the Proceedings of the 2009 International Workshop on Software Support for Portable Storage (IWSSPS)* [14].

6. CONCLUSION

This work is motivated by the urgent need to enhance the reliability of file systems in embedded computing systems and consumer electronics that utilize flash storage devices. To this end, we propose a migration-based caching strategy that considers the access patterns, access performance, and characteristics of flash storage devices. In particular, we introduce an enhanced caching architecture with a dirty cache tree design to manage data efficiently, where clean data and dirty data are cached in the DRAM and NVRAM respectively. We also present a segment-based management scheme with a two-phase selection policy to manage the NVRAM space and allocate free NVRAM space appropriately. The results of experiments based on traces generated by representative benchmarks demonstrate that the proposed approach significantly improves the performance of file systems in flash storage devices.

In our future research, we will exploit the characteristics of various types of NVRAM to evaluate the proposed strategy and to boost the booting and shutdown performance of operating systems. We will also explore the possibility of utilizing NVRAM as a new layer between the DRAM and storage devices to form a hybrid multi-layer storage system.

7. REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *the USENIX Annual Technical Conference (ATC)*, pages 57–70, Jun. 2008.

[2] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, Third Edition*. O'Reilly, Nov. 2005.

[3] J. S. Bucy and G. R. Ganger. *The DiskSim Simulation Environment Version 3.0 Reference Manual, http://citeseer.ist.psu.edu/bucy03disksim.html*, 2003.

[4] L.-P. Chang and T.-W. Kuo. An Efficient Management Scheme for Large-Scale Flash-Memory Storage Systems. In *the ACM Symposium on Applied Computing (SAC)*, pages 862–868, Mar. 2004.

[5] Y.-H. Chang and T.-W. Kuo. A Commitment-based Management Strategy for the Performance and Reliability Enhancement of Flash-memory Storage Systems. In *the 46th ACM/IEEE Design Automation Conference (DAC)*, 2009.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[7] H. Dai, M. Neufeld, and R. Han. ELF: an Efficient Log-structured Flash File System for Micro Sensor Nodes. In *the Second International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 176–187. ACM, 2004.

[8] I. H. Doh, J. Choi, D. Lee, and S. H. Noh. Exploiting Non-Volatile RAM to Enhance Flash File System Performance. In *the International Conference on Embedded Software (EMSOFT)*, Sept. 2007.

[9] I. H. Doh, H. J. Lee, Y. J. Moon, E. Kim, J. Choi, D. Lee, and S. H. Noh. Impact of NVRAM write cache for file system metadata on I/O performance in embedded systems. In *the 2009 ACM Symposium on Applied Computing (SAC)*, pages 1658–1663. ACM, 2009.

[10] Y. Du, M. Cai, and J. Dong. Adaptive Energy-aware Design of a Multi-bank Flash-memory Storage System. *the 11th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2005.

[11] Eklektix Inc. *Trees I: Radix Trees, http://lwn.net/Articles/175432/*, March 2006.

[12] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings. In *the 14th International Conference on Architectural Support for*

Programming Languages and Operating Systems (ASPLOS), Mar. 2009.

[13] H. Gyu and N. Chang. Energy-aware Memory Allocation in Heterogeneous Non-volatile Memory Systems. In *the International Symposium on Low Power Electronics and Design (ISLPED)*, 2003.

[14] C.-K. Hsieh, Y.-H. Chang, C.-W. Chang, and T.-W. Kuo. Performance and Reliability Enhancement for File Systems with Non-Volatile RAM over Solid-State Drives. In *the International Workshop on Software Support for Portable Storage (IWSSPS)*, Oct. 2009.

[15] Y. Joo, Y. Cho, D. Shin, and N. Chang. Energy-aware Data Compression for Multi-level Cell (MLC) Flash Memory. In *the 44th ACM/IEEE Design Automation Conference (DAC)*, pages 716–719, 2007.

[16] W. K. Josephson, L. A. Bongo, K. Li, and D. Flynn. DFS: A file system for virtualized flash storage. *ACM Transactions on Storage (TOS)*, 6(3):14:1–14:25, Sept. 2010.

[17] S. Kang, S. Park, H. Jung, H. Shim, and J. Cha. Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices. *IEEE Transactions on Computers (TC)*, 58(6):744–758, 2009.

[18] K. Kim and G.-H. Koh. Future memory Technology Including Emerging New Memories. In *the 24th International Conference of Microelectronics*, May 2004.

[19] C. Lee and S.-H. Lim. Efficient logging of metadata using NVRAM for NAND flash based file system. *IEEE Transactions on Consumer Electronics (TCE)*, 58(1):86–94, Feb. 2012.

[20] J.-H. Lee, G.-H. Park, and S.-D. Kim. A New NAND-type Flash Memory Package with Smart Buffer System for Spatial and Temporal Localities. *Journal of Systems Architecture (JSA)*, 51:111–123, 2004.

[21] S.-W. Lee, W.-K. Choi, and D.-J. Park. FAST: An Efficient Flash Translation Layer for Flash Memory. *Lecture Notes in Computer Science (LNCS)*, 4096:879–887, 2006.

[22] S.-H. Lim and K.-H. Park. An efficient NAND flash file system for flash memory storage. *IEEE Transactions on Computers (TC)*, 55(7):906–912, 2006.

[23] J.-H. Lin, Y.-H. Chang, J.-W. Hsieh, T.-W. Kuo, and C.-C. Yang. A NOR Emulation Strategy over NAND Flash Memory. In *the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2007.

[24] Aleph One Company. Yet Another Flash Filing System (YAFFS), *http://www.yaffs.net*. Technical report, 2012.

[25] Mtron. *MSD-SATA3025 Product Specification*, 2008.

[26] Open Source Development Lab (OSDL). *Iometer User's Guide, http://www.iometer.org/ doc/documents.html*, 12 2003.

[27] C. Park, J.-U. Kang, S.-Y. Park, and J.-S. Kim. Energy-Aware Demand Paging on NAND Flash-based Embedded Storages. In *the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2004.

[28] Y. Park, S.-H. Lim, C. Lee, and K. H. Park. PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash. In *the 2008 ACM Symposium on Applied Computing (SAC)*, SAC '08, pages 1498–1503, New York, NY, USA, 2008. ACM.

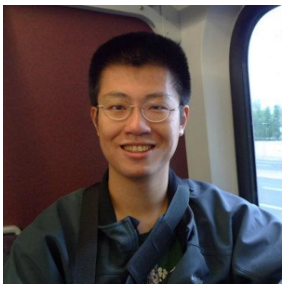
[29] STEC Incorporation. *ZeusIOPS Solid State Drive Specification*, 2007.

[30] A.-I. A. Wang, G. Kuenning, P. Reiher, and G. Popek. The Conquest file system: Better performance through a disk/persistent-RAM hybrid design. *ACM Transactions on Storage (TOS)*, 2(3):309–348, Aug. 2006.

[31] D. Woodhouse. JFFS: The Journalling Flash File System. In *Ottawa Linux Symposium*, 2001.

[32] C.-H. Wu and T.-W. Kuo. An Adaptive Two-level Management for the Flash Translation Layer in Embedded Systems. In *the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 601–606, 2006.

ABOUT THE AUTHORS:



Po-Chun Huang received his Ph.D. degree in Computer Science and Information Engineering from National Taiwan University in June 2012. He worked as a teaching assistant while pursuing his Ph.D. degree, and he received the Best Teaching Assistant Award in 2012. Also, he received the Third Place Award in the Embedded System Design Contest in 2008 and 2009 respectively. Once graduated, he worked as a postdoctoral research fellow at Institute of Information Science, Academia Sinica. His primary research interests include storage systems, operating systems, embedded systems, computer system architectures and cloud computing systems.

Yuan-Hao Chang received his Ph.D. degree in Networking and Multimedia of Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 2009. Now he is an assistant research fellow at the Institute of Information Science, Academia Sinica, Taipei, Taiwan, since Aug. 2011. Previously, he was an assistant professor at the Department of Electronic Engineering, National Taipei University of Technology, Taipei, Taiwan, between Feb. 2010 and Jul. 2011. His research interests include storage systems, embedded systems, operating systems, and computer system architecture.

Che-Wei Tsao is currently pursuing his Ph.D. degree in Graduate Institute of Networking and Multimedia in National Taiwan University, Taipei, Taiwan (R.O.C.). Formerly, he received his M.S. degree in Graduate Institute of Computer and Communication Engineering in National Taipei University of Technology, Taipei, Taiwan (R.O.C.) in June 2012. Now, he serves in R&D alternative military service at Institute of Information Science, Academia Sinica. His main research interests include storage systems, real-time operating systems, embedded systems, and disk IO communication protocols.

Ming-Chang Yang received his B.S. degree in Computer Science from National Chiao-Tung University, Hsinchu, Taiwan, in 2010. He received his M.S. degree in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 2012. Now he is a Ph.D. candidate in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, and is also a research assistant in the Institute of Information Science of Academia Sinica, Taipei, Taiwan. His primary research interests include storage systems, embedded systems, and next-generation memory architectures.

Cheng-Kang Hsieh received the BS degree in computer science and information engineering from National Taiwan University in 2009. He is currently a PhD student in UCLA Computer Science. Previously, he was a research assistant in the Research Center for Information Technology Innovation (CITI), Academia Sinica, Taiwan. His research interests include mobile systems, storage systems, and personal data analysis.

SimPal – A Design Study on a Framework for Flexible Safety-Critical Software Development

Jesper Pedersen Notander
Dept. of Computer Science,
Lund University, Sweden
jesper.notander@cs.lth.se

Per Runeson
Dept. of Computer Science,
Lund University, Sweden
per.runeson@cs.lth.se

Martin Höst
Dept. of Computer Science,
Lund University, Sweden
martin.host@cs.lth.se

ABSTRACT

This paper presents the findings from a design study on a framework for flexible safety-critical software development, called *SimPal*. It is an extended version of a paper that was published in SAC'13 Proceedings of the 2013 ACM Symposium on Applied Computing, in which additional details about *SimPal* as well as a more extensive evaluation of the framework is presented. The objective is to identify necessary quality properties and to learn more about the challenges of realizing frameworks such as SimPal. We approach our research questions by developing a framework and by analysing our experiences from the design and evaluation process. Some necessary quality characteristics has been identified by discussing the ISO25010 *quality in use* quality model in relation to the problem domain, which were then used to design and evaluate the developed framework. The evaluation was conducted as a design case in which a *soft safety controller* was developed following the methodology outlined in the paper. We show that our approach, which tries to merge service-oriented practices with model-based development techniques, has potential considering safety-critical software development. However, there are some concerns about run-time performance as well as the ability to qualify the tool for safety-critical development. Based on our results we conclude that the ideas behind the *SimPal* framework are sound but more work is required to investigate how they can be realized. For the future, we plan on further investigating the code generating capabilities of the modelling tool we are using to see if and how it can be utilised to increase performance. We also plan on adding more features to the framework, for instance coordination and configuration of services, as well as monitoring of messages and system properties.[1]

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design, Verification

Keywords

Frameworks and Tools, Verification, Safety-Critical Development, Real-Time Systems, Service-Oriented Computing

1. INTRODUCTION

Safety critical software development is characterized by long lead times due to inflexible development processes and rigorous requirements on verification and validation of the source code and the overall system. Traditionally plan driven process models have been used and many certification standards stipulate or implicitly assume the use of such a process model [1]. Plan driven processes paint an appealing picture of software development as a linear path starting with the conception of an idea towards design, implementation and finally verification and validation. Several standards advocate independence between activities [16, 9], i.e. different persons are needed to perform different activities, e.g. programming and reviewing code, which fits well with a linear process model, where one person is assigned one task.

In recent years the software industry has pushed towards more iterative and agile process models [12]. The reasons are several but the most interesting, for safety critical software development, is that agile processes capture requirement related problems early, e.g. not fulfilling requirements, wrong requirements, unrealistic requirements, missing requirements, unwanted or wanted requirements etc. In the traditional processes many of these problems are first found at a late stage in the development, where it is costly and time consuming to fix them. From a business perspective agile methods promises shorter lead times and better handling of changing requirements, which in turn promises faster deliveries and releases, even larger profits and market shares [18].

To support a transition to *agile development* in safety critical software development, tools, methods and architectures need to be aligned with the strict assessment requirements, dictated by the safety standards and the flexibility offered by the agile processes.

Because safety at its core is a requirements management problem [11], tools and methods need to take into account both the changing nature of requirements and the verification and validation of them, preferably as early as possible in the development process. The *model-based development* paradigm tries to solve this by using models of the system at different levels of abstraction to verify specific properties of the system [7]. If a property has been proven correct or tested to an acceptable level of correctness in a model, it is assumed that it will hold for the final system as well, i.e. if the implementation process is correct.

Another benefit of model-based development is that it can be combined with automatic code generation so that executable code can be generated directly from the models. The full

benefit of this union is achieved if the code generator is certified for use in *safety critical development*, because then it can be argued that the code generated will satisfy the properties of its model, thus further verification of these properties in the source code is superfluous. However, it might not be feasible due to high costs, non-deterministic behaviour or unspecified behaviour, to certify the code generator, although there exist examples where it has been done, e.g. SCADE from Esterel-Technologies. Despite that, code generation can still be useful for testing software and system level properties in a simulator or in a controlled test environment.

Generation of executable code is only the first step toward a fully integrated development environment for safety critical software, system integration and deployment must also be considered. *Service-oriented architecture* (SoA) provides mechanisms for publishing, discovering and integrating services. Although, SoA has primarily been used for non safety critical systems in the past [21] the core concepts fit very well with some of the basic design principles of safety critical software systems, e.g. code encapsulation, separation of concerns and memory/execution partitioning.

The goal of this paper is to identify some of the necessary characteristics of a framework for model-based flexible safety critical software development and to learn more about the challenges of realizing such a framework. To aid us in understanding the challenges, we developed a framework, called *SimPal*, based on our early assumptions about the necessary characteristics and applied it to the development of a soft safety controller in a mobile robotics system.

This paper is an extended version of a paper published in SAC'13 Proceedings of the 2013 ACM Symposium on Applied Computing [13]. In this version, additional details about *SimPal* has been included as well as a more comprehensive evaluation of the framework. We also present some more background on the problem formulation behind the framework.

The rest of the paper is structured as follows: We start by describing our research approach, followed by, presenting the characteristics we identified as necessary for a flexible safety-critical development framework. Then we move on to describe the SimPal framework: its design and how it is supposed to be used. The paper concludes with the evaluation of the SimPal framework, our conclusions and suggestions for future work.

2. RESEARCH METHOD

The research presented in this paper could be classified as design research [8, 2]. In design research the creation and evaluation of artefacts is used to increase our understanding of the phenomenon under study. Collins et al. [2] outline, what they conceive as, the seven steps of design research. The work in this paper focuses mainly on the steps: artefact design, problem relevance and evaluation of the design.

2.1 Design Input

The input to the design process was threefold. Most significant was our experiences from the *EnGroSS* project , which is a multi-disciplinary research project in the mobile robotics domain. The goal of the project is to show the utility of flexible software architectures, in a context with real-time and safety-critical constraints, by developing a mobile robot demonstrator.

Our second input was the preliminary results of a qualitative industrial survey on best practices in safety critical software development [14], which had impact on the analysis of the problem relevance and the initial framework evaluation.

Our final input was the outcome from the discussions among the authors during the design process. It should be noted that one of the authors has recent industrial experience of working with software in a safety-critical context.

2.2 Problem Relevance

In our discussions we identified the need for tools and methodologies that bind the different parts of a development process together, in particular we saw the need for a framework that is able to handle changing requirements, specification, verification and validation. Furthermore, we concluded from the survey results [14] that we need to be able to cope with component based systems, e.g. based on service-oriented architectures. An early assumption was that such a framework should be able to integrate with existing development processes and tools.

Moreover, we considered the following situation. In the EnGroSS project there are both software engineers and function specialists, mainly navigation, robot and vision specialists. Software is developed using PalCom [20], which is a service-oriented middleware that is built around *services* that are provided by *devices*, and coordinated and configured with *assemblies*.

Services are usually programmed by the function specialists directly using some high-level language, e.g. Java, C, or Python. In the case of PalCom this typically entails the extension of a base service class with additional function specific code. This means that the function specialist must understand the PalCom service and device API and have a good understanding of how the callback semantics of PalCom works. Furthermore, the developer needs to be knowledgeable about software engineering practices and how to produce high quality software.

Deployment, coordinating and configuring of services requires the use of *The Thing* and the *PalCom browser*, which are two PalCom specific programs that requires additional knowledge to use.

Function specialists should not spend time on software engineering tasks, instead they should focus their effort on designing, and testing algorithms, which can be done in a suitable modelling tool. However, at a certain point algorithms need to be deployed on their target platform to fully test whether they work as intended or not. At that point, currently two options exists, either the function specialist writes the software implementing the algorithm or a software engineer is given the task. The first option is sensitive to knowledge gaps and the second to communication gaps.

Based on this problem formulation and the aim of gaining further insights into the technical challenges associated with the kind of frameworks we had identified the need for, we

developed a prototype tool called SimPal. The purpose of the tool was to simplify the development of software components by integrating the software platform used in the EnGroSS project with a model-based development tool, by doing so we saw that the effort of integration, deployment and testing could be reduced. In particular, we wanted to hide the underlying software platform PalCom from the function specialists, by integrating it with Matlab and Simulink.

2.3 Research Questions

Based on the initial discussions and the problem formulation above we formulate the following research questions.

RQ1 Which quality characteristics of a development framework are necessary in flexible safety-critical software development?

RQ2 What are the challenges with employing model-based design in a flexible safety-critical software architecture?

The first question is addressed by presenting our effort to identify what we consider to be the relevant quality characteristics of a framework intended for flexible safety-critical software development. Our identification is based on a literature study and the preliminary results from the industrial study previously mentioned [14].

The identification of the quality characteristics was done by mapping capabilities that we consider necessary or desirable in a flexible safety-critical framework to the quality model of ISO25010 [10], using informed arguments [8]. We chose the standard because its quality model, which is an extension of the one presented in ISO 9126 [6], considers not only technical properties of software but also properties that are related to the use of software, Moreover, it is considered to be reasonably accurate both for industrial practice and academic work.

The analysis began with the definition of the premises including the assumed context of the framework. In the next step we identified the capabilities by reasoning about challenges, constraints and properties of a flexible framework from four different aspects: *model-based development*, *agile development*, *service-oriented architecture* and *safety-critical software development*. The choice of these aspects was primarily based on the constraints on the research project as a whole and to a lesser degree from a suitability perspective regarding safety. Furthermore, we consider them to be highly relevant as they correlated with current trends in the safety-critical software industry. Finally, the capabilities were, to the best of our abilities, mapped to the quality models of ISO25010.

Our approach to the second research question is to report on the observations and experiences gained during the development of a soft safety controller for the mobile robotics platform in the EnGroSS project, using the SimPal framework.

The development of the soft safety controller was conducted in iterations, according to our proposed work methodology. First, a pure software system was developed running on a desktop computer. Next we deployed a dummy service on the device that was going to communicate and relay scan data from the laser scanner on the robot platform, a Raspberry-PI. Finally, we replaced the dummy service with a service that connect to the laser scanner and relays scan data to all listening PalCom services. In addition, performance measurements were conducted for some iterations.

3. FRAMEWORK CHARACTERISTICS

In this section we identify the necessary characteristics of a model-based framework for flexible safety-critical software development, using a standard model and related work in several fields. The resulting properties are summarized in Table 1.

3.1 Premises

We have based our analysis on the quality models of ISO25010. Our main effort was put in identifying the desired quality in use sub-characteristics of the development framework and specific external quality attributes.

Quality in use is a system level quality model that considers not only software but also: hardware, operating environment, users, tasks and social factors. It is divided into three main characteristics: *usability*, *flexibility* and *safety*. To be able to apply the model, the context of use for the analysed system must be defined, as the characteristics is defined in terms of it.

External quality attributes are software quality attributes that can be measured on a software system when looking at it as a black box. They are thus mainly concerned with the interaction of the system with the outside world and the expectations on the system.

We have assumed in our analysis that the context of the framework is that it will be used inside a company that has prior experience with developing safety-critical systems and, because of this, has a process model and tool chain that conforms to applicable safety standards. It is however not assumed that there exist a seamless integration between different tools and activities in the process model.

3.2 Model-Based Development

Seamless integration with model-based methodologies have become more and more important with the increasing popularity and subsequent introduction of the model-based development paradigm in software development [7], especially in the safety-critical and real-time systems domain. Although, we do not feel it necessary at this time to accommodate the full range of proposed model-based techniques (i.e. full support of formal modelling techniques, e.g. model checking, constraint programming, theorem solvers, Petri net modelling etc.) a framework should at least provide the equivalent functionality of industrial best practice.

Cost is an important aspect that must be taken into account because if it is too expensive to introduce a new framework, in relation to the expected gain, it will not be adopted by industry. Although the cost of new infrastructure in terms of better equipment and software might not be negligible the cost of training, especially in a resource constrained development environment, might be the greatest limiting factor. It follows that it should be easy to introduce such a framework in an existing development organization without too much effort spent on training and integration.

Table 1. Identified properties of a flexible model-based safety-critical development framework, mapped to the *quality in use* quality model in ISO25010.

Characteristics		Aspect	Argument
Main	**Sub**		
Usability	Efficiency	Model-Based Development	A1. A framework should not be harder to learn than already existing tools. It should be easy to use and be adaptable to current work procedures.
		Agile Development	A2. The development overhead of preparing software for verification, e.g. code instrumentation or wrapper code, might work as a deterrent against small incremental changes, thus a framework useable in an agile context would need to minimize such overhead, e.g. by providing a uniform deployment mechanism.
Flexibility	Context Conformity	Model-Based Development	A3. The framework should be easy to deploy in a development organization and be able to interface directly with existing tools.
	Context Extendability	Model-Based Development	A4. The framework should not be static in terms of functionality and development practices, and fully accommodate changing needs and new developments of best practices.
		Service-Oriented Architecture	A5. The architecture should allow flexible extensions beyond original intents of the system.
Safety	Safety Compliance	Safety-Critical Software	A6. The framework should be analysable and predictable, and be compliant with safety standards.

Another limiting factor is the risk of becoming dependent of a framework that cannot adapt to the evolution of its constituent parts. For this reason a framework must be able to continuously incorporate new functionality and be able to adapt to evolving practices or else it becomes obsolete.

3.3 Agile Development

The introduction of agile methodologies in safety-critical software development has implications for the design of a framework, and for the capabilities it should incorporate. The agile movement proposes that software should be developed in small steps with few new functions added in each iteration, but fully tested and in a working condition, i.e. after each iteration the delivered source code should be executable and meet the requirements of the implemented functions [4]. Thus, for a framework to be successful in an agile context it must be able to handle incremental development, including verification and validation.

Software components, which is understood to loosely mean *a collection of software functions with some commonality*, in the real-time domain tend to be developed in isolation by function specialists, for later use in larger systems. Thus, considering the agile way of incrementally adding functionality, these systems will be composed of smaller parts that are added, removed, changed and integrated with each other over time. Herein lies a conflict with the verification and validation of system level properties, e.g. safety.

The model-based approach to solve this conflict is to define the desired properties in a system level model and formulate the requirements that satisfy the properties as constraints on the components of the system. Although this is appealing in theory, in practice it is much more difficult because properties of real-time systems depends on the target hardware platform. For the model-based approach to work, it would be necessary to have accurate hardware models, which might not be possible, e.g. missing data about the hardware and the target environment.

In the absence of accurate hardware models, simulators and test environments can be used to gain confidence in the software implementation. However, normally the use of such techniques generates an overhead in the form of additional effort spent on interfacing the software, e.g. instrumentation or writing code wrappers, with the simulator or test environment, thus working as a deterrent against changing the software in small steps. It could thus be argued that a framework should provide an uniform mechanism for interfacing software components with test environments, simulators and the target platform, so that the overhead of deploying and running it would be independent of the target platform. For such a mechanism to be useful it must be efficient and not incur too high a run-time performance overhead.

3.4 Service-Oriented Architecture

Service-oriented architecture (SoA) is a fundamental architectural model for software systems. The architecture consists of services, which are self-contained reusable software components, provided by a service provider and consumed by a service requester [21]. SoA originates from the Information Systems (IS) domain and web services, where the SoA provides a flexible approach to adapt the IS to changing business needs. As an architectural concept, SoA may guide activities at different levels of abstraction; programming, middleware and business process levels [21].

More recently, attempts have been made to introduce SoA concepts into the embedded systems domain, specifically in pervasive systems [20], i.e. systems that tries to connect and coordinate devices in our environment to satisfy user needs [17]. The SoA model provides a feasible way when trying to meet demands on flexible interconnection of devices and composition of services. In contrast to fixed system architectures, a key characteristic of SoA is that it supports interoperability between various services by providing general service concepts, while the exact content of the service may change. This characteristic present additional opportunities for evolution of SoA-based systems. However, the implications for safety certification of SoA systems are not yet fully explored [15].

3.5 Safety-Critical Software

Development of safety-critical software is traditionally slow-paced and process-focused, as depicted by several safety standards, e.g. IEC 61508 [9] and EN50128 [5]. In contrast, the class of systems we discuss, in which flexibility and evolvability are key characteristics, requires a more agile development approach. However, such an approach should still be the subject of operator and public safety considerations as well as being compliant with safety standards.

Previous work on SoA in safety critical systems, involve approaches that utilize the SoA for non-critical parts only [15], as well as middleware for run-time verification [3]. The aim for us is to enable the flexibility of SoA not only for non-critical parts, but also for critical parts of a safety-critical system, which excludes the first approach. The run-time verification approach allows flexibility but must be efficient enough for the real-time behaviour of the system.

When introducing new tools and methods in a safety-critical development context it is imperative that they conform to the applicable safety standards. Depending on the standard the amount of evidence and assessment activities varies, e.g. IEC 61508 says that development tools should be certified or be *proven by use*, in some cases it would suffice to establish their *fitness for purpose* [19]. For a tool to be assessable it must be analysable and predictable, it must be transparent to the user and the user should be confident that it works as expected.

4. FRAMEWORK DESIGN

In this section we describe the SimPal framework. It was developed to increase our understanding about the necessary characteristics of a flexible development framework in the context of in safety-critical development, and how such frameworks can be realized. SimPal is built around a development tool, which combines the PalCom middleware with the commercial modelling tool Simulink.

PalCom was originally developed for the pervasive computing domain, but has recently been proposed as a suitable architecture for real-time systems, in large part, due to its strong emphasis on loose-coupling of configuration, coordination, communication and computation.

Simulink on the other hand has been used for many years and have become akin to an industrial *de facto* standard for real-time systems modelling. It is particularly strong at modelling control algorithms and other data flow based algorithms. With the inclusion of Stateflow Simulink has increased its ability to model state-full and event-driven systems. Furthermore, it has several more appealing features with regard to the capabilities identified in the previous section, e.g. it is fairly easy to integrate with other software.

SimPal consists of three parts: the development tool, some general architectural constraints that must be satisfied by the developed software, and the development methodology presented in this section, which should be followed to gain the full advantage of the framework.

In this section we will describe the two main technologies used in SimPal: PalCom and Simulink. After that, the design of the SimPal tool, the architectural constraints of the developed software and the intended work methodology for working with SimPal are presented.

4.1 PalCom

PalCom is a service-oriented architecture [21] that was originally developed for the pervasive computing domain [17]. It emphasizes loose coupling between configuration and coordination of services and describes a protocol for discovering and describing services on a network [20].

The top-level entities in the PalCom architecture are: *devices*, *services* and *assemblies*, which are connected by *connections* and exchange data with typed messages, called *commands*, asynchronously. All entities have unique and persistent ids and are able to describe themselves on the network.

A device can either be a software representation of a physical device that are tightly coupled to the hardware or a virtual device, like *The Thing*, which is a device that can dynamically load and execute services and assemblies. Another important virtual device is the *Palcom Browser*, which is used to discover entities in a PalCom network and to construct assemblies.

A service is either bound, unbound or synthesized. Bound services are tightly coupled with a certain device whereas unbound services belong to The Thing. An unbound service can migrate between different instances of The Thing in the network. Synthesized services are service interfaces, provided by assemblies, which expose some kind of desired capability offered by the assembly. Each service carries a description of the commands it requires and provides. A command is a named message that contains a list of typed *parameters*.

Assemblies describe configuration and coordination of devices and services, i.e. which services interact with each other and how do they interact. The semantics of the assembly language is fairly simple. Coordination is achieved by specifying a list of event handlers that reacts to events, which can be incoming messages or events related to connections. A typical reaction is to extract some parameters from an incoming message and put them in a new message, which is then sent to another connected service. Configuration on the other hand, is performed by binding services and devices to the assembly using their unique IDs, and specifying which connections that should be established between the services.

At the start of the design process, there existed two implementations of PalCom, one in Java and one in C. Although,

the C version has significantly better run-time performance than the Java version the latter was chosen for SimPal. The reason for that is that the C version does not implement the full PalCom protocol and that it was considered harder to integrate the C version with Simulink than the Java version. Since then, an additional C version has been developed, targeting small resource constrained embedded systems.

4.2 Simulink

Matlab is a tool that many engineers come in contact with during their education and later in industry. It is a powerful tool that can be used to design algorithms and build mathematical models. The power of the tool comes from the extendability of the software, especially the many toolboxes that are shipped with it that address specific problem domains, but also from the seamless integration of Java with its interpreter.

Simulink is a simulation and modelling tool fully integrated with Matlab that can be used to model algorithms using classic *boxes and lines* notation. It is mostly used to design and simulate control algorithms.

Over the years Simulink has been extended with several toolboxes that enable certain capabilities, e.g. *Stateflow*, which enables state machine based control algorithms to be modelled using a modified version of the UML statechart notation, and *RealTimeToolbox*, which makes it possible to generate C and ADA code directly from model diagrams for a specific target platform. The most common type of extension is the model libraries that contains the building blocks that Simulink models are composed of. Another interesting extension is the *Verification and Validation* toolbox that gives the ability to link models and blocks to a requirements database, e.g. Doors, or ordinary requirements documents, e.g. Word or PDF.

New blocks can be constructed in several ways. The most common is to create a subsystem containing the combination of blocks that will provide the desired functionality. Another way is to assign callbacks to a block, which are Matlab functions that are executed at predefined times of a block's execution cycle, e.g. at initialization-time or after termination. A third way to modify a block is to create a mask, thus making the content of the block opaque to the end user. When masking a block, certain properties can be hard coded or left to the user to fill in at a later time. The most powerful way to make a custom block is to use the level 1 and level 2 *s-function* blocks. The level 1 s-function block is a wrapper around C-code callbacks, which defines the block behaviour. The source code is compiled prior to model execution and can be linked to external libraries. Level 2 s-functions are wrappers around Matlab code that is interpreted when the model is executed. Because the interpreter is fully integrated with Java, Simulink can execute Java code using level 2 s-functions.

4.3 Integrated Development Tool

The main idea behind the design of the SimPal tool is to treat Simulink models as PalCom top-level entities, i.e. devices, services and assemblies, which can be dynamically deployed and executed on a network through the simulation facility

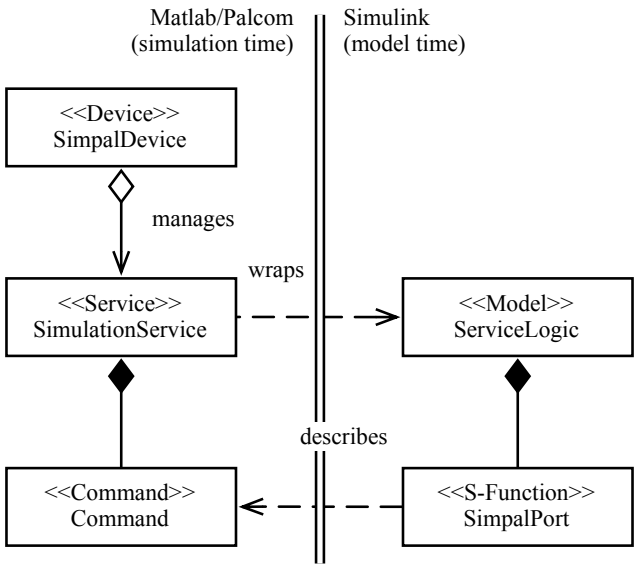


Figure 1. A simplified architecture of the SimPal development tool that shows the main entities and their relations. The dividing line is not a strict delimiter separating two logically independent parts, it is more an indication on what exist at design time (right side) and what exist at run-time (left side). For instance at simulation time, the *Model* entity, which corresponds to a Simulink model, is translated into a *SimulationService*.

in Simulink. Depending on the type of entity a model corresponds to, its content will be interpreted differently. As an assembly, a model describes the connections and passing of messages between services and which synthesized services are provided. As a service, a model describes a service interface, i.e. provided commands and their parameters, and the computing logic of the service. As a device, a model describes provided services and assemblies. In the current version of SimPal, only *model as a service* is implemented.

The SimPal tool consist of three parts: I) a Simulink library that contains s-function blocks that describe PalCom commands, which also corresponds to model inports and outports, II) a virtual PalCom device, called the SimPal device, which can dynamically load and unload services, and III) the simulation service, which is a generic PalCom service that can be dynamically loaded and configured by a Simulink model at simulation time. The SimPal device executes in the Matlab base workspace and provides access for the simulation services to the PalCom world. The simulation service is a communication wrapper that publishes the service interface that is defined by the SimPal inports and outports present in the corresponding Simulink model. In Figure 1 a simplified architecture of the tool is shown.

The SimPal Simulink library contains blocks for describing incoming and outgoing commands of a PalCom service, and are implemented as masked level 2 s-function blocks. A command block describes an incoming or outgoing service command with one parameter. The name of the command

and the type of the parameter is defined by the modeller in the block mask. Moreover, the command blocks can be used as sources and sinks in a model, which is to say that they can be regarded, from a modelling perspective, as ordinary model inports and outports. The difference is that instead of being ports to an enclosing model they are ports to the enclosing PalCom world.

The s-functions that the command blocks masks contain callbacks to the SimPal device. These callbacks are used during model simulation to, initially, create and configure a simulation service for the model, and then, for the remainder of the simulation, handle passing of message data between the model and the service. In detail, when a model containing SimPal command blocks is simulated a simulation service is created with an interface that is described by the models SimPal command blocks. It is then loaded into the SimPal device, which announces its existence to the rest of the PalCom network, which makes it possible to connect it with other services and assemblies. The service logic resides in the model and will execute according to the simulation policy chosen. It is assumed that the policy is discrete fixed-step simulation with infinite simulation time. SimPal will read and send data through the simulation service whenever the model state is updating. Thus, the full extent of Simulink's capabilities for monitoring, debugging and testing can still be used for verification and validation of the model. However, this have implications for the performance of the tool.

During the design of the SimPal tool the communication model of PalCom, which is based on asynchronous messages, had to be considered in relation to the execution model of the Simulink simulation facility. We also had to factor in that simulation time is different from real-time, i.e. simulation is done as fast as possible. We considered two cases, discrete event simulation and fixed step simulation. Note that the cases only applies for input commands, for output commands a model pushes data to the simulation service regardless of the simulation model.

In the case of discrete event simulation, a model executes whenever an event is generated in the model, and is inactive between events. For instance, we could let a SimPal inport generate an event whenever its wrapper service receives a command that corresponds to the inport. This would make the model reactive to incoming commands, which is in line with how most PalCom service are written. Due to difficulties with the threading model of Matlab and the design of the discrete event simulation in Simulink, we deemed this case to be infeasible to implement, at the time. However, we do believe this case will have to be addressed in the future.

In the case of fixed step simulation, a model executes with a predefined frequency. This is a simpler case than the discrete event simulation case because the model is pulling data from the SimPal device, as opposed to the SimPal device pushing data to the model. Although the frequency is specified in real-time units the model has to be modified to actually execute at that frequency. One way is to use a s-function that calls system functions that halt the execution for the desired amount of time. A key issue is that during the time the model is halted, the SimPal device should still be able to receive commands.

4.4 Methodology and Constraints

The usefulness of a tool is not determined by its existence but by how it is applied. During the design of the SimPal tool several assumptions, i.e. constraints, about the software architecture and development methodologies were made. In this section we summarize these assumptions and present our suggestion on how to use the tool in a development context. What is described herein should not be conceived as a full-fledged process model, which can be used *as is*. Instead, we envision that the ideas and concepts presented will be incorporated into an existing process.

Starting out with the architectural constraints, the service-oriented design paradigm lies at the core of the framework, which means that it should be possible to reason about software systems developed with SimPal in SoA terms. It follows that such a software system should be decomposable into services. We define a service as a software component that encapsulates some sort of functionality, which is exposed through a well defined interface to its environment, which can be the local device or several devices in a distributed system.

Borrowing the ideas from PalCom about loosely coupled coordination of services, we suggest that services should be loosely coupled, be as small as possible and that complex functionality is implemented using some sort of aggregated service construct, e.g. assemblies. Because we are mainly interested in real-time and safety-critical software systems there is also a need for the ability to express tight coupling of services to physical devices in the architecture.

The methodology that we suggest for SimPal assumes a work flow where software functions are defined at one level, using some kind of modelling language, and further refined at lower levels, with the lowest being the source code implementation. At each level, the functions would be verified using, for instance, formal verifiers, simulation and testing, against the previous level and constraints added at the current. Because SimPal has a uniform deployment mechanism, there is no clear dividing line between desktop testing, simulation or real world testing. Thus, it would be possible to reuse test cases developed for a higher level at lower levels. Although, this requires that the interfaces of the software functions are defined at an early stage and do not change that often.

5. FRAMEWORK EVALUATION

This section presents the evaluation of the SimPal framework. The evaluation was performed in three steps: first, the feasibility of the technical aspects of the framework was studied by developing a soft safety controller for a mobile robot; second, the framework was evaluated against the criteria identified in Section 3 by reasoning about the current implementation; third, further evaluation was conducted with the soft safety controller case, including live tests with hardware components.

5.1 The Soft Safety Controller Case

The aim of the EnGroSS project is to develop a technology demonstrator that implements the concept of loose coupling in a mobile robotics systems. The concrete case is to develop a mobile robot platform that can navigate and restock shelves in a grocery store.

5.1.1 Background

One part of the EnGroSS demonstrator that is not coordinated using PalCom, are the two safety laser scanners. The scanners detect obstacles and foreign objects moving inside their scanning range. If something is too close to the platform the scanners will activate the emergency stop, which immediately cuts the power to the wheels and forces the platform to stop. The emergency brakes are activated without regard to what other actions are currently performed by the system, which might damage the platform as well as what it was working with at the time, it might even create another hazardous situation. Consider the following two scenarios.

During a routine transport of some heavy goods by the robot platform, an employee intersects the robot's path. The laser scanners immediately activates the emergency stop and the platform starts decelerating. Under normal conditions everything would be fine but due to some unforeseen circumstances the load is unbalanced, which has the consequence that the robot tips and falls over the employee.

The second scenario unfolds in a grocery store during peak hours. The robot platform is restocking shelves, trying to avoid people by planning its path through less travelled aisles, when the fire alarm goes off. People try to evacuate as fast as possible through the nearest exits but there is a lot of people and confusion. Due to bad timing the robot platform finds itself blocking one of the emergency exits, unable to move because people are trying to get past it, constantly triggering the emergency stop.

The scenarios above might not seem especially likely but they show that there are situations where an emergency stop is not a safe state. Moreover, different contexts requires different constraints. For instance, when the robot platform is alone it can move at higher speeds because it does not need to consider that people can appear around corners. Whereas, when people surrounds it the speed at which tasks are performed should be reduced, so as not to cause concern or prevent it from stopping before a collision could happen. Taken together, it would seem that there is a need for a more intelligent safety controller.

The laser scanners used in the EnGroSS project supports partitioning of their scan area into several different safety zones. The reaction to a detected obstacle depends on the zone it was detected in. Although, this is a great improvement it is not flexible enough. The zones are configured offline which means that we cannot change them on-the-fly according to context, e.g. day time, night time. Secondly, we believe that a safety controller need additional sensor data to fine-tune its behaviour.

The idea behind the soft safety controller is to use software to plan and make the transition to a safe(er) state smoother by gracefully degrading the operation of the system. Ideally, by going beyond mere partitioning of the local area into safety zones, and by combining inputs from several different sources, e.g. laser scanners and cameras.

5.1.2 Design and Evaluation

For the intents and purposes of this paper, we have made a minimal implementation of the envisioned soft safety controller, to show the utility of the SimPal framework.

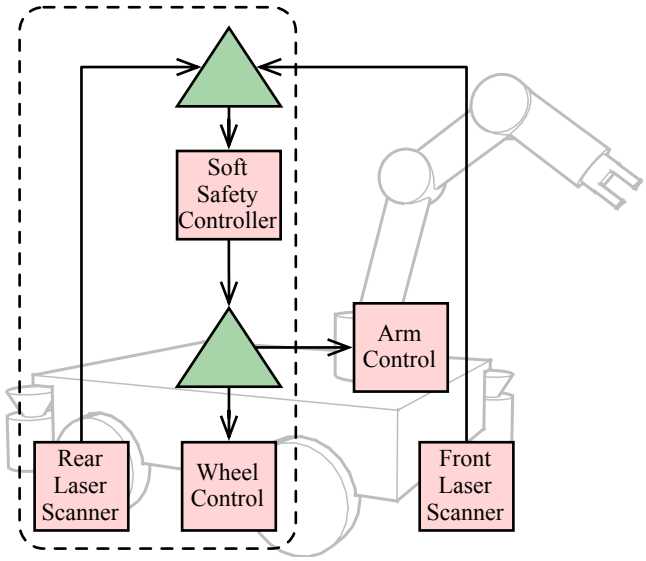


Figure 2. An overview of the robot platform services architecture for the soft safety controller during Phase 1, including coordination. Boxes represent services, triangles assemblies and lines flow of information. The dotted rounded rectangle encloses the components that were also considered in Phase 2.

The SimPal methodology is to gradually improve the software under development until it is mature enough to be run against the target platform. During the maturing process the use of models, simulators, and explicit test cases are encouraged. In line with this, we started out with developing a platform simulator that can generate scan messages and receive control commands. In the second phase we moved on to using a hardware laser scanner demonstrator, which was developed for the EnGroSS project. Both the simulator and the demonstrator provides the same scan data interfaces.

Phase 1. In the first phase, which was presented in the shorter version of this paper [13], we developed an initial version of the soft safety controller as well as a robot platform simulator, which provides several simulated services, e.g. a scanner service and a platform service. Figure 2 shows the components and their connections in this phase. Although, not explicitly shown in the figure, the platform simulator is represented by the *Control* and *Scanner* services.

The soft safety controller was modelled, using the SimPal tool, as a service that consumes collision data from a scanner service and calculates velocity constraints for the arm and wheel services, see Figure 2. We connected the soft safety controller service to the simulated scanner and platform services, running them all on the same computer. The connections were coordinated with PalCom assemblies. To gain further insights, we conducted a small performance test were we sent collision data from the platform simulator to the safety controller and received back velocity commands. The measured round-trip time was on the order of 200 ms.

Phase 2. In this phase, we have studied the implementation of an improved version of the soft safety controller from

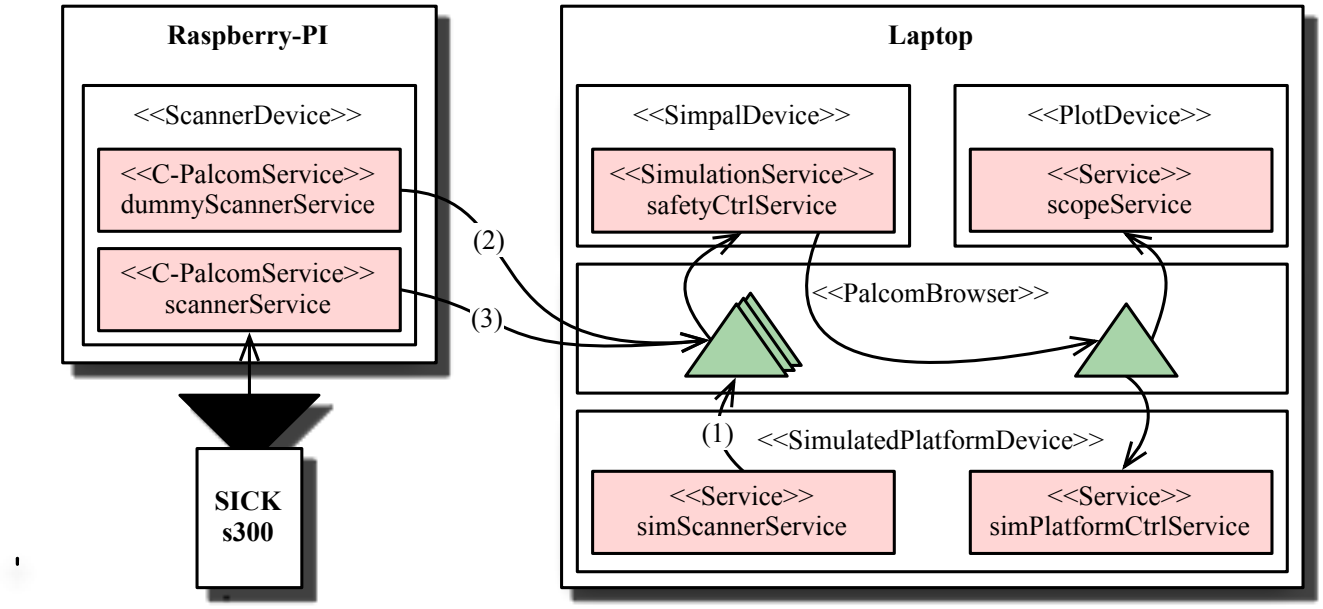


Figure 3. The deployment diagram of the soft safety controller system, showing all PalCom services and devices. Services are represented by red boxes, devices with white, assemblies with green triangles and physical devices with white boxes with black shadows. Arrows point in the direction messages flow. Three lines have been marked with a number, which indicates the order they were connected with the system.

Phase 1, using the SimPal tool, and evaluated it against a laser scanner demonstrator developed for the EnGroSS project. The demonstrator consist of a SICK s300 laser scanner connected to a Raspberry-PI, which publishes scan data through a PalCom service. In addition, we developed a generic plot service to monitor signals, e.g. scan data, during run-time.

In Figure 3, the deployment diagram of the soft safety scanner case can be seen. The numbers on the lines indicate the order in which scanner services were connected for the first time with the soft safety controller. First, we connected the new controller to the simulated scanner service that were used in the first phase, Secondly, we connected the controller to a dummy scanner service on the Raspberry-PI. Finally, we ran the controller directly against the real scanner service, also on the Raspberry-PI device, with real scanner data.

Two performance measurements were conducted. The results can be seen in Figure 4. The ”box plot” with label A, shows the results from measuring the time from a command is relayed from the scanner service to the safety controller and until a response is issued from the controller. The other ”box plot” shows the results from conducting a similar measurement as was done during the first phase.

5.2 Analysis

In this section we discuss strengths and weaknesses of the SimPal framework. We will also reason about our experiences from developing the soft safety controller. A summary of the analysis results, mapped to the identified properties in Table 1, can be seen in Table 2. [Ax] in the text below refers to argument x in Table 2.

The development of the soft safety controller showed that the SimPal methodology and tool worked well together to enable iterative development, which is a prerequisite for agile development [A2]. It was possible to start small with simple control logic, in a single service, and then gradually increase the complexity of the software system, by first connecting the soft safety controller service to the robot simulator, and finally to the real laser scanner. Although we did not have the opportunity to use the safety controller in the real robot system, we are confident that the integration effort would be similar to the cases we have considered in this paper, because the service interfaces are identical in all cases.

Simulink is used by many engineers both in academia and industry and there is a wealth of experience and knowledge available online. Most notably is the *Matlab Central*, managed by Mathworks, where users can exchange files, experiences and get the newest information about the product. Many engineering students, especially those studying automatic control, learn how to use Simulink and relevant toolboxes during their studies. Thus, it would seem that the threshold of learning to use the SimPal tool would be fairly low [A1].

Simulink has good support for model testing and simulation as well as tools for checking coverage. It also supports formal verification of system properties and static code analysis of generated code. Simulink’s built-in support for requirements tracing, enables requirements tracing from models to requirement management tools, e.g. DOORS, or plain documents, e.g. Word or PDF. In addition, there are several toolboxes for specific application domains, e.g. signal processing, model predictive control, power systems. It is also fairly easy to integrate Simulink with third party tools. Taken together,

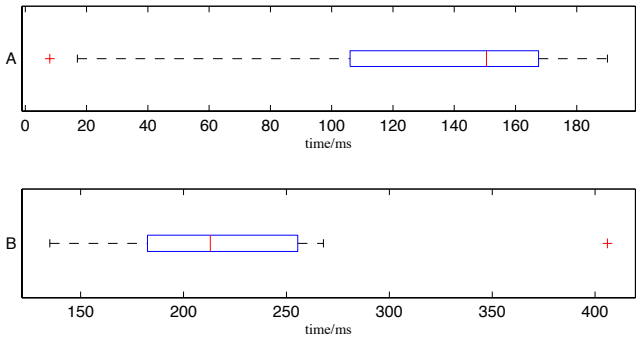


Figure 4. Box plots of the distribution of the time delay between messages. The upper shows the mean delay between incoming messages to the soft safety controller and the return messages. The bottom, shows the mean delay between outgoing messages from the simulated platform service and incoming messages to the simulated navigation service.

these features make Simulink, and hence SimPal, a viable candidate in any development process that uses some or all of these features [A4]. However, the complexity and closed nature of Simulink makes it hard to implement certain kinds of metaphors, for instance external event driven execution of models.

The flexibility of the PalCom architecture makes it possible to extend a system beyond what was originally intended. Due to the loose coupling of configuration and coordination, services can be added, removed and combined in a flexible way through the use of the assembly construct [A5]. For this to fully work it is imperative that the architectural constraints mandated by the SimPal framework are satisfied. Through the soft safety controller case, we have shown that SimPal together with PalCom facilitates the extension and evolution of an existing system, by providing good mechanisms for coordination and deployment of services.

To show compliance with applicable safety standards a tool must be analysable and have a predictable behaviour. Simulink is a commercial tool with closed source code, thus the responsibility of showing compliance with applicable standards falls on the vendor. PalCom on the other hand is open-source with freely available source code. The Java implementation is around 30k lines of code, where large parts are automatically generated. In comparison to the first C implementations of PalCom that is only about 1.5k lines of code. It would be reasonable to assume that the Java implementation is overly complex [A6].

SimPal uses the current reference implementation of PalCom, which is made in Java. The run-time performance of the reference implementation is an issue, even without the added overhead of using SimPal. If we measure the round-trip time in a small system, similar to the soft safety controller, it will be in the order of $20ms$ without SimPal and around $200ms$ with SimPal. As a comparison the round-trip time using the first C implementation of PalCom is $200\mu s$. From this, it would seem that the current implementation of SimPal is not

Table 2. Summary of the analysis mapped to the arguments in Table 1.

Analysis	ID
Simulink has a large user community and many engineers are taught to use it during their studies.	A1, A3
SimPal supports iterative development with its uniform deployment mechanism	A2
Simulink supports integration with third party libraries and tools.	A3, A4
Simulink has several features that are useful in safety-critical development, e.g. debugging, formal verification, coverage analysis, requirements tracing et.c.	A3, A6
PalCom enables flexible extensions of software systems	A5
The complexity of the Java implementation of PalCom makes it hard to show compliance with safety standards.	A6

viable for anything other than very small or non time critical cases. Switching from the Java implementation of PalCom to the C implementation seems to be a promising approach to increase the performance of SimPal. The C implementation also has the benefit that it can be used together with the code generation facility in Simulink thus enabling automatic code generation of the complete system.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have identified what we believe to be some of the necessary characteristics that a framework for flexible safety-critical software development should have, to be successfully deployed in an industrial context. The identified characteristics are based on the ISO25010 *quality in use* quality model, which focuses on how software is used in its context.

Our key findings are that it should be easy to learn how to use such a framework, it should be easy to use it and easy to adapt the framework to existing methods. It should also be easy to introduce into a development organization and be able to interface with existing tools, e.g. requirements or configuration management tools. To support agile development a framework should minimize verification and validation overhead so that the overhead is proportional to the size of the change and not to the number of changes. From an architectural viewpoint a framework should accommodate flexible extensions that go beyond what was initially intended. One way would be to use SoA and loosely coupled components, rather than tightly coupled. Finally to be able to build a trustworthy safety case using the framework, it should be analysable and predictable, as well as conform to applicable safety standards.

We have also presented the SimPal framework, which is a development framework for safety-critical software that was developed and evaluated on the basis of the identified quality characteristics. We show how it can be used to develop a

soft safety controller and describe in detail the premisses for such a controller and how it was evaluated against both a simulator and a technology demonstrator.

The SimPal framework is based on the PalCom middleware, which is a loosely coupled service-oriented middleware that decouples configuration and coordination of services, and the commercial modelling tool Simulink, which is used for designing service and system logic.

The design of the soft safety controller shows that the desired functionality can be realized using the SimPal framework, but run-time performance is a significant challenge that must be addressed.

The evaluation of the tool against the criteria in Section 3 suggests that the choice of Simulink might be a good choice as a modelling tool due to its strong user community and its ability to integrate with other tools and processes. However, from the perspective of the SimPal tool, the complexity and closed nature of Simulink might prevent its full realisation. The PalCom platform enables the SimPal framework to support flexible system evolution and reduces verification overhead due to the uniform deployment mechanism.

In the future we seek to develop a more robust implementation of the framework and try to evaluate the principles in an industrial setting. Moreover, we are considering adding new features such as *model-as-an-assembly* as well as improving the support for specifying commands with multiple parameters. Run-time performance is a key challenge for the future. A possible way forward would be to replace the Java version of PalCom with a C version, thus gaining the benefit of a leaner implementation as well as the possibility to use Simulink's code generation facilities. We are also considering investigating alternative modelling tools to Simulink that has better support for event driven execution.

7. ACKNOWLEDGEMENTS

The work in this paper was funded by the Swedish Foundation for Strategic Research under a grant to Lund University for ENGROSS- ENabling GROwing Software Systems.

8. REFERENCES

[1] P. Baufreton, J. P. Blanquart, J. L. Boulanger, H. Delseny, J. C. Derrien, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, P. Quéré, and B. Ricque. Multi-domain comparison of safety standards. In *Proceedings of the 5th International Conference on Embedded Real Time Software and Systems (ERTS2), Toulouse, France*, 2010.

[2] A. Collins, D. Joseph, and K. Bielaczyc. Design research: Theoretical and methodological issues. *Journal of the Learning Sciences*, 13(1):15–42, Jan. 2004.

[3] A. Coronato and G. De Pietro. A middleware architecture for safety critical ambient intelligence applications. In S. Balandin, R. Dunaytsev, and Y. Koucheryavy, editors, *Smart Spaces and Next Generation Wired/Wireless Networking*, volume 6294 of *Lecture Notes in Computer Science*, pages 26–37. Springer Berlin / Heidelberg, 2010.

[4] T. Dingsøyr, T. Dybå, and N. B. Moe. *Agile Software Development - Current Research and Future Directions*. Springer Berlin Heidelberg, 2010.

[5] EN 50126 Railway applications - The specification and demonstration of reliability, availability, maintainability and safety (RAMS), 2007.

[6] I. O. for Standardization/International Electrotechnical Commission et al. Iso/iec 9126. *Information Technology, Software Product Evaluation, Quality Characteristics and Guidelines for their Use*, 1991.

[7] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[8] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, Mar. 2004.

[9] International Electrotechnical Commission. IEC 61508, Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety Related Systems - Part 3: Software requirements. *65A/550/FDIS*, 2009.

[10] ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

[11] N. Leveson. *Engineering a safer world: systems thinking applied to safety*. MIT Press, Cambridge, Mass., 2011.

[12] D. Mishra and A. Mishra. Complex software project development: Agile methods adoption. *Journal of Software Maintenance and Evolution*, 23(8):549–564, 2011.

[13] J. P. Notander, P. Runeson, and M. Höst. A model-based framework for flexible safety-critical software development: A design study. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1137–1144, New York, NY, USA, 2013. ACM.

[14] J. Pedersen Notander, M. Höst, and P. Runeson. Challenges in flexible safety-critical software development – an industrial qualitative survey. In J. Heidrich, M. Oivo, A. Jedlitschka, and M. Baldassarre, editors, *Product-Focused Software Process Improvement*, volume 7983 of *Lecture Notes in Computer Science*, pages 283–297. Springer Berlin Heidelberg, 2013.

[15] D. Rodrigues, R. Melo Pires, J. C. Estrella, M. Vieira, M. Corrêa, J. B. Camargo Júnior, K. R. L. J. C. Branco, and O. T. Júnior. Application of SOA in safety-critical embedded systems. In G. Lee, D. Howard, and D. Ślęzak, editors, *Convergence and Hybrid Information Technology*, volume 206 of *Communications in Computer and Information Science*, pages 345–354. Springer Berlin Heidelberg, 2011.

[16] RTCA/DO178B Software Considerations in Airborne Systems and Equipment Certification, 1992.

[17] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, Aug. 2001.

[18] J. Srinivasan, R. Dobrin, and K. Lundqvist. 'State of the art' in using agile methods for embedded systems development. In *33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC*, volume 2, pages 522–527. IEEE Computer Society, 2009.

[19] I. Stürmer, D. Weinberg, and M. Conrad. Overview of existing safeguarding techniques for automatically generated code. In *Proceedings of the second international workshop on Software engineering for automotive systems*, SEAS '05, pages 1–6, New York, NY, USA, 2005. ACM.

[20] D. Svensson Fors, B. Magnusson, S. Gestegård Robertz, G. Hedin, and E. Nilsson-Nyman. Ad-hoc composition of pervasive services in the PalCom architecture. In *Proceedings of the 2009 ACM international conference on pervasive services (ICPS'09)*, pages 83–92. ACM, 2009.

[21] L.-J. Zhang, J. Zhang, and H. Cai. *Services Computing*. Springer Berlin Heidelberg, 2007.

ABOUT THE AUTHORS:

Jesper Pedersen Notander is a doctoral student in Computer Science at Lund University, Sweden. He received a M.Sc. degree from Lund University in 2008 and a Licentiate degree in Software Engineering from the same university in 2013. In addition, he has worked a couple of years as a software engineer in the aerospace industry. His research interests include safety-critical software development, service-oriented architectures, model-based development and runtime-verification.

Per Runeson is a professor of software engineering at Lund University, Sweden, and is the leader of its Software Engineering Research Group. His research interests include software development methods, in particular for verification and validation. He has co-authored hand-books for experiments and case studies in software engineering, serves on the editorial board of the Empirical Software Engineering Journal and several IEEE program committees.

Martin Höst is a Professor in Software Engineering at Lund University, Sweden. He received a M.Sc. degree from Lund University in 1992 and a Ph.D. degree in Software Engineering from the same university in 1999. His main research interests include software process improvement, software quality, risk analysis, and empirical software engineering. The research is mainly conducted through empirical methods such as case studies, controlled experiments, and surveys. He has published more than 60 articles in international journals and proceedings from conferences and workshops.

Genealogical Insights into the Facts and Fictions of Clone Removal

Minhaz F. Zibran
University of Saskatchewan
minhaz.zibran@usask.ca

Ripon K. Saha
The University of Texas at Austin
ripon@utexas.edu

Chanchal K. Roy
University of Saskatchewan
croy@cs.usask.ca

Kevin A. Schneider
University of Saskatchewan
kas@cs.usask.ca

ABSTRACT

Clone management has drawn immense interest from the research community in recent years. It is recognized that a deep understanding of how code clones change and are refactored is necessary for devising effective clone management tools and techniques. This paper presents an empirical study based on the clone genealogies from a significant number of releases of nine software systems, to characterize the patterns of clone change and removal in evolving software systems. With a blend of qualitative analysis, quantitative analysis and statistical tests of significance, we address a number of research questions. Our findings reveal insights into the removal of individual clone fragments and provide empirical evidence in support of conventional clone evolution wisdom. The results can be used to devise informed clone management tools and techniques.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*restructuring, and reverse engineering*

General Terms

Experementation, Management, Measurement

Keywords

clone removal, clone evolution, refactoring, reengineering

1. INTRODUCTION

Duplicate or similar code fragments are known as code clones. Previous studies report that software systems typically contain 9%-17% [40] of code clones, and the proportion may be as high as 50% [26, 27]. Code snippets that have identical source text except for comments and layout are called *Type-1* (exact) clones. Syntactically similar code snippets, where there may be variations in the names of the identifiers/variables are known as *Type-2* clones. Code fragments that exhibit *Type-2* clone similarity but also have other differences such as added, deleted or modified statements are *Type-3* clones.

Code cloning is a popular code reuse mechanism that is used to speedup the development process and facilitate independent evolution of similar program units. However, the use of code clones may be detrimental at times. For example, copy-pasting a code fragment already containing an unknown bug may cause fault propagation. Moreover, during the maintenance phase, a change in a clone fragment may necessitate consistent changes in all of its cloned fragments, and any inconsistencies may introduce vulnerabilities [41, 42, 43]. Thus, code clones may have a significant impact on the development and maintenance of software systems.

Despite ongoing research on the positive and negative effects of code clones [11, 12, 15, 21, 24, 25, 36], researchers and practitioners have come to an accord for the need of active and informed clone management [44, 45] including documentation and removal of clones through refactoring. However, code clones can often be desirable, and aggressive removal of clones through refactoring may not be a good idea [15, 41, 42, 43], given the risks and efforts involved in such activities. In this regard, a number of classification schemes [2, 13, 16, 33], metric based selection approaches [1, 4, 10], and an effort model [41, 42, 43] have been proposed to identify potential clones for refactoring. Still, for many systems, clone management and removal is yet to be a part of the daily maintenance activities [8]. Despite more than a decade of software clone research, clone management remains far from industrial adoption, and this area has gained more focus from the community in recent years [47].

A deep understanding of how individual clones change during their evolution, and which criteria cause their removal from the system, can help in devising effective strategies and tool support for clone management. A number of studies on near-miss clone evolution [30, 32, 40, 46] are found in the literature, which attempt to inform clone management [39, 44, 47]. These studies on clone evolution and programmers' psychology lead to some common beliefs and at times even contradictions about the traits of clone evolution. For example, the study of Kim et al. [15] suggests that many clones

are *volatile* (i.e., disappear shortly after they are created), while the study of Lozano and Wermelinger [18] suggests otherwise.

This paper focuses on the patterns of changes and removal of code clones during the evolution of software systems. In particular, we formulate the following eight research questions to capture different characteristics of clone change and removal. Some of the research questions correspond to common beliefs (or, contradictions) in the community; but we want to develop empirical evidence based on a systematic genealogy-based study on clone change and removal in evolving software systems.

RQ1: *Do the sizes of the groups of clones make any difference in clone removal in practice?* — Kim et al. [14] suspected that frequently copied code fragments (i.e., larger clone-groups) can be good candidates for clone refactoring.

RQ2: *Do the sizes of the individual clone fragments in terms of the number of lines impact clone removal in practice?* — Larger clone fragments can be attractive candidates for refactoring, as conjectured by Kim et al. [14].

RQ3: *For a group of clones, does the distribution of the clones in the file system hierarchy impact their removal in practice?* — Göde [8] reported that the developers were more interested in refactoring closely located clones.

RQ4: *Is there any relationship between any particular type of changes in the clones and their removal?* — This is still an open question, as far as we are concerned. If there exists any relationship between a particular type of changes and clone removal, the clone management tools can focus on supporting that category of changes.

RQ5: *How frequently do the clones experience changes before they are removed from the system?* — There is an ongoing debate on the stability of code clones [9, 17, 20].

RQ6: *Does the granularity (entire function bodies or syntactic blocks) of clones make any difference in their removal in practice?* — A recent study of Göde [8] reported many instances of removal of block clones by *extract method* refactoring.

RQ7: *Does the textual similarity in the source code of the clones have any effect in the removal of clones in practice?* — Very similar (e.g., *Type-1*) clones can be expected to be easier to refactor than very dissimilar (e.g., *Type-3*) clones.

RQ8: *During the evolution of the software systems, when does clone removal take place?* — This question addresses the aforementioned contradiction about the volatility of clones.

To address the research questions, we carry out a systematic study based on code clone *genealogy* [15, 31], which maps the individual clone fragments across subsequent releases over their evolution. We investigate the changes and removal of individual clones in 329 releases of nine diverse open-source software systems written in Java, C, and C#. Then we analyze them against a wide range of metrics and characterization criteria. In the light of a combination of qualitative analysis, quantitative analysis and statistical tests of significance, we derive the answers to the research questions.

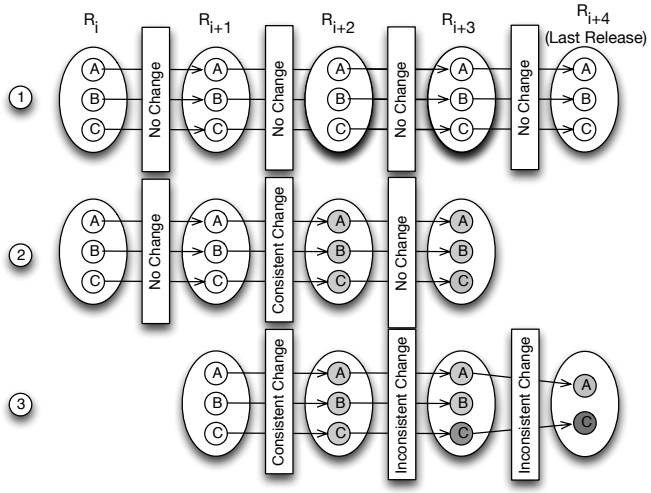


Figure 1. Different types of clone genealogies

We believe, such an empirical study on the characteristics of changes and removal of individual near-miss clone fragments is timely and addresses a gap in the literature. Our study is based on genealogies of near-miss clones including not only *Type-1* and *Type-2* clones, but also *Type-3* clones. This work is an extension to our previous work [46], which was the first genealogy-based study on the evolution, changes, and removal of near-miss code clones including *Type-3*. In this work, we significantly extend our previous study by including 101 more releases of three additional subject systems, an additional research question, and more in-depth analysis.

The rest of this paper is organized as follows. In Section 2, we introduce the terminology and metrics used in our study. In Section 3, we describe the setup and procedure of our empirical study. Section 4 presents the findings our study. In Section 5, we discuss the possible threats to the validity of our study. Section 6 accommodates related work, and Section 7 concludes the paper.

2. TERMINOLOGY AND METRICS

In this section, we introduce the terminology and metrics used in this paper to characterize the changes and removal of code clones. Some of the metrics and and criteria are adopted from earlier studies found in the literature [3, 7, 8, 15].

Clone Genealogy: A set of clone fragments that are clones of each other form a *clone-group*. A *clone genealogy* refers to a set of one or more lineage(s) originating from the same clone-group, whereas, a *clone lineage* is a sequence of clone-groups evolving over a series of releases of the software system. Figure 1 shows several examples.

Consistent and Inconsistent Change: If all clones in the clone-group experience the same set of changes during the transition between releases, then such changes are characterized as being a *consistent change*, otherwise the changes are regarded as being *inconsistent*.

Table 1. Software systems subject to our empirical study

Prog. Lang.	Subject System	No. of Releases	Releases Start	Releases End	Dates (mm/dd/yy) Start	Dates (mm/dd/yy) End	Duration (months)	Source Lines of Code (SLOC ranges)
Java	dnsjava	50	0.9.2	2.1.1	04/19/99	02/10/11	131	6,290 – 15,018
	JabRef	27	1.5	2.4.2	08/15/04	11/01/08	50	22,041 – 69,170
	ArgoUML	48	0.27.1	0.32.beta2	10/04/08	01/24/11	26	176,618 – 202,555
C	ZABBIX	31	1.0	1.8.4	03/23/04	06/01/11	86	9,252 – 62,845
	Conky	28	1.1	1.8.1	06/20/05	10/05/10	62	6,555 – 39,810
	Claws Mail	44	2.0.0	3.7.9	06/30/06	04/09/11	63	1,33,642 – 1,89,786
C#	CruiseControl	31	0.7.rc1	1.8.4	11/08/04	09/01/13	98	35,895 – 1,82,032
	iTextSharp	22	5.0.0	5.4.4	12/08/09	09/16/13	45	1,72,573 – 2,17,328
	ZedGraph	48	1.1	5.1.5	08/02/04	12/12/08	52	2,439 – 26,433

Consistently Changed Clone-Group: If the genealogy of a clone-group has any consistent change pattern(s) but does not have any inconsistent change patterns during evolution, it is classified as a *consistently changed* clone-group. The clone-group associated with the second genealogy in Figure 1 is an example of a consistently changed clone-group as there is a consistent change between releases R_{i+1} and R_{i+2}.

Inconsistently Changed Clone-Group: If the genealogy of a clone-group has any inconsistent change pattern(s) throughout the entire evolution period, it is characterized as an *inconsistently changed* clone-group. The clone-group associated with the third genealogy in Figure 1 is an inconsistently changed clone-group as there is an inconsistent change between releases R_{i+2} and R_{i+3}.

Static, Alive, Dead Clone-Group: Static clone-groups are those which propagate through subsequent releases having no textual change in the clones. A clone-group is called *dead* if it disappears before reaching the final release under consideration, otherwise the clone-group is considered *alive*. The clone-groups associated with the first, second and third genealogies in Figure 1 represents static, dead, and alive clone-groups respectively.

Textual Similarity: The textual similarity between two code snippets S_1 and S_2, denoted by $\int(S_1, S_2)$, is determined by calculating the identical lines with respect to their sizes, as defined by the following formula[1].

$$\int(S_1, S_2) = \frac{2 \times |\ell_1 \cap \ell_2|}{|\ell_1| + |\ell_2|} \quad (1)$$

where ℓ_1 and ℓ_2 are the ordered sets of pretty-printed lines in S_1 and S_2 respectively. $|\ell_1 \cap \ell_2|$ is the number of common ordered lines between ℓ_1 and ℓ_2, calculated using the longest common subsequence (LCS) algorithm. The textual similarity of a clone-group G, denoted as $\int(G)$ is the average of the textual similarities between all clone pairs in that group. Mathematically,

$$\int(G) = \frac{\sum_{S_i, S_j \in G} \int(S_i, S_j)}{\binom{|G|}{2}} \quad (2)$$

[1]In the area of Information Retrieval, this similarity measurement is known as the *Dice Coefficient*.

Entropy of Dispersion: We used an entropy measure to characterize the file level physical distribution of the clones in a clone-group. Such an entropy measurement, sometimes referred to as *Shannon entropy*, is commonly used in the area of Information Theory. In this work, the entropy of dispersion of the clones in clone-group G is calculated using Equation 3 as follows:

$$entropy(G) = \sum_{i \in \mathcal{F}_G} -p_i \log(p_i) \quad (3)$$

where, $\mathcal{F}_G$ denotes the set of distinct files hosting the clones in clone-group G, and p_i denotes the probability of the clones being located in file i.

For example, if all the clone fragments reside in the same file, the dispersion entropy will be 0.0. If the entropy is low, clones are densely located in only a few files. If the entropy is high, the clones are scattered across different files.

3. STUDY SETUP

To investigate the research questions outlined in Section 1, we study the clone genealogies across releases of nine diverse open-source software systems (Table 1) written in Java, C, and C#.

In the selection of the subject systems, we followed a number of criteria. First, we tried to include software systems that had reasonably large sizes and large number of releases. In computation of a system's size, we took into account only the source code lines (SLOC) written in the particular programming language that the software system is categorized in Table 1. We excluded comments, blank lines, and lines of code written in any other programming languages. Second, in our study, we tried to include subject systems from diverse application domains. Third, we preferred those open-source software systems, which were used in earlier studies [15, 30, 31, 40] reported in the literature.

3.1 Extraction of Genealogies

For the extraction of clone genealogies, we used an extended version of `gCad` [31] clone genealogy extractor that we developed. `gCad` can construct and classify genealogies of all three types (*Type-1, Type-2*, and *Type-3*) of clones that we are interested in. Details of how `gCad` operates and computes clone genealogies can be found elsewhere [31]. As per

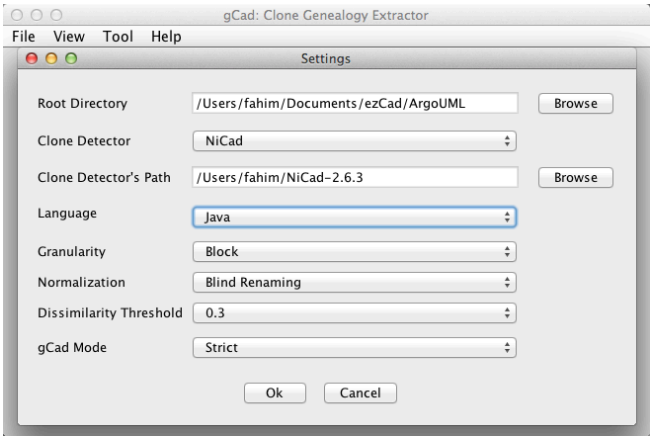


Figure 2. `gCad` settings for genealogy extraction

the need of this study, we significantly extended and customized the tool with a carefully designed graphical user interface (GUI), and a set of appropriate features to compute the necessary metrics. For the purpose of our study, we carefully chose a set of `gCad`'s configuration parameters as shown in Figure 2.

For the detection of code clones, we selected the `NiCad-2.6.3` [5, 6, 29] clone detector, which is a state-of-the-art clone detection tool reported to be effective in detecting both exact (*Type-1*) and near-miss (*Type-2* and *Type-3*) clones with high precision and recall [28, 29, 34]. `gCad` invokes `NiCad` to separately detect clones from every release of each of the subject systems. In invoking the clone detector, some of the `gCad` parameters are passed to `NiCad` to guide the process of detecting *Type-1, Type-2,* and *Type-3* clones at the chosen granularity (syntactic blocks for our study) and dissimilarity threshold (0.3 for our study).

The dissimilarity threshold (Figure 2) is a size-sensitive dissimilarity threshold that plays a vital role in guiding `NiCad` in the detection of *Type-3* clones. For our study, the dissimilarity threshold was set to 0.3, which signifies that `NiCad` detects two code fragments as clones if at least 70% of their pretty-printed text lines are the same (i.e., if at most 30% lines are different). The normalization option "blind-renaming" tells `NiCad` to ignore the differences in the names of identifiers/variables, and thus it is a significant parameter for the detection of *Type-2* clones.

From the clone detection results obtained from `NiCad`, for each of the subject systems, we separately constructed the clone genealogies using `gCad`. We operated `gCad` in 'strict' mode to construct and characterize clone genealogies. In 'strict' mode, `gCad` captures and takes into account all types of changes in the source code lines of clone-pairs, irrespective of whether those changes took place in their corresponding similar or dissimilar lines of code. Details of how `gCad` operates in different modes can be found elsewhere [31].

3.2 Investigation

We examined all the dead genealogies to see how the clones were removed. We also examined how the individual clone fragments changed during their evolution over a series of releases. Since, the inconsistent changes to clones are believed to be a common phenomena that produce vulnerabilities in a system [40, 41, 43], we characterized the clone changes as consistent versus inconsistent. In addition, we captured how frequently a clone-group changes during the evolution before its removal. For quantitative analysis, we computed the necessary metrics according to the categorization described in Section 2.

4. FINDINGS

The findings of our study are derived from qualitative and quantitative analyses of the changes and removal of individual clone fragments. We also apply the statistical *Mann-Whitney-Wilcoxon (MWW)* test [19] with $\alpha = 0.05$, to determine the statistical significance of the findings. Using the *Shapiro-Wilk* test [19] and *Q-Q* plot [19], we examined the distribution of the data, and found that some of the observations exhibited normal distributions while some others did not. Therefore, we chose to use the *MWW* non-parametric test, which does not assume the normal distribution of the data, and thus, is appropriate for data that exhibit or do not exhibit normal distribution.

4.1 Size of the Clone-Groups

To capture the relationship between the number of fragments in a clone-group and clone removal, we computed the average number of fragments in the removed clone-groups and that of the alive clone-groups for each of the subject systems (Table 2). As seen in Table 2, for all the subject systems, the average size of the alive clone-groups is higher than those of the removed clone-groups. During our manual investigation, we found that the developers refactored clone-groups that had only two or three clone fragments. Similarly, we found that in `JabRef`, there were 74 clone-groups having more than three fragments, and only four of them were refactored. This gives the impression that developers are more inclined to remove smaller clone-groups. To statistically verify this, we address the second research question *RQ1*, and formulate our null hypothesis as follows.

H_0^1**:** *The size of a clone group does NOT make a difference in clone removal in practice.*

A *MWW* test ($P = 0.35$) *fails to reject* (as, $P > \alpha$) the null hypothesis, which implies that the difference is not statistically significant. Hence, we answer the research question *RQ1* as follows.

Ans. to RQ1: *The size of the clone-groups (in terms of the number of member clone fragments) does not make a statistically significant difference in clone removal in practice.*

Although, in our study, the sizes of the removed clone-groups (in terms of the number of clone fragments) appears to be consistently lower than the alive clone-groups, this might have happened simply by chance in the software systems in our study. A larger study with many software systems may be required to further investigate the possibility of statistical significance of the pattern we found between the sizes of the clone-groups and their removal.

Table 2. Sizes of removed and alive clone-groups

Prog. Lang.	Subject System	Avg. Sizes of Clone-groups	
		Removed	Alive
Java	dnsjava	2.25	2.75
	JabRef	2.31	4.17
	ArgoUML	2.12	9.12
C	ZABBIX	2.31	4.53
	Conky	2.37	9.31
	Claws Mail	2.88	2.95
C#	CruiseControl	2.32	3.58
	iTextSharp	2.21	5.80
	ZedGraph	2.15	2.50

Table 3. Average sizes (SLOC) of clone fragments

Subject System	Removed		Alive	
	Average	SD	Average	SD
dnsjava	10.00	3.00	11.00	5.00
JabRef	17.00	15.00	13.00	9.00
ArgoUML	16.00	13.00	15.00	19.00
ZABBIX	26.00	25.00	21.00	21.00
Conky	20.00	23.00	15.00	7.00
ClawsMail	15.00	9.00	16.00	18.00
CruiseControl	8.85	4.50	9.46	5.61
iTextSharp	13.99	13.62	10.76	11.00
ZedGraph	15.70	15.45	12.34	12.67

SD = Standard Deviation

4.2 Size of the Clone Fragments

The sizes of the clone fragments can be expected to have a relationship with the refactoring effort, especially when the candidate clone-group includes near-miss (*Type-2* and *Type-3*) clones beyond *Type-1*.

To examine the relationship between clone removal and the SLOC per clone fragment in the clone-groups, we separately computed the average number of pretty-printed SLOC per fragment for the removed clones as well as for the alive clones. We also calculated the standard deviations for each of the measurements to capture the degree of variations. The results are presented in Table 3.

As can be observed from Table 3, there are subtle differences in the sizes of the clone fragments of both the removed and alive clone-groups. For six of the nine subject systems (`ZabRef`, `ArgoUML`, `ZABBIX`, `Conky`, `iTextSharp`, and `ZedGraph`), the average sizes of clone fragments of removed clone-groups appear to be significantly higher than those of the alive clone-groups. Hence, the anticipation of Kim et al. [14] saying – developers are more interested in getting rid of larger clones – appears to be true.

Addressing the research question *RQ2*, we now formulate our null hypothesis as follows.

H_0^2**:** *The size of the individual clones in terms of number of lines does NOT impact clone removal in practice.*

A MWW test ($P = 0.001$) over the series of sizes for the removed and alive clones *rejects* (as, $P < \alpha$) the null hypothesis. From the analysis described above, we answer research question *RQ2* as follows.

Ans. to RQ2: *The size of the individual clones in terms of number of lines does have a statistically significant impact on clone removal in practice, and larger clone fragments appear to be attractive for removal in practice.*

4.3 Entropy of Dispersion

In Table 4, we present the entropy of dispersion of both the removed and alive clones for all the subject systems. From the developer's perspective, refactoring/removal of co-located clones may require less effort than that needed for refactoring clones scattered over the code base. This can be expected to hold true due to several reasons. In the refactoring of scattered clones the developers might need to spend much time and effort to navigate to, understand the contexts, and make careful modifications at different locations of the code base.

In Table 4, we see that for each of the subject systems, the average entropy of dispersion for the removed clones is much lower than that for the alive clones. This indicates those clone-groups whose member clone fragments are closely located in the code base are relatively more attractive for refactoring/removal. To determine whether the initial observation *significantly* supports the expectation, we again conducted a MWW test with the null hypothesis as follows.

H_0^3**:** *For a group of clones, the distribution of individual clones in the file system hierarchy does NOT impact their removal in practice.*

The hypothesis addresses the research question *RQ3*. A MWW test ($P = 0.233$) between the entropy values for both the removed and alive clones (over all the systems) *fails to reject* (as, $P > \alpha$) the null hypothesis. This implies that there exists no relationship between the entropy of dispersion and clone removal in practice. Therefore, we derive the answer to research question *RQ3* as follows.

Ans. to RQ3: *For a group of clones, the distribution of individual clones in the file system hierarchy does not have a statistically significant impact on their removal in practice.*

As we delved deeper through manual investigation, we found a strange phenomenon in the relationship between entropy and the number of clone fragments that were removed. Most of the removed clone-groups had two fragments, if their entropy was greater than zero, i.e., they were not really located in the same file. For example, in `JabRef` and `ZABBIX`, developers refactored 37 and 43 clone-groups respectively, all of which had entropy higher than zero. Among them only two clone-groups in JabRef and 10 clone-groups in `ZABBIX` had three clone fragments, while the rest had only two fragments.

4.4 Change Patterns

Despite the realized advantages of code cloning, it is also true that code clones may have a significant impact on software development and maintenance in several ways. First,

Table 5. Removal of clone-groups classified by change patterns

Prog. Lang.	Subject System	Static Clone-Groups Total	Removed	[%]	Consistently Changed CG Total	Removed	[%]	Inconsistently Changed CG Total	Removed	[%]
Java	dnsjava	60	27	45.00	8	3	37.50	49	27	55.10
	JabRef	217	52	23.96	53	3	5.66	132	15	11.36
	ArgoUML	1435	109	7.60	39	4	10.26	440	19	4.31
C	ZABBIX	166	88	53.01	61	18	29.51	109	35	32.11
	Conky	121	44	36.36	19	7	36.84	37	16	43.24
	ClawsMail	445	58	13.03	172	7	4.07	304	7	2.30
C#	CruiseControl	528	187	35.41	119	35	29.41	388	133	34.27
	iTextSharp	1259	99	7.86	103	8	7.76	325	66	20.31
	ZedGraph	225	161	74.22	33	12	36.36	79	45	56.96

Table 4. Comparison of entropy of dispersion

Prog. Lang.	Subject System	Clones Removed	Alive
Java	dnsjava	0.71	0.90
	JabRef	0.53	0.98
	ArgoUML	0.82	1.30
C	ZABBIX	0.35	0.53
	Conky	0.18	0.24
	Claws Mail	0.30	0.70
C#	CruiseControl	0.18	0.23
	iTextSharp	0.22	0.38
	ZedGraph	0.17	0.10

the reuse by copy-pasting of any code segment that already contains unknown faults, results in propagation of those faults to all the copies. Second, when a change is made in a code fragment, consistent changes are often expected in all its clone fragments, while any inconsistencies may introduce new faults. Third, if a bug is found in a certain code fragment, there remains a possibility that similar bugs can be found in the clones of the fragment, and thus may necessitate consistent propagation of that bug-fix to all the clones.

Thus, whether the clones changed consistently, inconsistently, or remained static during the evolution of a software system, may have implications in clone management in future releases. Therefore, we categorized the clones based on whether they remained unchanged, or changed consistently or inconsistently, and what percentage of such clones were actually removed during the evolution of the system. For each of the systems, the total number of clones of each of these three categories and the percentage of them that were removed, are presented in Table 5.

As we can see in Table 5, for each of the subject system, the number of static clone-groups is the highest while the number of the consistently changed clone-groups is the lowest. To examine any trends in the existence of static, consistently changed, and inconsistently changed clone-groups in the systems, we again conducted *MWW* tests between each two of the three categories of changes (total number) occurred in the clone-groups over all the systems. The results

Table 6. *MWW* tests over of categories of changes

Change Types	No Change	Consistent Change	Inconsistent Change
No Change	-	$P = 0.003$	$P = 0.158$
Consistent Change	$P = 0.003$	-	$P = 0.042$
Inconsistent Change	$P = 0.158$	$P = 0.042$	-

Table 7. *MWW* tests over removal of clones

Clone Categories	Static	Consistently Changed	Inconsistently Changed
Static	-	$P = 0.31$	$P = 0.695$
Consistently Changed	$P = 0.31$	-	$P = 0.536$
Inconsistently Changed	$P = 0.695$	$P = 0.536$	-

of the *MWW* tests are presented in Table 6, which suggest significant difference in occurrence of the three categories of changes (as, $P < \alpha$), except that the difference in the number of inconsistent changes clones and no-changes are found to be statistically insignificant (as, $P > \alpha$).

With respect to clone removal, from Table 5, we see that for six of the nine systems (`JabRef, ZABBIX`, `ClawsMail`, `CruiseControl`, `iTextSharp`, and `ZedGraph`), the majority of the removed clones are static clone-groups. The removal of inconsistently changed clone-groups were found to occur most often in two of the systems (`dnsjava` and `Conky`), whereas, the removal of consistently changed clones dominated in `ArgoUML`.

A high-level perception from the results in the table may indicate that the static clone-groups can be more susceptible to removal. To verify such an observation, we carried out *MWW* tests between each pair of the three categories of clone *removal* over all the systems. The results of the *MWW* tests, as presented in Table 7, also suggest that there is no significant difference in the removal of static, consistently changed and inconsistently changed clone-groups (as, $P > \alpha$ in all cases).

Table 8. Frequency of changes before removal

Prog. Lang.	Subject System	Change Frequency 1	2	>2	Average
Java	dnsjava	16	9	5	1.80
	JabRef	11	4	3	1.72
	ArgoUML	17	4	2	1.48
C	ZABBIX	30	16	7	1.74
	Conky	10	8	5	1.95
	ClawsMail	9	2	5	1.57
C#	CruiseControl	103	45	20	0.75
	iTextSharp	62	11	1	0.50
	ZedGraph	40	12	5	0.39

These observations lead to the answer to the research question *RQ4* as follows.

Ans. to RQ4: *The majority of the clones do not experience any changes during their evolution. Those clones that experience changes, majority of those clone-groups undergo inconsistent changes. However, there is no statistically significant relationship between any particular type of changes in the clones, and their removal at a later release.*

4.5 Frequency of Changes

The frequency of changes to the clone-groups is an important criterion in clone management, since changing source code can be expensive, while making consistent changes to clones may involve significant effort and risks. Indeed, the modifications of a clone fragment needing effort, and the required effort can be multiplied by the size of the corresponding clone-group, to make consistent changes to all clone fragments in the clone-group. This is one of the areas where clone management tool support may contribute by facilitating clone merging, or consistent change propagation.

Thus, we examined how frequently the clone-groups underwent changes before their removal. In Table 8, we present the number of clone-groups that, before removal, underwent changes only once, twice, and more than twice. As seen in the table, most of the removed clones were changed only once. For the clone-groups that changed at least once, their average change frequency is less than two, over all the subject systems. From our manual verification, we found that very few clone-groups underwent changes more than twice before their removal. On the other hand, we also found many clone-groups remained alive although they experienced minor or significant changes. However, we confined our focus to the changes of the removed clone-groups to get a complete picture over the entire life-time of the clone-groups. Now, we derive the answer to the research question *RQ5* as follows.

Ans. to RQ5: *Most clones do not undergo frequent changes before their removal.*

4.6 Level of Granularity

The *extract method* refactoring pattern is perhaps the most highlighted technique for removing clones at the granularity of syntactic blocks. Thus, we may expect evidence of many instances of block clone removal. Alternatively, functions typically contain a somewhat complete implementation of certain features or program logic and so it may be easier to remove/refactor clones at the granularity of entire function bodies, rather than at the granularity of smaller syntactic blocks.

To determine whether there exist any relationships between clone removal and clone granularity, we examined both levels of granularities – function/method and syntactic block. Note that the body of a function also constitutes a block. Therefore, we distinguish *true* functions clones from the *true* block clones. A true function clone fragment spans the entire body of a function, whereas a true block clone must not constitute the entire body of a function.

Extended `gCad` is capable of differentiating true function clones from the true block clones. Any clone-group that is composed of only true function clones is categorized as a group of function clones, whereas, clone-groups consisting of only true block clones are categorized as groups of block clones. Separate genealogies are constructed for the clones at these two levels of granularity.

Over all releases of each of the subject systems, the total number and proportions of both the groups of function clones versus the block clones are presented in Table 9. The clone detection results for each of the systems identified clone-groups that contained both true function clones and true block clones. Therefore, it is not possible to categorize such a group as a group of only true function clones or only true block clones. This is why the total number of clone-groups reported in Table 9 is lower than that of Table 5. Addressing the research question *RQ*6, we now formulate our null hypothesis as follows.

H_0^6**:** *The granularity (entire function bodies or syntactic blocks) of clones does NOT make any difference in their removal in practice.*

A *MWW* test ($P = 0.93$) over the proportions of the removal of both true function and block clones *fails to reject* (as, $P > \alpha$) the null hypothesis.

Table 9 shows that developers remove both function and block clones as per their needs, as we do not see significant differences between the proportions of removal of function clones and block clones. For `ZABBIX` and `Conky`, the proportion of block clones removal is slightly higher. It seems that the clone removal rates for the two larger systems, `ArgoUML` and `Claws Mail` are far lower than the smaller systems. On the other hand, it appears that the developers of the relatively small systems `dnsjava`, `ZABBIX`, and `Conky` were more aware of the clones and were active in removing them through refactoring. From manual investigation, we found only one and two *Type-1* function clones in `dnsjava` and `Conky` respectively. Though as many as eight *Type-1* function clones were found in `ZABBIX`, seven of them were removed during the evolution of the system. Based on the above discussion, we now derive the answer to the research question *RQ6* as follows.

Ans. to RQ6: *In practice, the granularity (entire function bodies or syntactic blocks) of clones does not make any statistically significant difference in their removal.*

Table 9. Removal of clone-groups at the granularities of function and block

Prog. Lang.	Subject System	Function Clones Total	Removed	[%]	Block Clones Total	Removed	[%]
Java	dnsjava	69	37	53.62	25	15	60.00
	JabRef	204	41	20.09	110	21	19.09
	ArgoUML	1183	97	8.19	305	20	6.55
C	ZABBIX	201	78	38.80	134	62	46.26
	Conky	115	35	30.43	59	30	50.84
	Claws Mail	510	40	7.84	337	29	8.60
C#	CruiseControl	889	354	39.82	1032	355	34.40
	iTextSharp	999	186	18.62	1687	173	10.25
	ZedGraph	229	162	70.74	337	218	64.69

Table 10. Actual and normalized textual similarity of removed and alive clone-groups

Prog. Lang.	Subject System	Actual Textual Similarity: Removed Clones Average	SD	Alive Clones Average	SD	Normalized Textual Similarity: Removed Clones Average	SD	Alive Clones Average	SD
Java	dnsjava	0.60	0.20	0.67	0.18	0.80	0.12	0.81	0.10
	JabRef	0.76	0.18	0.68	0.18	0.85	0.13	0.82	0.11
	ArgoUML	0.76	0.20	0.66	0.17	0.85	0.14	0.80	0.14
C	ZABBIX	0.72	0.19	0.73	0.17	0.83	0.16	0.83	0.11
	Conky	0.76	0.16	0.69	0.15	0.88	0.09	0.84	0.09
	Claws Mail	0.73	0.20	0.65	0.22	0.87	0.12	0.82	0.15
C#	CruiseControl	0.67	0.19	0.68	0.18	0.83	0.09	0.84	0.09
	iTextSharp	0.72	0.22	0.66	0.19	0.86	0.11	0.84	0.12
	ZedGraph	0.80	0.22	0.70	0.22	0.91	0.14	0.87	0.14

SD = Standard Deviation

4.7 Textual Similarity

In Table 10, we present the average text similarities (actual and normalized) of the removed clones and the alive clones for each of the subject systems. Indeed, the degree of textual similarity among the clone fragments in a clone-group is important information as it corresponds to the differences between the clone fragments. Refactoring a clone-group with many variations can require more effort than refactoring a group of identical or very similar clones. Thus, the textual similarity for the clones in a clone-group can be expected to be proportional to the necessary efforts for refactoring them. Taking these into consideration, we address the research question *RQ1*, and formulate our null hypothesis as follows:

H_0^{7a}**:** *Clone removal by the developers is NOT dictated by the similarity of program text (without normalization) in the clone fragments.*

From the table, we can see that the average *actual* textual similarity of removed clones for four systems (`JabRef, ArgoUML, Conky`, and `Claws Mail`) is higher than that of alive clones, while the other two subject systems (`dnsjava` and `ZABBIX`) exhibit slightly the opposite trend. A *MWW test* ($P = 0.041$), on the data of actual textual similarity in the alive and removed clones, *rejects* (as, $P < \alpha$) the null hypothesis H_0^{7a}, which indicates statistically significant relationship between the textual similarity of clones and their removal.

Sometimes the actual textual similarity does not estimate the actual effort for refactoring. For example, in case of different identifiers names in different clone fragments, textual similarity of a clone-group may be very low although they are easily refactorable, especially when there is refactoring support from the IDEs (Integrated Development Environments). That is why we also investigated the normalized textual similarity by removing the identifier differences. If we look at the normalized text similarities of removed and alive clones, we again see that the average *normalized* textual similarity for the removed clones is slightly higher than the alive clones in those four systems. The trend is slightly the opposite for `dnsjava`, while for `ZABBIX`, both the removed and alive clones exhibit equal average normalized textual similarity. With respect to the normalized text similarities between the remove and alive clones, we formulate another hypothesis as follows:

H_0^{7b}**:** *Clone removal by the developers is NOT influenced by the similarity of* ***normalized*** *program text in the clone fragments.*

A *MWW* test ($P = 0.091$) *fails to reject* (as, $P > \alpha$) the null hypothesis H_0^{7b}, suggesting that there is *no statistically significant difference* in the normalized text similarities between the removed and alive clones. However, the P value in the case of normalized textual similarity is much lower than that of the actual textual similarity, and this hints that there might be some influence of the differences in the names of variables/identifiers over clone removal.

Combining the observations for the actual and normalized text similarities over the removed and alive clones, we can now derive the answer to the research question *RQ7* as follows.

Ans. to RQ7: *The textual similarity in the source code of the clones does have statistically significant effect in the removal of clones, and the differences in the names of the variable/identifiers play the major role in this regard.*

This finding indicates that *Type-1* clones are most attractive to the developer for refactoring, and the developers, in practice, are more inclined in refactoring *Type-2* clones than refactoring *Type-3*.

4.8 Age

The information about the age (in terms of the number of releases the clone-groups remain alive before removal) of clone genealogies can indicate how quickly the developers act to remove clones. In order to examine this phenomenon, for each of the systems, we computed the age of each clone-group that was removed in any of the subsequent releases.

In Figure 3, we present the proportion of clone-groups found to have been removed in a subsequent releases. As the figures (Figure 2(a), Figure 2(b), and Figure 2(c)) show, majority of the dead clones in five of the subject systems (`ArgoUML, JabRef, ZABBIX, Conky,` and `ZedGraph`) were removed within the initial five to ten releases. This observation is consistent with that reported by Kim et al. [15], suggesting that many of the clones are possibly *volatile*.

However, in all the systems, a good number clones remained alive over a long sequence of releases before their removal. For example, 17% of the refactored clone-groups in `ArgoUML` remained alive in 43 subsequent releases, while 35% of the clone-groups in `Claws Mail` propagated over 27 subsequent releases, before their removal. Similar trends were found in other systems as well. From the above discussion, we answer the research question *RQ8* as follows.

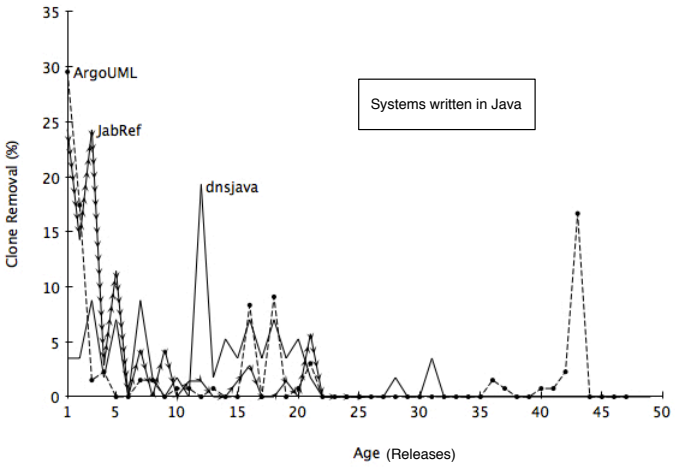
Ans. to RQ8: *During the evolution of the software systems, a few early releases experience significant clone removal. Nevertheless, some clones propagated over a relatively long sequence of releases before they were finally removed.*

This finding is also in keeping with the answer to the research question *RQ5* (Section 4.5), which indicates that most of the clones do not undergo frequent changes before their removal. We suspect that once a developer comes to know of a clone during its first change, this awareness might drive the removal of the clone at a later release. This indicates an area where informed clone management can play a significant role.

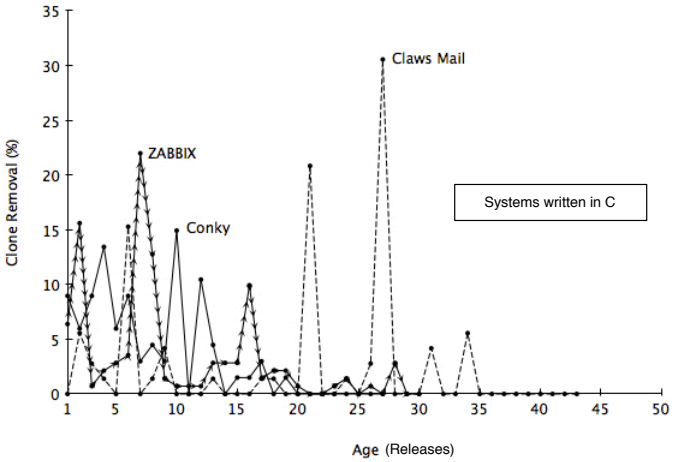
5. THREATS TO VALIDITY

In this section, we discuss possible threats to the validity of our study and how we mitigated their effects.

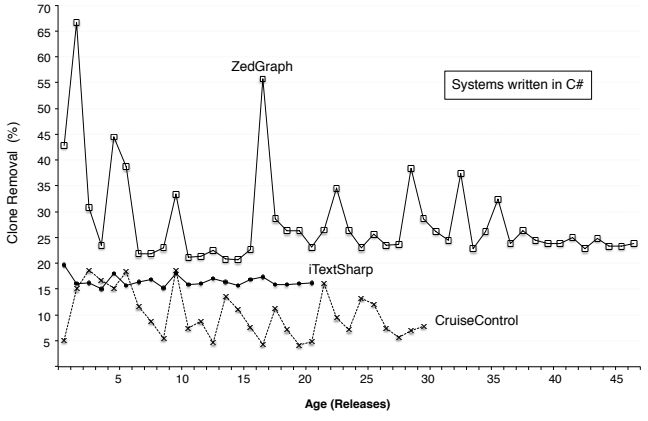
Construct Validity: Perhaps the best way to investigate change and evolution of clones is to study of the individual clone fragments in terms of genealogies across versions of the



(a)



(b)



(c)

Figure 3. Clone removal over sequences of releases

system. As versions one might choose programmers' commit transactions or weekly/monthly snapshots of the code base, or the stable releases of the system. A number of the earlier studies [8, 15] used the programmers' commit transactions or weekly/monthly snapshots of the code base, while many other studies [30, 31, 40] used software releases as the versions.

Programmers often create clones for experimental purposes, which they remove shortly after creation [15]. Thus, daily, weekly or monthly snapshots can be too frequent to capture stable changes in the code base. Indeed, commit transactions are more susceptible to this issue, in addition to their sensitivity to the developers' commit styles [40]. However, when a version of a software is officially released, the source code is expected to be in a stable form. Moreover, even a large number of weekly/monthly revisions may correspond to only a few stable releases, whereas series of releases typically span a longer period of development time. Therefore, for our study, we selected stable releases of the systems instead of commit transactions or snapshots at certain time intervals.

Internal Validity: The internal validity of our study is subject to the accuracy in clone detection and genealogy extraction. The `NiCad` clone detector used in our study, is reported to be effective in detecting both exact (*Type-1*) and near-miss (*Type-2* and *Type-3*) clones with high precision and recall [28, 29]. Moreover, our manual verification of random samples from the detected clones found no false positives. The genealogy extractor `gCad`, used in our study, is also reported to be accurate in the computation of near-miss clone genealogies [31]. Nevertheless, we carried out manual investigation to verify the correctness of the genealogies and to fix any inconsistencies. Indeed, the manual assessment can be subject to human errors. However, all the human participants of this study are faculty and graduate students carrying out research in the area of software clones, and thus we believe that they have affluent expertise to keep the probable human errors to the minimum.

External Validity: Our study is based on nine medium to fairly large open-source software systems, and thus one may question the *generalizability* of the findings. However, for each of the subject systems, we studied a significant number of releases, and we expect this to help minimize the threat to some extent. To further mitigate the threat, we carefully chose the subject systems from different application domains, and written in different programming languages.

Reliability: The methodology of this study including the procedure for data collection are documented in this paper. The subject systems are open-source, while the `NiCad-2.6.3` clone detector and the `gCad` genealogy extractor are also available online[2]. Therefore, it should be possible to replicate the study.

6. RELATED WORK

There has been considerable research in characterizing clone evolution and distinguishing clones of interest for removal by refactoring.

From a manual analysis of 800 function/method level clones over six different open-source Java systems, Balazinska et al. [2] proposed a taxonomy of function clones, based on the differences and similarities in the program elements. On the basis of the location of clones in the inheritance hierarchy, Koni-N'Sapu [16] proposed another clone taxonomy and a set of object-oriented refactoring patterns for refactoring each category of code clones. Later, Kapser and Godfrey [13] proposed a clone taxonomy based on the locations of clones in the file-system hierarchy and (dis)similarities in the code functionalities.

Schulze et al. [33] proposed a code clone classification scheme to support the decision of whether to use Object-Oriented Refactoring (OOR) or Aspect Oriented Refactoring (AOR) for clone removal. Other techniques, such as design patterns [1] and traits [22] were also attempted to identify and refactor clones of interest. Torres [37] applied a concept-lattice based data mining approach to derive four categories of concepts containing duplicated code and suggested refactoring patterns suitable for refactoring clones in each of the categories.

Higo et al. [10] proposed a software-metrics-based approach to identify potential clones that can be easier to refactor using the *extract method* and *pull-up method* refactoring patterns. Variations of such metrics-based approaches are realized in tools namely `Gemini` [38] and `ARIES` [10]. Choi et al. [4] carried out a developer-centric study to determine the effectiveness of different combinations of metrics in distinguishing clones of interest for refactoring.

None of the aforementioned work was based on code clone genealogies as ours, where we examined the evolution of individual clone fragments to characterize the patterns of change and removal of clones. Based on the experience from an ethnographic study on copy and paste programming practices, Kim et al. [14] reported that "larger or frequently copied code fragments are good candidates for refactoring." The findings of our study also to support their conjecture.

Based on a case study on two open source Java systems, Tairas and Gray [35] reported that in some cases clone refactorings were partially performed on only parts of the clones (i.e., sub-clones). However, their focus was only on the occurrences of refactorings composed of the *extract method* refactoring pattern. The objective of our work was to investigate and characterize removal and refactoring of clones not only through the *extract method* refactoring patterns, but also by all other possible means.

Göde [8] conducted a case study over four systems, and investigated the extent clones were removed from the systems. He found many instances of deliberate clone removal, and the majority of those removals were performed by the *extract method* refactoring pattern. He further reported that the developers refactored mostly the closely located clones, which is also consistent with our findings.

The study of Göde was based on only three metrics, and he concluded that more complex metrics such as change frequency of clones should be examined to better understand the phenomenon. In our study, based on clone genealogies over 329 releases of nine software systems and using a wide

[2]http://usask.ca/∼minhaz.zibran/pages/projects.html

range of characterization criteria, we captured a broader picture of clone removal and changes in open-source software systems.

7. CONCLUSION

This paper presents a genealogy-based empirical study on the evolution of individual clone fragments to characterize the changes and removal of exact (*Type-1*) and near-miss (*Type-2* and *Type3*) code clones. We examined a total of 329 releases from nine open-source software systems written in Java, C, and C#.

In the study, we addressed eight research questions, and derived answers to those with a combination of qualitative and quantitative analyses as well as statistical tests of significance. The findings of our study shed light on the conventional wisdom about clone evolution, in particular, derive useful insights into the patterns of changes and removals of code clones in practice.

From the study, we found that the sizes of the clone-groups (in terms of the number of member clone fragments), or the granularity (i.e., functions or blocks) of clones, or their dispersion in the file-system hierarchy do not have any significant effect on clone removal in practice. In terms of change patterns, we did not find any relationships between clone removal and any particular type of changes (i.e., consistent or inconsistent).

However, highly similar or larger clone fragments appear to be attractive for removal. A few early releases of the software systems experienced significantly more changes and removal of clones than the later releases. Inconsistent changes are found to have dominated over consistent changes of code clones. We also found that the majority of clones that were removed, did not experience frequent changes before removal, and surprisingly, most of those clones underwent changes only once, before they were removed from their respective systems.

During manual investigation, we discovered many instances of clones, which could be attractive for refactoring, but those were left alone, perhaps due to the lack of proper tool support. We believe that the practical findings from this study make significant contributions to the existing wisdom about clone evolution, refactoring, and removal, which in turn, will be useful for devising effective tools and techniques for informed clone management.

Acknowledgement: This work is supported in part by the Walter C. Sumner Memorial Foundation.

8. REFERENCES

[1] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis. Advanced clone-analysis to support object-oriented system refactoring. In *Proc. of the 7th Working Conference on Reverse Engineering*, pp. 98–107, 2000.

[2] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Measuring clone based reengineering opportunities. In *Proc. of the 6th International Symposium on Software Metrics*, pp. 292–303, 1999.

[3] D. Cai and M. Kim. An empirical study of long-lived code clones. In *Proc. of the International Conference on Fundamental Approaches to Software Engineering*, pp. 432–446, 2011.

[4] E. Choi, N. Yoshida, T. Ishio, K. Inoue, and T. Sano. Extracting code clones for refactoring using combinations of clone metrics. In *Proc. of the 5th International Workshop on Software Clones*, pp. 7–13, 2011.

[5] J. Cordy and C. Roy. The NiCad clone detector. In *Proc. of the Tool Demo Track of the 19th International Conference on Program Comprehension*, pp. 219–220, 2011.

[6] J. Cordy and C. Roy. Tuning research tools for scalability and performance: the NiCad experience. In *Science of Computer Programming*, 79(1):158–171, 2014.

[7] N. Göde and R. Koschke. Frequency and risks of changes to clones. In *Proc. of the 33rd International Conference on Software Engineering*, pp. 311–320, 2011.

[8] N. Göde. Clone removal: fact or fiction? In *Proc. of the 4th International Workshop on Software Clones*, pp. 33–40, 2010.

[9] N. Göde and J. Harder. Clone stability. In *Proc. of the 15th European Conference on Software Maintenance and Reengineering*, pp. 65–74, 2011.

[10] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. Aries: Refactoring support environment based on code clone analysis. In *Proc. of the 8th IASTED International Conference on Software Engineering and Applications*, pp. 222–229, 2004.

[11] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proc. of the 31st International Conference of Software Engineering*, pp. 485–495, 2009.

[12] C. Kapser and M. Godfrey, "Cloning considered harmful" considered harmful: patterns of cloning in software entities, In *Journal of Empirical Software Engineering*, 13(6):645–692, 2004.

[13] C. Kapser and M. Godfrey. Aiding comprehension of cloning through categorization. In *Proc. of the 7th International Workshop on Principles of Software Evolution*, pp. 85–94, 2004.

[14] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in OOPL. In *Proc. of the International Symposium on Empirical Software Engineering*, pp. 83–92, 2004.

[15] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. In *Proc. of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 187–196, 2005.

[16] G. Koni-N'Sapu. A scenario based approach for refactoring duplicated code in OO systems. Diploma thesis, University of Bern, 116 pp., 2001.

[17] J. Krinke. Is cloned code more stable than non-cloned code? In *Proc. of the 8th International Conference on Source Code Analysis and Manipulation*, pp. 57–66, 2008.

[18] A. Lozano and M. Wermelinger. Tracking clones' imprint. In *Proc. of the 4th International Workshop on Software Clones*, pp. 65–72, 2010.

[19] D. Anderson, D. Sweeney, and T. Williams. *Statistics for Business and Economics*. Thomson Higher Education,10th Edition, 2009.

[20] M. Mondal, C. Roy, M. Rahman, R. Saha, J. Krinke, and K. Schneider. Comparative stability of cloned and non-cloned code: An empirical study. In *Proc. of the 27th ACM Symposium On Applied Computing (SE Track)*, pp., 1227–1234, 2012.

[21] M. Mondal, C. Roy, and K. Schneider. An empirical study on clone stability. In *Applied Computing Review*, 12(3):20–36, 2013.

[22] E. Murphy-Hill, P. Quitslund, and A. Black. Removing duplication from java.io: a case study using traits. In *Proc. of the ACM SIGPLAN conference on Systems, Programming, Languages and Applications*, pp. 282–291, 2005.

[23] H. Nguyen, T. Nguyen, N. Pham, J. Al-Kofahi, and T. Nguyen. Clone management for evolving software. In *IEEE Transaction on Software Engineering*, 38(5):1008–1026, 2011.

[24] J. Pate, R. Tairas, and N. Kraft. Clone evolution: a systematic review. In *Journal of Software: Evolution and Process* , 25(3): 261–283, 2013.

[25] F. Rahman, C. Bird, and P. Devanbu. Clones: what is that smell? In *Proc. of the 7th Working Conference on Mining Software Repository*, pp. 72–81, 2010.

[26] M. Rieger, S. Ducasse, and M. Lanza. Insights into system-wide code duplication. In *Proc. of the 11th Working Conference on Reverse Engineering*, pp. 100–109, 2004.

[27] C. Roy and J. Cordy. A survey on software clone detection research, *Technical Report 2007-541*, School of Computing, Queen's University, 115 pp., 2007.

[28] C. Roy and J. Cordy. A mutation/ injection-based automatic framework for evaluating code clone detection tools. In *Proc. of Mutation*, pp. 157–166, 2009.

[29] C. Roy and J. Cordy. NiCad: Accurate Detection of Near-Miss Intentional clones using flexible pretty-printing and code Normalization. In *Proc. of the 16th International Conference on Program Comprehension*, pp. 172–181, 2008.

[30] R. Saha, M. Asaduzzaman, M. Zibran, C. Roy, and K. Schneider. Evaluating code clone genealogies at release level: an empirical study. In *Proc. of the 10th International Conference on Source Code Analysis and Manipulation*, pp. 87–96, 2010.

[31] R. Saha, C. Roy, and K. Schneider. An automatic framework for extracting and classifying near-miss clone genealogies. In *Proc. of the International Conference on Software Maintenace*, pp. 293–302, 2011.

[32] R. Saha, C. Roy, K. Schneider, and D. Perry. Understanding the evolution of Type-3 clones: an exploratory study. In *Proc. of the 10th Working Conference on Mining Software Repositories*, pp. 139–148, 2013.

[33] S. Schulze, M. Kuhlemann, and M. Rosenmüller. Towards a refactoring guideline using code clone classification. In *Proc. of the 1st Workshop on Refactoring Tools*, pp. 6:1–6:4, 2008.

[34] J. Svajlenko, C. Roy, and J. Cordy. A mutation analysis based benchmarking framework for clone detectors. In *Proc. of the Tool Demonstration Track of the 7th International Workshop on Software Clones*, pp. 8–9, 2013.

[35] R. Tairas and J. Gray. Sub-clones: Considering the part rather than the whole. In *Proc. of the 9th International Conference on Software Engineering Research and Practice*, pp. 284–290, 2010.

[36] S. Thummalapenta, L. Cerulo, L. Aversano, and M. D. Penta. An empirical study on the maintenance of source code clones. In *Journal of Empirical Software Engineering*, 15(1):1–34, 2009.

[37] R. Torres. Source code mining for code duplication refactorings with formal concept analysis. M.Sc. thesis, Vrije Universiteit Brussel, 53 pp., 2004.

[38] Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue. Gemini: Maintenance support environment based on code clone analysis. In *Proc. of the 9th International Symposium on Software Metrics*, pp. 67–76, 2002.

[39] R. Venkatasubramanyam, S. Gupta, and H. Singh. Prioritizing Code Clone detection results for clone management. In *Proc. of the 7th International Workshop on Software Clones*, pp. 30–36, 2013.

[40] M. Zibran, R. Saha, M. Asaduzzaman, and C. Roy. Analyzing and forecasting near-miss clones in evolving software: an empirical study. In *Proc. of the 16th International Conference on Engineering of Complex Computer System*, pp. 295–304, 2011.

[41] M. Zibran and C. Roy. A constraint programming approach to conflict-aware optimal scheduling of prioritized code clone refactoring. In *Proc. of the 11th International Conference on Source Code Analysis and Manipulation*, pp. 105–114, 2011.

[42] M. Zibran and C. Roy. Conflict-aware Optimal Scheduling of Code Clone Refactoring: A Constraint Programming Approach. In *Proc. of the 19th International Conference on Program Comprehension*, pp. 266–269, 2011.

[43] M. Zibran and C. Roy. Conflict-aware optimal scheduling of code clone refactoring. In *IET Software*, 7(3):167–186, 2013.

[44] M. Zibran and C. Roy. Towards flexible code clone detection, management, and refactoring in IDE. In *Proc. of 5th the International Workshop on Software Clones*, pp. 75–76, 2011.

[45] M. Zibran and C. Roy. IDE-based real-time focused search for near-miss clones. In *Proc. of the 27th ACM Symposium On Applied Computing (SE Track)*, pp. 1235–1242, 2012.

[46] M. Zibran, R. Saha, C. Roy, and K. Schneider. Evaluating the conventional wisdom in clone removal: A genealogy-based empirical study. In *Proc. of the 28th ACM Symposium On Applied Computing (SE Track)*, pp. 1123–1130, 2013.

[47] M. Zibran and C. Roy. The road to software clone management: A survey. *Tech. Report 2012-03*, Department of Computer Science, University of Saskatchewan, Canada, pp. 1–62, 2012.

ABOUT THE AUTHORS:

Minhaz F. Zibran is a Ph.D. candidate at the Department of Computer Science, University of Saskatchewan, Canada. His research interests include various aspects of software engineering with particular focus on the detection, analysis, and management of code clones in evolving software systems. Minhaz has co-authored scholarly articles published in ACM and IEEE sponsored international conferences and reputed journals. Throughout his career, Minhaz also earned both teaching and industry experience. He has been actively involved in organizing international conferences (e.g., ICPC'2011, SCAM'2012, ICPC'2012, WCRE'2012, ICSM'2013) in his area of research. His scholarly excellence enabled him earning many scholarships and awards including the postgraduate scholarship from the Natural Science and Engineering Research Council (NSERC) of Canada.

Ripon K. Saha is a Ph.D. student in the Department of Electrical and Computer Engineering at The University of Texas at Austin. He received his B.Sc. degree in Computer Science and Engineering from Khulna University, Bangladesh and M.Sc. degree in computer science from University of Saskatchewan, Canada. His research interests include program analysis, mining software repositories, and empirical software engineering.

Chanchal Roy is an assistant professor of Software Engineering/Computer Science at the University of Saskatchewan, Canada. While he has been working on a broad range of topics in Computer Science, his chief research interest is Software Engineering. In particular, he is interested in software maintenance and evolution, including clone detection, analysis and management, reverse engineering, empirical software engineering, and mining software repositories. He served or has been serving in the program committee of major software engineering conferences (e.g., ICSM, WCRE, MSR, ICPC and SCAM). He served as the Finance Chair for ICPC'11, Tool Co-chairs for ICSM'12 and WCRE'12, Tool Chair for SCAM'12, Poster Co-chair for ICPC'12, Program Co-chair for IWSC'12, and Finance Chair for ICSM'13. He has been working as the General Chair for ICPC'14.

Dr. Kevin Schneider is a Professor of Computer Science, Special Advisor ICT Research and Director of the Software Research Lab at the University of Saskatchewan. Dr. Schneider has held appointments as Computer Science Department Head, Vice-Dean Science, and Acting Chief Information Officer and Associate Vice-President Information and Communications Technology. Before joining the University in 2001, Dr. Schneider was CEO and President of Legasys Corp., a software research and development company specializing in design recovery and automated software engineering. His research investigates models, notations and techniques that are designed to assist software project teams develop and evolve large, interactive and usable systems. Dr. Schneider is a member of the ACM and IEEE CS, an elected member of the International Federation for Information Processing working group 2.7/13.4 on user interface engineering and past Prairie representative for the Canadian Association of Computer Science.

A Boosted SVM based Ensemble Classifier for Sentiment Analysis of Online Reviews

Anuj Sharma
Chandragupt Institute of Management
Hindi Bhavan,
Patna – 800001, India
f09anujs@iimidr.ac.in

Shubhamoy Dey
Indian Institute of Management
Prabandh Shikhar, Rau,
Indore – 453331, India
shubhamoy@iimidr.ac.in

ABSTRACT
In recent years, several approaches have been proposed for sentiment based classification of online text. Out of the different contemporary approaches, supervised machine learning techniques like Naive Bayes (NB) and Support Vector Machines (SVM) are found to be very effective, as reported in literature. However, some studies have reported that the conditional independence assumption of NB makes feature selection a crucial problem. Moreover, SVM also suffers from other issues like selection of kernel functions, skewed vector spaces and heterogeneity in the training examples. In this paper, we propose a hybrid method by integrating "weak" support vector machine classifiers using boosting techniques. The proposed model exploits classification performance of Boosting while using SVM as the base classifier, applied for sentiment based classification of online reviews. The results on movies and hotel review corpora of 2000 reviews have shown that the proposed approach has succeeded in improving the performance of SVM. The resultant ensemble classifier has performed better than the single base SVM classifier, and the results confirm that ensemble SVM with boosting, significantly outperforms single SVM in terms of accuracy. [1]

Categories and Subject Descriptors
I.5.2 [**Pattern Recognition**]: Design Methodology—*classifier design and evaluation, feature evaluation and selection*; I.5.1 [**Pattern Recognition**]*:* Models—*SVM*; I.2.7 [**Natural Language Processing**] – *Text analysis*

General Terms
Performance, Design, Experimentation, Theory

Keywords
SVM, Sentiment Analysis, Classification, Sentiment Lexicon, Text Mining

1. INTRODUCTION
Sentiment analysis and opinion mining of online user generated text content has already proved to be a promising research domain with growing popularity of Web 2.0 social media [12, 15]. Consumers and users have enthusiastically raised their voices and expressed their sentiments in the form of textual posts on social media for virtually anything they care about. Web 2.0 based mediums like message forums, blogs and review sites have emerged as good sources of expressed opinion and sentiments on a large scale [23].

The large scale opinionated text available on the Internet and Web 2.0 social media has created ample research opportunities for business and academia. Different research works have associated opinion expressed in online reviews to product sales [42], opinion in online discussion to prediction of best travel destinations [41], and public sentiments in political debates to results of general elections [35], the list is limitless. In case of online reviews, researchers have concluded that web based opinion are a good proxy for word-of-mouth [5, 23].

With the rapid growth of the social media, more and more users post reviews for all types of products and services and place them on online forums. It is becoming a common practice for a potential consumer to learn how much others like or dislike a product before arriving at a purchase decision. By processing the reviews, product manufacturers and marketing professionals can keep track of customer opinions of their products, with the aim of improving the user satisfaction. However, as the number of reviews available for any given product grows, it becomes a more time consuming task for buyers to understand and evaluate what the prevailing opinion trend about the product is. So, from the point of view of users, to read these millions of reviews from different Web 2.0 based sources is nearly impossible. Moreover, it is also an expensive process for the companies to track the opinion about their products or services in the large volume of online reviews.

The large volume of opinionated data poses severe data processing and sentiment extraction related challenges. Different contemporary solutions based on different machine learning, dictionary, statistical, and semantic based approaches have been proposed for sentiment analysis of online textual data [6, 23, 37]. Existing machine learning approaches have given promising results [16, 30]. Therefore, it is important to enhance these existing techniques that can extract knowledge from voluminous subjective or opinionated texts.

Though the other approaches like dictionary and semantic orientation based approaches perform quicker than machine learning based approaches and have no requirement of pre-annotated text, studies have reported poor results, in terms of accuracy, in real-life applications. The maintenance of sentiment

dictionaries for different domains is also a critically important task related to lexicon based methods [20]. The statistical approach aims to exploit statistics based on co-occurrence of words to derive sentiment polarity of features, which in turn, is found to be a very inefficient way to analyze sentiment of text in terms of time complexity [31, 38].

This paper proposes a hybrid sentiment classification model based on Boosted SVM. The proposed model exploits classification performance of two techniques (boosting and SVM) applied for the task of sentiment based classification of online reviews. SVM has been found to be one of best machine learning classifiers in recent studies [31, 34]. Literature reports that the learning phase of SVM is very time consuming for large textual data and some approximate algorithms can reduce the learning time of SVM [30]. Although, the approximate algorithms can reduce the computation time, they significantly degrade the classification accuracy.

To address the above issues, this study proposes to use Boosting techniques for the SVM ensemble. The Boosted SVM ensemble is experimented for sentiment classification of online reviews. Our working hypothesis is that Boosting can improve classification accuracy compared to a single SVM.

Boosting algorithms like Adaboost can find a good final hypothesis combining multiple appropriate hypotheses produced by weak (or base) classifier(s) [10]. Roughly, we denote a weak classifier as a classifier that returns a hypothesis that outperforms just simple random guessing. Using a classifier like SVM as the base weak learner, this study has shown that a weakened version of SVM can be useful as a base weak classifier suitable for boosting. Previous studies have also supported the view that the size of the weights assigned by Adaboost to the weak SVMs, serve as an indication of which data points are likely to become support vectors in the final model, and hence can be useful for constructing a strong classifier.

The results of our experiments on movie reviews and a hotel reviews corpora of 2000 reviews have shown that the proposed approach has succeeded in improving performance of SVM when used as a weak learner for sentiment based classification. The results show that SVM ensemble with bagging or boosting outperforms a single SVM in terms of accuracy of sentiment classification. The rest of this paper is organized as follows: Section 2 presents related work on sentiment analysis. The proposed methodology is described in Section 3. Experimental results are given in Section 4. Finally, Section 5 concludes the paper.

2. RELATED WORK

The last few years have witnessed significant developments in research related to opinion mining and sentiment analysis. Majority of the works reported are related to lexicon and supervised machine learning based techniques. The details of work apart from machine learning approaches are out of scope of this study and can be found in recent surveys [23, 37].

Table 1 summarizes some of the studies related to sentiment based classification of text documents using SVM and other machine learning classifiers. The basic machine learning approach for sentiment based classification of opinionated text aims at finding patterns from pre-annotated text documents during learning and uses n-fold cross validation for assessment of the accuracy of the built model. The build model is then used to classify the test and/or validation corpuses.

Table 1. Studies Related to SVM for Sentiment Analysis

Author	Model	Data Source and Dataset	Accuracy (%)
Pang et al. (2002) [25]	NB, ME, SVM	Movie reviews (IMDb)- 700 (+) and 700 (-) reviews	77–82.9
Dave et al. (2003) [8]	NB, ME, SVM	Product reviews (Amazon)	88.9
Pang & Lee (2004) [22]	NB, SVM	Movie reviews (IMDb)- 1000 (+) and 1000 (-) reviews	86.4-87.2
Gamon (2004) [11]	SVM	Customer reviews (feedback)	69.5- 77.5
Pang & Lee (2005) [24]	SVM, SVR, Regression, Metric Labeling	Movie reviews (IMDb)- 5006 reviews	54.6- 66.3
Chen et al. (2006) [4]	Decision Trees C4.5, SVM, NB	Books Reviews (Amazon)- 3,168 reviews	84.59
Boiy et al. (2007) [2]	SVM, Multinomial NB, ME	Movie reviews (IMDb)- 1000 (+) and 1000 (-) reviews, Car reviews- 550 (+) and 222 (-) reviews	90.25
Annett & Kondrak (2008) [1]	SVM, NB, Decision Tree	Movie reviews (IMDb)- 1000 (+) and 1000 (-) reviews	Greater than 75%
Shimada & Endo (2008) [33]	SVR, SVM OVA, ME	Product Reviews (video games)	NA
Dasgupta & Ng (2009) [7]	SVM and Clustering based	Movie reviews (IMDb) and product reviews (Amazon)- 1000 (+) and 1000 (-) reviews	69.5- 93.7
Ye et al. (2009) [41]	NB, SVM and Character based N-gram model	Travel blogs from travel.yahoo.com- 591 (-) and 600 (+) reviews	80.71-85.14
Paltoglou & Thelwall (2010) [21]	SVM	Movie Reviews (IMDb)- 1000 (+) and 1000 (-) reviews, Multi-Domain Sentiment Dataset (MDSD)- 8000 reviews	MR- 96.90, MDSD- 96.40
Xia et. al. (2011) [40]	NB, ME, SVM, meta-classifier combination	Movie Reviews (IMDB), product reviews (Amazon)- 1000 (+) and 1000 (-) reviews	88.65
Sharma & Dey (2012a) [31]	SVM, Feature Selection Techniques	Movie Reviews (IMDb)	90.1
Sharma & Dey (2012b) [30]	SVM & 6 other classifiers	Movie Reviews (IMDb)	90.9
Sharma & Dey (2012c) [32]	ANN	Movie and Hotel Reviews	95

Machine learning techniques became popular for sentiment analysis following the seminal studies by Pang et al. [25] and Pang and Lee [22, 24]. Different supervised machine learning classifiers like Naive Bayes, maximum entropy and support vector machines were compared and SVM showed the best performance on the movie reviews corpus with average precision of around

80% [25]. Many other works experimented with different machine learning classifiers with different types of features like single words, character N-grams, and word N-grams, like bigrams and trigrams. Feature selection methods like information gain, gain ratio, CHI statistics, Relief-F etc. were also used with different machine learning techniques for sentiment based classification [31].

From review of literature it can be concluded that support vector machine has performed better in term of accuracy than other machine learning based methods of sentiment analysis. The original SVM algorithm for sentiment based classification is implemented using the approximation based optimization algorithms which perform structural risk minimization in order to reduce the computation complexity of time and space.

Literature also suggests that a single SVM may not be able to learn the exact parameters for globally optimum results. The support vectors obtained from the learning data (pre-annotated as per sentiment orientation) are not sufficient to classify all unknown test/validation instances of opinionated documents completely. So, it cannot be guaranteed that SVM always provides the globally optimum sentiment based classification performance over all test cases (including all n-folds, in case of n-fold cross-validation). To overcome this drawback, this study proposes a novel boosted SVM approach for sentiment analysis.

3. METHODOLOGY

First, the opinionated text documents were collected and then, pre-processed. The vector space model (VSM) was utilized in order to generate the bag of words representation for each document. The text documents were pre-processed with basic natural language processing techniques like word tokenization, stop word removal and stemming. In this study, we have used Snowball stemmer algorithm for the English language [27]. Some useful sentiment expressing terms such as “ok” and “not” were consciously preserved as these words carry sentiment bearing subjective expression by online users.

The residual tokens were arranged as per their frequencies or occurrences in whole documents set. Following that, we used information gain feature selection method to pick out top *n*-ranked discriminating attributes to train the classifiers, as information gain has been reported to be one of the most effective feature selection methods for sentiment based classification of opinionated text [30]. The number of selected features (*n*) was varied from very small to very large (50-10000) for our experiments.

3.1 Support Vector Machine (SVM)

Support vector machines (SVMs) are highly effective for traditional text categorization, and can outperform Naive Bayes [22]. The SVM has been known to show a good generalization performance and can easily learn the exact parameters for the global optimum. SVM seeks a hyper-plane represented by vectors that splits the positive and negative training vectors of documents with maximum margin. The problem of finding this hyperplane can be translated into a constrained optimization problem. SVM algorithm classifies opinionated text vectors by separating it into positive and negative classes with a hyperplane, which can be further extended to non-linear decision boundaries using various kernels [14].

The structural risk minimization principle is utilized from the computational learning theory. The idea of structural risk minimization is to find a hypothesis *h* for which we can guarantee the lowest true error. In the presence of noise and outliers, the idea of using a soft margin was suggested in literature [39]. SVM seeks a decision surface to separate the training data points into two classes and makes decisions based on the support vectors that are selected as the only effective elements in the training set. Effectively, training a SVM classifier requires the solution of a very large quadratic programming optimization problem.

This study has used a variant of SVM for fast training using Sequential Minimal Optimization (SMO) [26]. SMO breaks this large quadratic programming problem into a series of smallest possible quadratic problems avoiding complex and resource-hungry matrix computations. The best case space complexity for SMO is a linear function of the training set size, which enables SMO to handle very large training sets. SMO's computation time is dominated by SVM evaluation; hence SMO is fastest for linear SVMs and sparse data sets.

The optimization of SVM (dual form) is to minimize the SVM Lagrangian equation expressed as:

$$\vec{\alpha}^{*} = \arg\min \left\{ -\sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left\langle \vec{x_i}, \vec{x_j} \right\rangle \right\} \quad (1)$$

$$\text{Subject to:} \quad \sum_{i=1}^{n} \alpha_i y_i = 0; \; 0 \le \alpha_i \le C \quad (2)$$

Where x_i’s are the input pattern and α_i’s are Lagrange multipliers. Depending on the choice of kernel functions, different classifiers including linear and nonlinear classifiers can be constructed.

The classification result of a practically implemented SVM is often far from the theoretically expected level because their implementations are based on the approximated algorithms due to the high complexity of time and space. A complete description of the SVM classifier and the kernel functions used for sentiment analysis is outside the scope of this study and can be found in [25, 28].

3.2 Boosting Algorithms

Boosting is a generic approach for improving performance of any given classifier [3]. It can effectively ensemble a number of weak classifiers into a strong pattern learner which can achieve an arbitrarily high accuracy given sufficient training data, although each weak classifier might do just a little better than random guessing. Some popular methods for selecting the representative training samples from a collection of datasets are bagging, boosting, randomization, stacking and dagging [10].

This study focuses on bagging and boosting. These algorithms are used as voting methods, which formulate a single strong classifier from a linear combination of a number of weak classifiers. In the domain of topic based text classification, some boosting algorithms have shown significantly better results. Bootstrapping techniques are used for assessing statistical accuracy of some estimate. Boosting approach can be assumed as a sample based statistical method which consists of drawing random samples with/without replacement from the set of data points.

3.2.1 Bagging

This research proposes to use bagging technique to construct the SVM ensemble. In bagging, several SVMs are trained independently via a bootstrap method and then they are aggregated to formulate a strong classifier via an appropriate combination technique. Let us consider a single training set TR = $\{(x_i, y_i)| i = 1, 2,\ldots, n\}$. To construct the SVM ensemble with K independent SVMs, K training sample sets will be needed. Also, in order to obtain higher improvement of the aggregation result, the training sample sets should be different from each other, as far as possible. For performing the aggregation, bootstrap techniques have been suggested [3].

The main idea of this algorithm is to fit a regression model which will develop a prediction f_x at input x_i. Bootstrapping builds K replica training data sets $\{\text{TR-bootstrap}_k \mid k = 1, 2,\ldots, K\}$ by randomly re-sampling with replacement, from the given training data set TR repeatedly. Each example x_i in the given training set TR may appear more than once or not at all in any particular replica training data set. Each replica training set will be used to train a particular SVM. By averaging classification prediction over the collection of bootstraps, bagging may significantly reduce variance and increase accuracy. The bagging algorithm is summarized as follows:

Input: training set TR = $\{(x_i, y_i)| i = 1, 2,\ldots, n\}$ where x_i is the document vector with class y_i.
Output: An ensemble classifier for the training set TR-bootstrap$_k$.
For K iterations-

1. Draw with replacement K≤ n sample from the training set TR, obtaining the jth sample TR^j to train the jth SVM.
2. Train the jth SVM for each K replicated training data sets TR-bootstrap$_k$.
3. Build the final ensemble classifier as a majority based vote of SVM$_j$ (j=1, …, K). The voting is simplifies as where θ is the voting parameter:

$$\text{SVM}_{\text{final}} = \text{sign}\left(\theta_j \sum_{j=i}^{k} SVM_j\right) \tag{3}$$

3.2.2 Boosting

Boosting is a sampling technique which is similar to bootstrapping and bagging approaches. The difference among them lies in the way the training set is prepared by taking samples from the population [10]. The bootstrapping and bagging techniques performs sampling with replacement but boosting performs sampling without replacement. Like bagging, each SVM is also trained using a different training set. But, the selection scheme of training samples in boosting method is different from the bagging method [9].

Initially, we have a training set TR = $\{(x_i, y_i)| i = 1, 2,\ldots, n\}$, consisting of all the *n* samples and each sample in the TR is assigned to have the same value of weights $w(x_i)=1/n$. For training the Kth SVM classifier, we build a set of training samples TR-boost$_k$ = $\{(x_i, y_i)| i = 1, 2,\ldots, k\}$, that is obtained by selecting K<n samples from the whole data set TR according to the weight values $w_{k-1}(x_i)$ at the (k-1)th iteration. This training sample is used for training the Kth SVM classifier. Then, we evaluate the classification performance of the Kth trained SVM classifier using the whole training set TR. This sampling procedure is repeated until K training sample sets have been built for the Kth SVM classifier. The boosting algorithm is summarized as follows:

Input: training set TR = $\{(x_i, y_i)| i = 1, 2,\ldots, n\}$, K=number of samples in the training set.
Output: An ensemble classifier for the training set TR-boost$_k$.
For K iterations do repeat step 1 to 3-

1. Draw without replacement K≤ n sample from the training set TR, obtaining the jth sample TR^j to train the jth SVM.
2. Train the jth SVM for each K replicated training data sets TR-bootstrap$_k$.
3. Select some of the misclassified samples by jth SVM with another drawn sample without replacement from TR to train jth+1 SVM.
4. Build the final ensemble classifier as a majority based vote of SVM$_j$ (j=1, …, K). The voting is as per Eq. 3.

3.2.3 Adaptive Boosting (AdaBoost)

One of the most popular Boosting methods, AdaBoost [10] creates a collection of weak learners by computing a set of weights over training samples in each iteration instead of performing random sampling. AdaBoost adjusts these weights based on the combined classifier that is formulated from combination of weak classifiers. The weights of the misclassified samples by the current classifier are increased while those of the correctly classified are be decreased.

Thus, AdaBoost establishes a collection of weak base classifiers by maintaining a set of weights over training samples and then, adjusting them adaptively after each Boosting iteration. Hence, the weights of the misclassified samples by current weak classifier will be increased while the weights of the correctly classified samples will be decreased. To implement the weight updates in Adaboost, several approaches have been proposed. The performance of AdaBoost can be attributed to its ability to enlarge the margin, which could enhance AdaBoost's generalization capability.

The effectiveness of the AdaBoost (like boosting algorithms) in improving generalization performance is based on the notion of a margin that can be interpreted as a measure of confidence in the prediction. Many studies that use decision trees or neural networks as weak learners for AdaBoost have reported good generalization performance. The AdaBoost algorithm is summarized as follows:

Input: training set TR = $\{(x_i, y_i)| i = 1, 2,\ldots, n\}$, K=number of samples
Output: An ensemble classifier for the training set TR-AdaBoost$_k$.
Initialize the weights $w_i=1/n$ for i= 1, 2,…, n.

For K iterations-

1. Train the base SVM$_j$ on the weighted training data using the weights w_i
2. Calculate error term for the wrong classification:

$$\varepsilon_t = \sum_{i=1}^{n} D_t(i) I_{y_i \neq SVM(x_i)} \tag{4}$$

3. compute the optimal weights update step to calculate weight contribution as:

$$\alpha_t = 0.5 \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \quad (5)$$

4. Update the sample distribution as:

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i SVM(x_i)}}{Z_t} \quad (6)$$

where Z_t is a normalization factor to ensure: $\sum_{i=1}^{n} D_{t+1}(i) = 1$

5. The final composite classifier is given by-

$$\text{SVM}_{\text{final}} = \text{sign}\left(\theta_j \sum_{j=i}^{k} SVM_j\right) \quad (7)$$

3.3 Boosting for Constructing SVM Ensemble

The key idea is integrating SVM with Boosting to enhance the generalization ability of a strong classifier by automatically selecting the best features for the base learners at each boosting step. While SVMs explicitly maximize the minimum margin, boosting tends to do the same indirectly through minimizing a cost function related to margin.

The need of creating an ensemble SVM is justified since the practical SVM has been implemented using approximated algorithms in order to reduce the computation complexity of time and space. For this reason, a single SVM may not learn exact parameters to reach at the global optimum solution. Sometimes, the support vectors obtained from the learning is not sufficient to classify all unknown test examples completely. So, we cannot guarantee that a single SVM always provides the global optimal classification performance over all test examples.

The idea of Boosted SVM has been proposed in this study for sentiment based classification for online text. This study assumes that the boosting technique can be used to train each individual SVM and thus several SVMs can be combined. This study proposes to use the SVM ensemble based on bagging and boosting techniques. In bootstrapping (bagging), each individual SVM is trained over the randomly chosen training samples via a bootstrap technique. In boosting, the training samples for each individual SVM is chosen according to updating probability distribution (related to error) for samples. Then, the independently trained SVMs can be aggregated in various ways such as the majority voting, least-squares estimation (LSE)-based weighting, and double-layer hierarchical combining.

3.3.1 Aggregation Strategies for SVM Ensemble

Literature has reported several linear and nonlinear combination methods to aggregate several independently trained SVMs. Majority voting and the LSE-based weighting are two popular linear combination methods proposed by different studies [18]. In a nonlinear method, another upper-layer SVM is applied to combine several lower-layer SVMs so as to create a nonlinear combination of several SVMs. This method is known as the double-layer hierarchical combining. This study has used majority voting as an aggregation strategy as it is the simplest method for combining several SVMs.

Let Q_p (p=1, 2, …, P) be a decision function of the p^{th} SVM in the SVM ensemble and C_j (j=1,2, …, C) denote a label of the j^{th} class. Then, let N_j= #{p| Q_p (x)=C_j}, which represents the number of SVMs whose decisions are known to the j^{th} class. Then, the final decision of the SVM ensemble Q_{mv}(x) for a given test vector x due to the majority voting is determined by:

$$Q_{mv}(x) = \arg \max_j N_j \quad (8)$$

The other methods for combining several SVMs can be LSE-based weighting and double-layer hierarchical combining. The LSE-based weighting treats several SVMs in the SVM ensemble with different weights [17]. Often, the weights of several SVMs are determined in proportional to their accuracies of classifications. As reported in some studies, LSE-based weighting cannot be applied to large size training set due to space and time complexity issues [19]. The hierarchical combining aggregation method uses another SVM to aggregate the outputs of several SVMs in the SVM ensemble. This combination consists of a double layer of SVMs hierarchically where the classification results of several SVMs in the lower layer feed into a super SVM in the upper layer.

The final decision of the SVM ensemble in double-layer hierarchical combining is determined by the decision function of the super SVM in the upper layer that takes outcome of all other SVMs decision function. However, using this double layer approach can severely affect the time and memory requirements of the approach. So, this study has used majority voting as an aggregation strategy.

4. EXPERIMENTAL EVALUATION

4.1 Corpora and Sentiment Lexicons

Two datasets belonging to totally different domains (movies and hotels) were selected to evaluate the proposed approaches in this paper. The movie reviews dataset was originally prepared by Pang and Lee (2004). This dataset contains movie reviews collected from IMDb.com (Internet Movie Database) [25]. This dataset is one of the best known databases to be found in the sentiment analysis literature and is also known as polarity dataset v2.0 or Cornell Movie Review Dataset. There are a total of 1,000 positive and 1,000 negative reviews in the dataset to represent two classes of sentiments.

The hotel reviews dataset containing 2000 hotel reviews was prepared by extracting hotel reviews of 54 different hotels in popular travel destinations in India from the websites: TripAdvisor (tripadvisor.com) and Yatra (yatra.com). The reviews were annotated by independent subject experts and were classified in terms of the overall sentiment orientations as being positive or negative. This study has adopted the same approach as in [22, 25] that annotated hotel reviews with more than 3 star ratings as being positive and hotel reviews with less than 3 star ratings as being negative. We have discarded reviews with 3 stars (neutral) to restrict our work for binary sentiment analysis.

4.2 Performance Evaluation

In this work we have used overall accuracy (OA) and *F*1 Score as performance evaluation matrices. The confusion matrix shown in Table 2 is for used for evaluating the performance of classifiers.

Table 2. The confusion matrix

	Predicted positives	Predicted negatives
Actual positive examples	Total True Positive examples (TP)	Total False Negative examples (FN)
Actual negative examples	Total False Positive examples (FP)	Total True Negative examples (TN)

The performance of sentiment classification is evaluated by the Overall Accuracy, which is given by:

$$\text{Overall Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \quad (9)$$

Another popular evaluation matrix is F1 score which is derived from the combination of Precision and Recall. *F*1 score, Precision, and Recall are defined by:

$$\text{Precision} = \frac{TP}{TP + FN} \quad (10)$$

$$\text{Recall} = \frac{TP}{TP + FP} \quad (11)$$

$$F1\ \text{Score} = \frac{2\ ?\ \text{Precision}\ ?\ \text{Recall}}{\text{Precision} + \text{Recall}} \quad (12)$$

4.3 Experimental Results

This section describes the experiments and results of the proposed approach. Java based implementations on Microsoft Windows platform were used to implement the boosting, bagging and SVM classifiers. The LibSVM wrapper classes were used to build the SVM classifier. LibSVM provides reports for statistics on the confusion matrix, precision, recall, ROC score, etc. This study has used radial basis kernel for SVM as this function has been reported to have good performance for sentiment based classification in prior research studies [31].

For bootstrapping, we randomly re-sampled the existing training dataset with replacement to make new training data sets. For boosting, we iteratively re-sampled data samples with replacement according to the updated probability distribution from the training data set. We trained each SVM independently over the replicated training data set and aggregated several trained SVMs via majority voting. For avoiding the over-fitting problem, each classifier has been validated using 10 folds in which we performed 10 independent runs of experiments and then took average performance of all the runs to report in the results.

However, some studies have criticized the use of the cross-validation tests for comparing classifiers. The research findings point out that, while the mean classification accuracy across the cross validation folds is an unbiased estimate of the true accuracy, the variance may be optimistically biased for many classifiers. This may lead to spurious ranking of the classifiers as per accuracy and performance. But, in this study we have used 10 folds cross validation for the sake of standard comparison.

4.3.1 Results on Movie Reviews Dataset

Instead of including an overwhelming number of tables, we have decided to visualize the results in a nontraditional way using comparative charts. Figure 1, 2, and 3 shows accuracy, precision and recall of sentiment based classification on movie reviews dataset. The proposed approach gave consistent accuracy with information gain based feature selection and gave up to 93% accuracy for 800-2000 selected features. SVM with boosting has shown best accuracy of 93% for movie reviews domain. The best accuracy of 92% was achieved by SVM with AdaBoost, and classical single SVM was the worst performer in all four SVM implementations. Similar results were obtained for recall and precision for the movie reviews dataset (Figures 2 and 3).

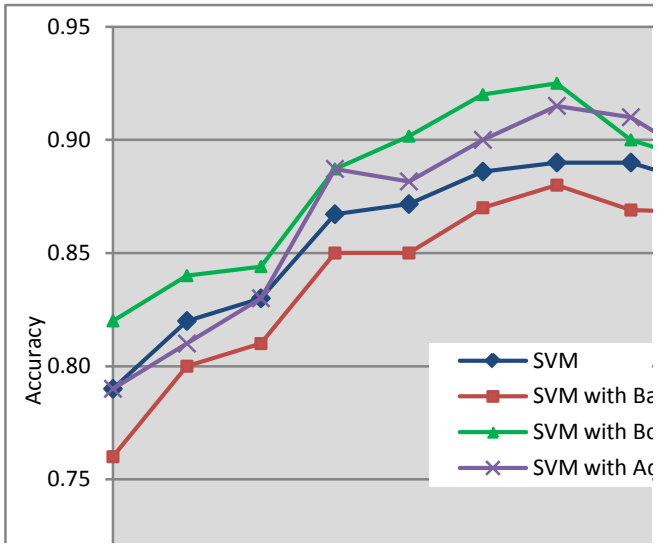


Figure 1. Accuracy of Boosted SVM on Movies Reviews

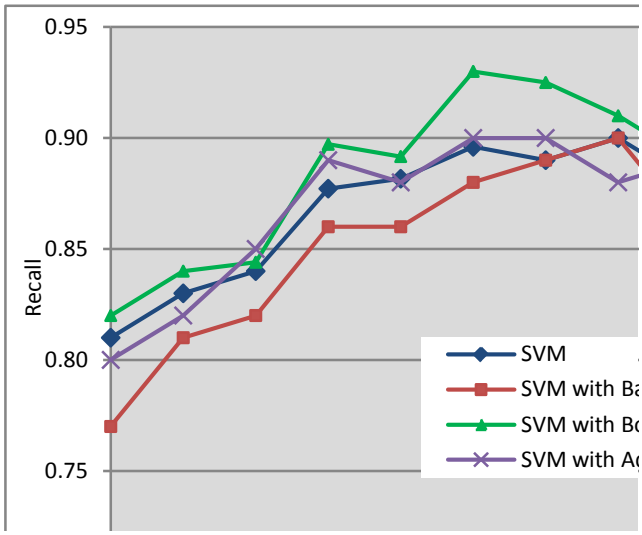


Figure 2. Recall of Boosted SVM on Movies Reviews

4.3.2 Results on Hotel Reviews Dataset

Similar results were obtained for the hotel reviews dataset. SVM with boosting and SVM with AdaBoost outperformed the other two methods. This study has selected top-n features (n=50 to 10000) ranked as per information gain in the hotel reviews corpus.

Boosting has clearly improved the classification performance of SVM. The results in recall and precision were also consistent with the results on accuracy (Figures 4 to 6). It is obvious that the proposed approach exceeds the base classifier at the accuracy of sentiment classification for both the datasets. We have selected sentiment bearing text from two different domains in order to elucidate the performance of the proposed approach for cross domain sentiment analysis.

The results on movie and hotel reviews are supporting the view that up to 2000 features selected by some well established feature selection method like information gain can give good results. Selecting features more than this range does not lead to improvement in classification results. Figures 1-6 confirm that number of features ranging from 500 to 2000 can be a reasonable trade-off among learning time, computational effort and acceptable level of ensemble classifier accuracy.

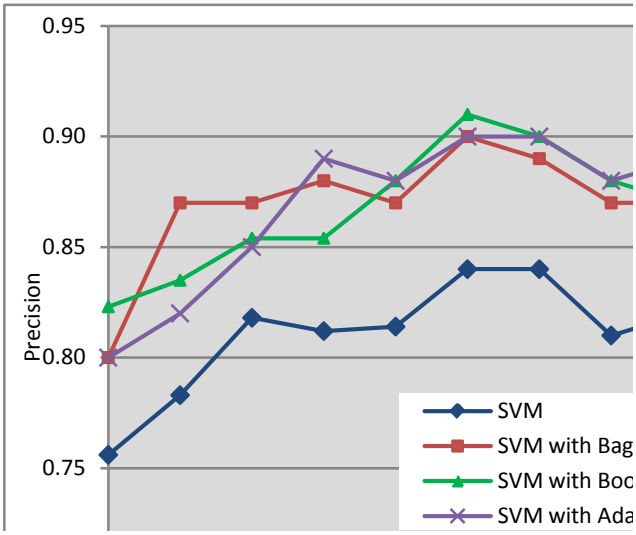


Figure 3. Precision of Boosted SVM on Movies Reviews

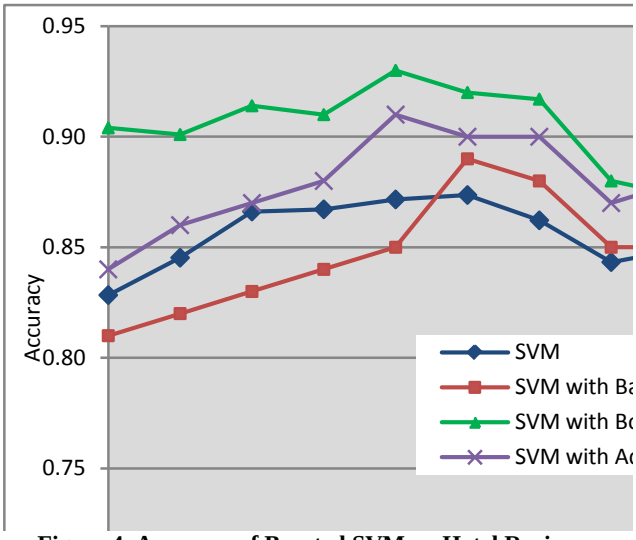


Figure 4. Accuracy of Boosted SVM on Hotel Reviews

Figure 5 and 6 shows comparison of precision and recall for all four SVM implementations. The *F*1-score values can be derived from the precision and recall values as per Equation 11. The results clearly indicate that boosting is suitable for sentiment based classification with performance better than other machine learning techniques used in previous studies [30]. Literature also confirms that boosting can be used with *any* base classifier capable of handling weighted training instances and SVM is a good example of such classifiers [9, 13, 36]. Although the base classifiers are not restricted to belong to a certain classifier family, this study supports the view that virtually all classifiers can work with boosting algorithms.

5. DISCUSSION

This paper proposes a sentiment classification approach using boosted SVM. Information gain was used to extract sentiment representing features, and different boosting and bagging approaches were used to train the SVMs. The results on movie reviews and hotel reviews corpuses have shown that the proposed approach has succeeded in improving the performance of SVM for sentiment based classification of online reviews from two different domains. The boosting of SVM and input feature selection are important tasks for better sentiment classification. The results on movies and hotel review corpora of 2000 reviews have shown that our approach has succeeded in improving performance of SVM when used as a weak learner for sentiment based classification.

SVMs usually suffer from biased decision boundaries (in case of the hyperplane), and studies have shown that their prediction performance drops dramatically when the data is highly skewed. Moreover, the practical SVM has been implemented based on the approximation algorithm to improve the time and space complexity of the algorithm. So, the obtained classification performance is far from the theoretically expected level of it.

The proposed approach combines integrated sampling techniques with an ensemble of SVMs to improve the sentiment prediction performance. The integrated sampling techniques can be further experimented with to deal with issues like over-sampling and under-sampling that play a dominant role in text classification and sentiment analysis tasks.

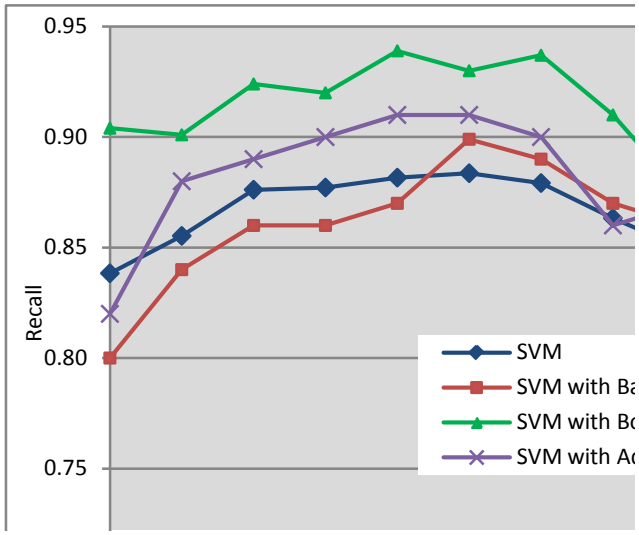


Figure 5. Recall of Boosted SVM on Hotel Reviews

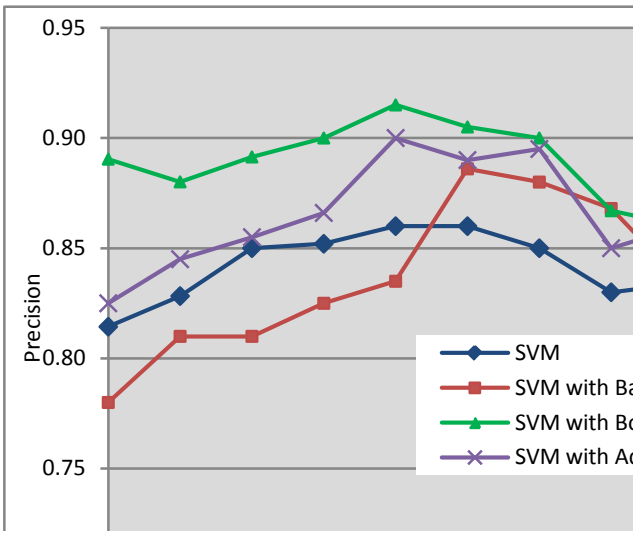


Figure 6. Precision of Boosted SVM on Hotel Reviews

However, this study has restricted its scope for binary sentiment analysis but we hypothesize that the advantage of using the SVM ensemble over a single SVM can be achieved equally in the case of multi-class classification (as in positive, negative and neutral sentiment). Since the SVM is originally a binary classifier, many SVMs should be combined to achieve multi-class classification. The SVM based ensemble classifier for the multi-class sentiment based classification can be a good extension of the approach described in this study. It will be interesting to explore the improvement of classification performance in the multi-class sentiment based classification by taking the SVM ensemble where each SVM classifier is designed for one of the multi-classes (positive, negative and neutral) sentiment.

6. CONCLUSION

Ensemble learning approaches for improving performance of weak classifiers is a significantly promising trend in current research on machine learning. The ensemble method finds a highly accurate classifier by combining many moderately accurate component classifiers. Bagging and boosting are powerful and popular approaches for creating ensemble classifiers. Specifically, the results show that SVM ensemble with bagging or boosting significantly outperforms a single SVM in terms of accuracy of sentiment based classification. Although, we have used a RBF kernel to create the boosted SVM, the same Framework can be tested with different kernel functions in future works.

The theoretical justifications and empirical findings of this study demonstrate that our method is effective. We find that the proposed boosted SVM classifiers are robust for sentiment based classification in two ways. Firstly, they improve the performance of the base SVM classifier when trained with training set; and secondly, they are sufficiently simple to be widely applicable. Future work may involve adopting this bagging / boosting based approach to classify sentiment in other types of text like blogs, Twitter posts, Facebook, and so on, and experiments with other types of base classifiers like neural networks.

In recent years, several other approaches have been proposed to develop ensemble classifiers like artificial immune-system algorithms [43], random subspace [13] based different feature subsets and Rotation Forest [29] with transforming the feature subsets by principal component analysis (PCA). Although there are many different proposed methods to create ensemble classifiers, all of them are constructed to combine more than two different classifiers based on the diversity and individual error of classifiers. Evaluating these recently proposed ensemble classifier approaches for sentiment classification would also be an interesting research area to explore in future.

7. ACKNOWLEDGMENTS

The authors wish to thank ACM RACS 2013 reviewers for the helpful comments to improve this work. Further, the authors wish to thank Mr. Biresh Kumar, for providing assistance in preparing the hotel reviews dataset and Prof. Lillian Lee, for providing Movie Reviews dataset for research purposes.

8. REFERENCES

[1] Annett, M., and Kondrak, G. A comparison of sentiment analysis techniques: Polarizing movie blogs. *Advances in Artificial Intelligence*, (2008), 5032, 25–35.

[2] Boiy, E., Hens, P., Deschacht, K. and Moens, M. F. Automatic sentiment analysis of on-line text. In *Proceedings of the 11th International Conference on Electronic Publishing* (Vienna, Austria) 2007.

[3] Breiman, L. Bagging predictors. *Machine Learning*, vol. 24, issue 2, August 1996, 123–140.

[4] Chen, C., Ibekwe-SanJuan, F., SanJuan, E., and Weaver, C. Visual analysis of conflicting opinions. In *IEEE Symposium on Visual Analytics Science and Technology*, 2006, 59–66.

[5] Chevalier, J. A., and Mayzlin, D. The effect of word of mouth on sales: Online book reviews. *Journal of Marketing Research*, 43, 3 (2006), 345−354.

[6] Cui, H., Mittal, V., and Datar, M. Comparative experiments on sentiment classification for online product reviews. In *Proceedings of AAAI* (Boston, Massachusetts, July 16-20, 2006). 2006, 1265–1270.

[7] Dasgupta, S., and Ng, V. Topic-wise, sentiment-wise, or otherwise? identifying the hidden dimension for unsupervised text classification. In *Proceedings of the EMNLP'09* (Morristown, NJ, USA), ACL, 2009, 580–589.

[8] Dave, K., Lawrence, S., and Pennock, D. M. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international WWW conference* (Budapest, Hungary, May 20–24, 2003). 2003, 519–528.

[9] Dietterich, T. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, Mach. Learn. 26 (1998) 1–22.

[10] Freund, Y., and Schapire, R. Experiments with a new boosting algorithm. In *Proc. 13th International Conf. on Machine Learning* (*ICML*), Bari, Italy, 1996, 148–156.

[11] Gamon, M. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In *Proceedings of the 20th international conference on Computational Linguistics* (Geneva, Switzerland). ACL, 2004.

[12] Godbole, N., Srinivasaiah, M., and Skiena, S. Large-scale sentiment analysis for news and blogs.In *Proceedings of the International Conference on Weblogs and Social Media (ICWSM'07)* 2007.

[13] Ho, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1998), 20 (8), 832–844.
[14] Joachims, T. Making large-scale support vector machine learning practical, *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, 1999.
[15] Kamps, J., Marx, M., Mokken, R. J., and De Rijke, M. Using wordnet to measure semantic orientations of adjectives. In *Proceedings of 4th International Conference on Language Resources and Evaluation* (Lisbon, PT). 2004, 1115–1118.
[16] Kang, H., Yoo, S. J., and Han, D. Senti-lexicon and improved Naïve Bayes algorithms for sentiment analysis of restaurant reviews. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2011.11.107.
[17] Kim, D. and Kim, C. Forecasting time series with genetic fuzzy predictor ensemble. *IEEE Trans. Fuzzy Systems* 5 (4) (1997) 523–535.
[18] Kim, H.C., Pang, S., Je, H.M., Kim. D., and Bang, S.Y. Support vector machine ensemble with bagging**.** *Lect Notes Comput Sci*. (2002) 131–141.
[19] Kim, H.C., Pang, S., Je, H.M., Kim. D., and Bang, S.Y. Constructing support vector machine ensemble. *Pattern Recognition* 36 (2003) 2757 – 2767.
[20] Osherenko, A. and André, E. Lexical affect sensing: Are affect dictionaries necessary to analyze affect? In *Proceedings of the 2nd international conference on affective computing and intelligent interaction (ACII'07)* Springer-Verlag, Berlin. 2007, 230-241.
[21] Paltoglou, G., and Thelwall, M. A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the 48th Annual Meeting of the ACL*, 2010, 1386–1395.
[22] Pang, B., and Lee, L. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting of the Association for Computational Linguistics (ACL)* (Barcelona, Spain, July 21–26, 2004). 2004, 271–278.
[23] Pang, B., and Lee, L. *Opinion mining and sentiment analysis*. Foundations and Trends in Information Retrieval, 2(1-2), (2008), 1-135.
[24] Pang, B., and Lee, L. Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting of the ACL* (University of Michigan, USA, June 25–30, 2005). 2005, 115–124.
[25] Pang, B., Lee, L., and Vaithyanathan, S. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*. ACL, 2002. Volume 10.
[26] Platt, J. Fast training of support vector machines using sequential minimal optimization. In: Scholkopf, B., Burges, C., Smola, A. (eds.): *Advances in Kernel Methods - Support Vector Learning*. MIT Press (1998).
[27] Porter, M. F. Snowball: A language for stemming algorithms. 2001.
[28] Prabowo, R., and Thelwall, M. Sentiment analysis: A combined approach. *Journal of Informetrics*, 3(2), (2009), 143–157.
[29] Rodriguez, J.J. and Kuncheva, L.I. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28 (10), (2006) 1619–1630.
[30] Sharma, A. and Dey, S. A comparative study of feature selection and machine learning techniques for sentiment analysis. In *Proceedings of the Proceedings of the ACM Research in Applied Computation Symposium* (San Antonio, Texas, 2012). ACM, 2012, 1-7.
[31] Sharma, A. and Dey, S. Performance investigation of feature selection methods and sentiment lexicons for sentiment analysis. *IJCA Special Issue on Advanced Computing and Communication Technologies for HPC Applications*, 3 (2012), 15-20.
[32] Sharma, A. and Dey, S. A document-level sentiment analysis approach using artificial neural network and sentiment lexicons. *ACM SIGAPP Applied Computing Review, 12,* 4 (2012), 67-75.
[33] Shimada, K., and Endo, T. Seeing several stars: A rating inference task for a document containing several evaluation criteria. In *Proceedings of the PAKDD*. Springer, LNCS volume 5012, 2008, 1006–1014.
[34] Tan, S., and Zhang, J. An empirical study of sentiment analysis for Chinese documents. *Expert Systems with Applications*, 34, 4 (2008), 2622-2629.
[35] Thomas, M., Pang, B. and Lee, L. Get out the vote: Determining support or opposition from congressional floor-debate transcripts. In *Proceedings of the 2006 conference on empirical methods in natural language processing (EMNLP 2006)* (Sydney). 2006, 327–335.
[36] Trivedi, S. K. and Dey, S. Interplay between Probabilistic Classifiers and Boosting Algorithms for Detecting Complex Unsolicited Emails. *Journal of Advances in Computer Networks* 1, 2 (2013), 132-136.
[37] Tsytsarau, M., and Palpanas, T. Survey on mining subjective data on the web. *Data Mining and Knowledge Discovery* (2011), 1-37.
[38] Turney, P. D. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on ACL* (ACL'02) (Morristown, NJ, USA) ACL, 2002, 417–424.
[39] Vapnik, V. *The nature of statistical learning theory*. New York Springer (1995).
[40] Xia, R., Zong, C., and Li, S. Ensemble of feature sets and classification algorithms for sentiment classification. *Information Sciences*, 181, 6 (2011), 1138-1152.
[41] Ye, Q., Zhang, Z., and Law, R. Sentiment classification of online reviews to travel destinations by supervised machine learning approaches. *Expert Systems with Applications*, 36, 3 (2009), 6527–6535.
[42] Zhu, F., and Zhang, X. Impact of online consumer reviews on sales: The moderating role of product and consumer characteristics. *Journal of Marketing*, 74, 2 (2010), 133-148.
[43] Zhang, X., Wang, S., Shan, T., and Jiao, L.C. Selective SVMs ensemble driven by immune clonal algorithm. In: Rothlauf, F. (Ed.) *Proceedings of the EvoWork- shops*, Springer, Berlin. (2005), 325–333.

ABOUT THE AUTHORS:

Anuj Sharma is Assistant Professor of Information Technology at Chandragupt Institute of Management, Patna, India. He completed his doctoral programme (Fellow Programme in Management) in Information Systems from Indian Institute of Management Indore, India, and Master of Technology from National Institute of Technology Jalandhar, India. His research interests are sentiment analysis and opinion mining, social media mining and artificial intelligence.

Dr. Shubhamoy Dey is Professor of Information Systems at Indian Institute of Management Indore, India. He completed his Ph. D from the School of Computing, University of Leeds, UK, and Master of Technology from Indian Institute of Technology (IIT- Kharagpur), India. He specializes in Data Mining and has 25 years of research, consulting and teaching experience in UK, USA and India.

Aspect-driven, Data-reflective and Context-aware User Interfaces Design

Tomas Cerny, Karel Cemus
Czech Technical University,
Charles square 13
12135 Prague 2, Czech Rep.
{tomas.cerny,cemuskar}@fel.cvut.cz

Michael J. Donahoo, Eunjee Song
Baylor University,
One Bear Place #97356
Waco, TX, 76798-7356, USA
{jeff_donahoo,eunjee_song}@baylor.edu

ABSTRACT

The increasing use of Web-based applications continues to broaden the user groups of enterprise applications at large. Since ordinary users often equate the quality of user interface (UI) with the quality of the entire application, the importance of providing easy-to-use UIs has been significantly increasing. Unfortunately, designing a single UI satisfying all end users remains challenging. To address this issue, researchers and developers are looking to Context-aware/Adaptive UIs (CUIs) that aim to provide end users with more personalized user interaction experiences. Although multiple proposals have been made, very few production systems provide such malleable interfaces due to the excessive cost of development and maintenance.

In this paper, we propose a technique that aims to reduce development and maintenance efforts of CUI to a level comparable with a single UI. Unlike most of the existing CUI approaches, our technique does not involve an external UI model. Instead, it aims to reflect runtime-information and structures already captured in the application, while extending them to provide an appropriate CUI. With this technique, developers do not design forms or tables directly for each page or panel. Instead they design generic and reusable transformation rules capable of presenting application data instances in the UI while considering the runtime context. To demonstrate our technique and its impact on CUI development and maintenance, we provide a case study. Moreover, we present our experience from its application to an existing production-level enterprise application, with high demands on performance.[1]

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*; I.2.2 [**Artificial intelligence**]: Automatic Programming—*Program synthesis*

General Terms

Design, Reliability

Keywords

Aspect-driven design, Inspection-based approach, Adaptive user interfaces, Reduced maintenance/development efforts

1. INTRODUCTION

Despite broad research in the area of Context-aware/Adaptive User Interfaces (CUIs), it is common practice in production applications to design a single UI that serves all types of users and contexts [26]. The primary reason for this one-size-fits-all approach to UI design relates to the costs of development and maintenance for multiple UI versions. For example, Kennard et al. [14] states that around 48% of application code and 50% of development time are devoted to implementing UIs. Thus, providing multiple versions of UIs for individual users is typically considered to be unrealistic.

With most existing programming techniques, it is difficult to support adaptive UI features because such approaches capture field-specific information twice, once in the data-model and again as a reference in the presentation that is often specified through a domain specific language (DSL)[23] with weak type safety. In addition, current practices realize multiple UI concerns [11] mixed together in a single component, which makes such a component less cohesive and hard to reuse. As shown later, this results from the inability of conventional approaches to capture different concerns separately [16]. The development of less cohesive components results in multiple, highly-similar components that only differ in details. Having a multi-location field definition and multiple similar components for a slightly different presentation brings further difficulties throughout the development. For example, changing the underlying data definition requires all of its presentation components to be updated, which is a non-trivial task. Considering that such a component update process is manual, it is most likely to introduce more errors (particularly with no type safety) or omit required component updates, which eventually results in presentation inconsistencies.

Our proposed technique avoids information restatement, as well as supports separation of concerns [11]. The first part is achieved by the utilization of information from an application's data-model and its existing structures that are obtained from the automated code-inspection and by reflecting its meta-model. Such information is then extended and transformed into the UI. To support the second part, this transformation takes multiple steps and bases itself on model-driven development (MDD) [17], generative programming (GP) [9] and aspect-oriented programming (AOP) [16]. Concerns that are in conventional approaches tangled together are now separated into easy-to-maintain, reusable units, called *aspects*. The transformation process weaves all separated concerns together at runtime and thus allows us to

consider user-context conditions individually. In addition, the transformation process uses a set of generic mapping rules [20] that allows designer to adjust the output as well as to integrate any third-party runtime conditions. From the end-user perspective, the resulting UI dynamically adapts to context and considers all concerns to satisfy expectations. To evaluate our technique, we developed an open-source library called AspectFaces and demonstrate its use in a case study with an enterprise JEE6 application. Furthermore, we provide our experience from a production-level application accessed by users from all around the world with high performance demands.

The main contribution of our approach is the *reduction of information restatement* in UI development and the *separation of UI concerns* that are directly responsible for tangled UI code. Multiple information restatement steps required in existing approaches collapse into a *single focal point of information* in our approach, which makes the *enforcement of its UI compliance* easier. Since it is executed at runtime, it can *dynamically adapt the UI to a user-specific context.* The approach *reduces both development and maintenance efforts* through component reuse. Despite the addition of these benefits, our approach has a *minimal impact on application performance.*

The remainder of this paper is organized as follows: Section 2 describes the background of adaptive user interface development. Section 3 provides an overview of existing approaches. Our approach is presented in detail in Section 4, and its evaluation is discussed in Section 5. The final section presents our conclusion and future work.

2. BACKGROUND

One approach often taken to deal with system complexity is to break the system down into units of behavior or function such as subsystems, modules or objects, called functional decomposition in Object-Oriented Programming [16] or more generally in General-Purpose Languages (GPL). Such a decomposition concept is necessary because it helps one to put logically-related concerns together, improves the readability and reusability, and eventually supports the ease of maintenance [18]. In addition to functional decompositions, GP [9] and AOP [16] proposes another way of thinking about program structure. GP proposes the use of a GPL language together with problem descriptions in the form of DSL [23]. The resulting application code is generated at compile time from the DSL specifications that extends the GPL code or produces its variations. In AOP the key unit of modularity is an aspect. Aspects can integrate to GPL modules at runtime or compile time. An aspect enables the modularization of concerns, such as transaction management, that normally cross-cut multiple modules and objects [18].

UI development also employs such decompositions, but data presentation makes the decomposition process more challenging, especially when using DSL to describe the UI, which is common for web systems. For example, consider designing the Person form given in Figure 1. The arrows highlight various concerns considered in the design. `Arrow 1` shows that form fields are bound to a particular data class - an entity, and its fields. This binding means that, for example,

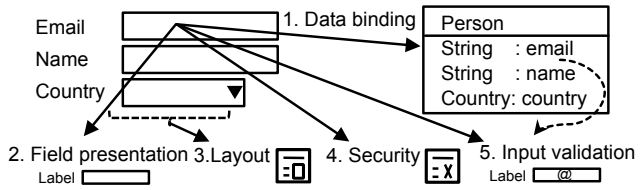


Figure 1. UI form decomposition

Listing 1. Sample code snippet for form in Figure 1

```
<table class="classLayout">
<tr>
 <td>Email:</td>
 <td><x:input id="email" value="#{i.email}"
     render="#{bean.render('email')}"
   validate="#{bean.validate('email')}"/></td>
</tr><tr>
 <td>Name:</td>
 <td><x:input id="name" value="#{i.name}"/></td>
</tr><tr>
 <td>Country:</td>
 <td><x:smenu id="country" value="#{i.country}"/></td>
</tr>
</table>
```

when the field called `name` in `Person` splits into `first name` and `last name`, its corresponding form field must split as well. Unfortunately, there is no enforcement mechanism to guarantee that the corresponding entity and its UI comply with each other unless a language with type-safety is used. An entity field UI presentation is denoted by `Arrow 2`; an appropriate UI widget with its properties are chosen based on the type of a particular field and its constraints. Anytime a field constraint changes, an underlying widget or its properties should reflect the change as well. However, there is no automated mechanism to do so, thus a manual update is necessary for each field change. `Arrow 3` demonstrates that the form may allow one to select a particular presentation layout. A layout is responsible for rearranging form fields in a given order, grouping them together or presenting them within a given screen size. Designing a non-trivial form layout often results in a layout code entangled together with form fields. We provide an example of tangling such concerns in Listing 1. When an application adjusts a form layout at runtime based on a given condition, it is possible that multiple cloned variants of the same form must physically exist. For example, consider a slight modification of layout in Listing 1 for a user with wide screen. We would place the `name` to top-left, `country` to top-right and the `email` to bottom spanning both columns, in this case only the layout concern changes, but other concerns are unchanged, but having all the concerns at the same place limits the reuse and we end up with two forms. Next, `Arrow 4` indicates that form fields should consider additional UI conditions such as security or visibility. For example, some fields should be rendered as read-only or left unrendered based on the given user authorization. In order to apply the conditionals, we further extend the form fragment, leading to more complex readability and perhaps duplication among fragments applying various layouts. Finally, `Arrow 5` shows that certain constraints from the bound entity fields should be applied for its input validation. For instance, web applications with client-side validation must restate constraints in a scripting language, such as JavaScript. Listing 1 shows a very sim-

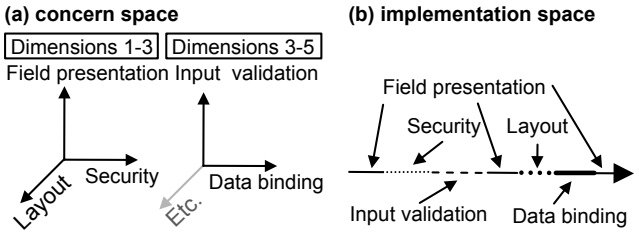


Figure 2. (a) Concern / (b) Implementation space

plified implementation of Figure 1; this JavaServer Faces (JSF) DSL code shows data binding to the form through a data instance *i* (*value* attribute in widgets), field representations through UI components (*x:input*), table layout tangled through the fields, security condition (*render* attribute) and validation (*validate* attribute) determined by methods in a controller accessible in the context as *bean*. The maintenance of such fragments becomes difficult because all five concerns are captured together, and it is non-obvious which code refers to a specific constraint such as security, presentation or layout. The reuse of individual concerns in such UI fragments is very limited since it only allows slight variations of concerns. CUI design only compounds the problem since it typically increases the number of concerns.

With the AOP approach [18] this same problem can be seen as an n-dimensional *concern space* that is expressed in the *implementation space* using a one-dimensional language. Unfortunately, orthogonality of concerns in the concern space gets lost (collapsed) when it is mapped to the one-dimensional implementation space. For our case, we have a 5-dimensional concern space as shown in Figure 2 (a). This concern space is mapped into one-dimensional implementation space in Figure 2 (b). This corresponds to what we see in Figure 1 and the one-dimensional implementation in Listing 1.

So far we looked over a basic UI design example. Next, let us consider our expectations of an effective UI design. In order to design a CUI effective from the development perspective, we must consider multiple quality attributes. A good CUI design should allow designers to capture the expected functionality but also other non-functional attributes. First, it provides multiple presentations, different layouts, easy addressing and integration of various concerns, adaptive to application runtime context, third-party integration (security), etc. Second, it should be easy to develop and maintain these concerns with low coding effort, while preserving development approaches already known to the developer. Third, a good design should reduce information restatement/duplication across the application and, if possible, mitigate errors caused by UI inconsistency. Fourth, while reducing restated information a single focal point of information should exist to reduce multi-location changes. Fifth, a good design separates tangled concerns [11] into readable code fragments to support their reuse and maintenance.

When we consider conventional approaches and look back to Figure 2, we should note that multiple other concerns may exist for CUI, thus growing the concern space. For example consider concerns such as user's location, data submission error-rate, age, temporal information or layout adjusted to the user's screen size, etc. With no doubt, since the number of concerns in Figure 2 (a) grows and the complexity represented by Figure 2 (b) becomes even greater, it is not reasonable to keep concerns tangled together. Such tangling is directly responsible for increased development and maintenance efforts, diminishing readability, limiting reuse, higher possibility creating errors, etc.

3. RELATED WORK

We split the description of related work onto two parts. The first introduces existing development approaches that could be used to improve the UI design. The second part describes existing UI design approaches and CUI.

3.1 Development approaches

Model-Driven-Development (MDD) [8] argues that a design should involve models that act as a centralized location for information and design decisions. Such a model is then transformed into application code. The idea of MDD is promising regards reduction of restated information, on the other hand we must consider that majority of productions systems involve coding and to transform existing systems to MDD would be expensive. For instance, consider a situation when an application applies pure Object-Oriented Design (OOD) to its backend, while the frontend is designed in a MDD way. Such model would most likely use a custom DSL language for its description. As a result, the model must restate information from the backend. In addition, maintaining the connection between UI and backend requires significant effort. Furthermore, in MDD, problems such as cross-cutting concern applicable to code development are present as well [25]. Although we could use multiple models to capture various concerns and integrate them together, the appropriate generic integration mechanism is missing [29]. Another issue is that MDD suffers during adaptation and evolution management [25]. Once deployed, such systems experience changes in variations, which often take place in code rather than in the model itself so code regeneration from the higher abstraction model can be impractical and the manually added information can get lost [8]. Designing a system that considers multiple variations with MDD often leads to a compile time allocation of many states and configurations that can grow exponentially [25]. A runtime solution is less common and such use can degrade performance [30].

Generative Programming (GP) [9] emphasizes specific domain methods and their integration with OOP. GP can be seen as programming that generates source code through domain-specific code fragments or templates to improve designer productivity. Authors of GP [9] define it as a design approach to combine and generate specialized and highly optimized systems fulfilling specific requirements. The goal is to address the gap between program code and domain concepts, support reuse, adaptation, simplify management of component variants and to increase efficiency. It addresses *separation of concerns* [11] that propose dealing with one issue at the time and avoiding code that mixes multiple concerns. It further addresses *parameterization* or *separation of the problem space from the solution space*. Such separation splits the problem space and its domain-specific ab-

stractions and maps them to the solution space with available implementation components (similar to what we show in Figure 2). GP uses DSL with the loss of language generality and emphasizes automatic configuration and generation at the compile time, similar to MDD. While there are substantive differences between MDD and GP, we note many similarities. While the MDD uses models and GP uses OOD and DSL, a model is often designed and described through a DSL, thus the parallel is close. GP directly addresses separation of concerns, which is not the primary goal for MDD. On the other hand, both lack the ability to use runtime information and effectively perform at runtime. The GP integration of concerns does not have a generic format, which exists in AOP.

The features lacking in MDD and GP are addressed by AOP [16, 18, 32]. AOP provides methods that allow us to capture different concerns separately in independent code fragments, as well as enabling runtime weaving. The mechanism to integrate concerns is well described and generic. The problem is decomposed to functional OOD units and aspects, and these weave together to obtain system implementation. In the OOD units, we indicate the possible aspect integration location through join points. These join point are recognized by the aspect weaver, a compiler that integrates aspects to the resulting code/behavior. The product of aspect weaver can have the same execution properties as tangled code, but with the advantage that all the concerns can be defined separately to support readability and maintenance. In [16] authors shows reduction of the total of lines of code (LOC) from 30,000 LOC to 1,000 LOC for the AOP version with the same properties. Compared with AOP, GP [9] has larger scope, involves DSL, and emphasizes the automatic configuration and genericity at the compile time. AOP weaving can be made at compile time or at runtime. The weaving process often involves meta-programming [9, 12] introduced next.

Meta-programming (MP) allows programs and languages to modify their structure and behavior at runtime. Many contemporary, statically-typed programming languages have the ability to describe themselves, which is called Reflection [12]. Reflection is based on an architectural design pattern [5], which gives us an opportunity to inspect classes, their fields and methods while not knowing their names at compile time. This mechanism allows programs to dynamically adapt the program to different situations. MP is a great instrument for code inspection and information extraction. On the other hand, we must consider the impact on performance. To face the performance bottleneck, it is possible to use cache or to involve code generation at application deployment time. Another issue for MP is its testing, because MP programs are not type safe and thus its maintenance becomes complex.

We summarize these approaches in Table 1. We compare the selected abilities such as, whether it applied to runtime, addresses separation of concerns, reduces restated decisions and information restated from existing structures, whether it applies inspection, how well it does for evolution management, if it is adaptable towards changes or existing structures and its compatibility with OOD.

Table 1. Comparison of design approaches

Ability/Approach	OOD	MDD	GP	AOP	MP
Compile time approach	yes	yes	yes	yes	yes
Runtime time approach	yes	slow	no	yes	yes
Separation of concerns	no	no	yes	yes	no
Reduces restated decisions	no	yes	yes	yes	no
Reduces restated information*	no	no	no	no	yes
Model or Code inspection	no	yes	no	no	yes
Evolution management	good	bad	good	good	bad
Adaptable towards changes*	no	no	yes	yes	yes
Transformation based	no	yes	yes	yes	no
Synergy with OOD	-	no	yes	yes	yes
**(from existing code structures/application backend)*					

3.2 UI design approaches

UI design approaches could be divided into three groups, as suggested by Kennard et. al. [15]: approaches using interactive graphical specification, model-based UIs, or language-based UI generation. The first group allows developers to sketch UIs on the screen with the corresponding source code automatically generated in the background. While this approach works for toy applications, it does not consider the maintenance and advanced manual code changes required by enterprise applications. Model-based UIs use a model to describe the UI, which is then transformed to an appropriate UI presentation considering various conditions. Unfortunately, when a model binds to an existing application backend, the model must restate information from it. The language-based tools derives the UIs from the language and domain objects. The advantage is that the UI is always consistent with the application backend, a feature missing from the previous groups, but domain objects are somewhat weak with respect to UI description and this only basic UIs can be derived [8].

In our research, we consider another view of UI design classification by classifying approaches into either *restate-to-extend* or *inspection-based*. The first approach requires that the same information in a system is captured twice at different locations, while preserving its integrity. Such information duplicity is then applied to a particular concern such as UI presentation. Development using this approach typically involves interactive graphical tools, UI model-based generation tools [19, 25], external models for UI representation [22], and DSL tools [13]. The main drawback of this approach stems from the duplication of source information and maintenance efforts when source information changes.

Inspection-based approaches use existing information accessible by code-inspection. The main effort is placed on the information source that must capture sufficient information to derive a specific concern. Design using this approach typically involves language-based tools. The disadvantage of this approach is that source information does not necessarily capture all needed concerns. Multiple research proposals such as [7, 14, 15] utilize automated UI generation by applying code-inspection. These approaches inspect previously captured information, build an ad hoc structural model, and transform it to the UI. This simplifies both the development and maintenance since it reduces restated information. The difficulty is that such an approach cannot generate the UI unless provided additional information, typically supplied by additional markup within the source information [8]. Experience from industrial standards such as Java

Table 2. Comparison of related work

Features	[28]	[22]	[3]	[4]	[19] [24]	[25]	[14] [15]	[8] [7]	[27]
Model-based	-	-	+	+	+	+	-	+	-
Runtime approach	+	+	+	+	-	+	+	+	-
Adaptive UI	+	+	+	+	+	+	-	-	+
Reduces code	-	-	+	-	+	+	+	+	+
Restate-to-extend	-	+	+	+	+	+	-	-	+
Addresses cross-cutting concerns	-	-	+	-	-	+	-	-	+
Code-inspection	-	-	-	-	-	-	+	+	-
Uses enterprise technology standards	-	-	-	-	-	-	+	+	+

EE [2, 10] shows that the data-model already capture additional markup for persistence and validation constraints, and the same approach can be applied also for presentation [8] and security. Our classification allows combining inspection-based approach with model-based design [21], which is not possible with the Kennard's [15] classification.

However, both *restate-to-extend* and *inspection-based* approaches need to deal with information transformation to the UI. Often we see hardcoded transformation rules, such as in interactive graphical specification tools. Generic and configurable rules allow designers wider options and adaptations. Based on [20], generic mapping rules allow easy reuse among systems. In our work, we provide such generic transformation rules while considering aspect-query-based features that involve data structures and field extensions. Furthermore, none of the above approaches directly addresses cross-cutting concerns, although related research on this topic exists for model-based [25, 21] and GP approaches [27].

3.3 Context-aware UI

A basic overview of adaptability and adaptivity is provided by [31]. Both terms refer to knowledge-based self-adaptation, but in the case of adaptivity, it relates to interactive session and adaptability that can be deduced before the interactive session. CUI may address both these features. The idea of CUI is studied in multiple domains. For example, we can see its application to [28] electronic cooking assistants in kitchens to adjust layout, in a hospital navigation case [22] and in a house control unit example [3]. Multiple CUI design methods require the target environment and possible variations of the user interface at design time [19, 24], but in [4] the authors argue that future adaptive systems need to consider runtime information to adapt, while design time approaches are not sufficient. [25] and [3] apply aspect-oriented techniques to a model-based approach to deal with multiple degrees of variability that depends on user needs and context, on the other hand they require to restate information. Although, most of the CUI work focus on presentation, adaptability and adaptivity features of UI, they typically apply model-based approaches and restate information from application backend. None of the related approaches provide evaluation regarding runtime performance and production experience, they rarely consider maintenance efforts, and only indirectly address specific concerns such as layout [28]. We provide the summary of selected related work regarding the UI design in Table 2. In our approach, we address all the mentioned features and elements as well as avoid the *restate-to-extend* approach.

4. READ : RICH ENTITY ASPECT/AUDIT DESIGN FRAMEWORK

As shown in the related work, in order to design a CUI with low development and maintenance efforts, we should avoid definition of an additional model that restates information captured elsewhere in the application. Instead we should consider a *code-inspection* approach (MP) and synergy with knowledge about transformations (MDD & GP) as well as to address separation of concerns (GP & AOP).

First, we specify information that we want to reuse such as data structural information and their constraints. All these can be found at the application data-model. Assuming that the data-model design uses OOP and the language supports reflective mechanisms, we gain access to data structures. Besides this we need an access to the application context at runtime. Thus when we need to display data in the UI, we can inspect the given data class and get its structural model, that captures information about the class, its fields and field constraints.

Application context and structural model is then the subject of transformation to the UI. In order to effectively handle both adaptivity and adaptability, the transformation takes place at runtime and uses generic, easy-to-extend transformation rules. In order to design such rules, a single rule instance cannot bind to an individual data or data field but to something more general. In our approach, each rule instance consists of a query part and a suggestion, in the AOP terminology a pointcut and an advice. The query part is an evaluable boolean indicating whether the rule applies for a given context (given data field in given context). If so, the rule's advice is given; if not a next rule in the list is considered. The query can question a data field structural model, application context or both using logical and arithmetical operations. The advice provides the integration DSL template that is used for the data field.

A collection of customizable DSL templates is associated with the transformation rules. Such template uses the target presentation language and integration rules in it, to integrate additional concerns. An integration rule again consists of a pointcut and an advice. The pointcut is uses the same query constructs to question the data field structural model and context. An advice is different; it is a DSL content that can reference the structural model properties or context variables. All integration rules are considered for given template if a rule pointcut holds, then the advice content embeds to the template or resolves the reference to the structural model (such as field name, type, etc.). The result of the template interpretation is a UI code fragment in the target DSL language representing given data field considering all concerns, but layout.

Right after all data fields process through the transformation, then a proper layout template integrates. This process is similar to XSLT. The resulting output is a DSL fragment reflecting data, context and integrates all considered concerns. The last part of our approach is runtime integration of the resulting DSL code to the application UI. This involves the DSL code compilation and UI embedding.

4.1 Introduction to READ conceptual model

Our framework involves AOP and, in order to describe an AOP framework, [32] suggests describing its conceptual model with three main components:

Join Point Model: defines available join points

Pointcut Language: defines the query language to select a subset of join points

Adaptation Mechanism: allows adding or modifying functionality at selected join points

These components describe existing frameworks such as AspectJ or Hyper/J in which we often modify or add functionality upon method call or code execution. In our case, the adaptation mechanism does not constraint any method or code execution but deals with transformation and composition.

In READ we identify two sources of join points: the 1) structural data model and 2) application runtime context (a subset exposed to the READ process). A structural data model provides entity and field names, data types and field meta-instructions with their parameters [10, 2]. Application runtime context can consist of any kind of information, such as user access rights, geo-location, local context for presentation, device screen size, etc. We could even count user error-rate throughout the application interaction and based on that show a tooltip or help upon page load. Both sources provide us join points that can be considered in the transformation process. More specifically these join points allow us to support generic/reusable transformation rules.

The AOP terminology deals with two types of join points [32], static and dynamic. *Static join points* can be characterized as a location in the source code; the selection criteria refers to static structure, and it is known at compile time where and how to enhance the code for all invocations. *Dynamic join points* correspond to elements in code, but at the same time to a runtime condition that specifies the selection only to certain invocations. While the application context corresponds to the dynamic join points the structural data information consists of both types of join points. For instance, the field name is static, while field access rights denoted by field annotation can be dynamic.

The *pointcut language* defines the query language to select a subset of join points. READ uses an expression language known as Unified Expression Language[1] (EL). EL consists of constructs for conditionals and arithmetical operations, understands basic types, and can evaluate any expression referring to its context. In READ, the EL context has access to the elements of the structural model (from the field perspective) and to dynamic context variables that are populated by application designer. The pointcut language can query all information in the EL context. It is also possible to define custom utilities or functions that integrate third party libraries and pass them to the context and thus expose them as dynamic join points. The language uses both the *state-based* and *specification-based* constructs [32]. Later, in the next section we show how to access elements for the structural model and context from the EL.

The *adaptation* mechanism uses the above described join points, and based on their association to a particular data field or global context it selects an appropriate UI transformation rule instance that suggests an integration template. An integration template applies the same join points for integration rules. In both cases, pointcuts query the join points to get an advices, either for the transformation or concerns integration. We give an example of both rules later. The integration template uses an aspect language for concern integration but at the same time uses the target UI language. The layout integration is the last part of the adaptation mechanism. It is similar to the integration template in that it uses a DSL language from the target domain language to describe the layout and an extra markup to locate specific or anonymous fields in the template.

In order to an a new concern to the system, we either need to expose it to the READ context, or to extend the data structure through new annotations. This way the novel concern becomes accessible by EL, thus by both the transformation or integration rules.

4.2 READ lifecycle

Next, we briefly look at the READ lifecycle in Figure 3 that denotes the main stages (a-f). In the UI, we aim to display a given data instance (a) in the target UI language. In order to do that, we use a custom component that is associated with a specific component handler (b,c) and the displayed data instance reference. The responsibility of such a handler is to provide the content for the component. Thus this handler is the connection between the target UI language and integration of our approach. The custom component takes as an input the data instance and considers other context information. For instance, the context can be an indication that the aimed content is a read-only presentation, fields named *"notes"* are ignored, etc. First, the handler aims to get the data structure, the data structural model. Either, this structural model is found in the cache from a previous use or the data instance goes through an MP inspection (d) and the result is passed to the cache. A cloned instance of the structural model, which is the result of the inspection (d1), is interpreted in a given context using the Annotation Driver Participant Pattern (ADPP). This may result in modification of the structural model. Such a context-aware structural model is then passed to the transformation phase (e) together with the context. In the above sections, we mentioned three phases of transformation. Each data field from the context-aware structural model of given data is the subject of transformation and concern integration (e1*). This results in a UI code representation in the target UI language for each field. After all fields process the layout is integrated (e21) receiving the entire data UI representation as result. The last stage interprets the resulting UI fragment and integrates it to the UI (f).

4.2.1 READ UI integration

Consider the process of designing a web page where we want to display application data in the main panel. Normally, such a main panel contains code similar to Listing 1 to describe the data. Instead a custom component is used as shown in Listing 2. A component prefixed *"af"* takes as an attribute a reference to a data instance accessible through a controller (in our case called a bean and a local con-

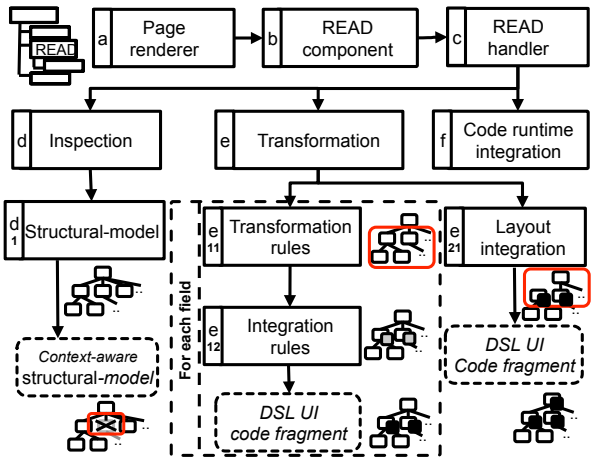


Figure 3. READ lifecycle

Listing 2. Example use of READ UI component

```
<h:outputText value="Person Info Form" />
<af:ui instance="#{bean.instance.personInfo}"
     layout="personInfo-wide-layout"
     edit="true" ignore="password,notes" />
<h:commandButton action="#{bean.save}" value="save"/>
```

text). This component is associated with custom handler that pushes the local context to be considered in the READ context and issues the phases described in Section 4.2 to receive the CUI for the given data instance.

4.2.2 *Inspection phase*

The inspection phase is rather complex; thus we provide more details. It audits classes of the data-model (entities). It specifically looks for class name, class restrictions, its fields and field constraints. While aiming to build on existing industry standards, we look into the following profiles for persistence and input validation. Java EE considers Java Persistence API data-model profile [10] to support object-relational mapping and Java Beans Validation [2] to validate input of the persistent objects. Both of these standards are applied to existing systems, and we consider them in the inspection. Model-based profiles reflecting the above data-model extensions were introduced in [8]. The same approach can be further extended for role-based access control and for presentation [8]. Table 3 shows the class structure and field elements and a subset of selected extensions applicable to data-model class fields. The table describes extension names, denotes their applicability and also highlights the name of a variable under which it is accessible as a join point. The inspection phase considers all such extensions and it is further possible to consider other custom ones. The result of the inspection is a structural model, a three-level composite structure reflecting the class-level, field-level constrains or extensions. The structural model provides these properties through the variable name (the right most column in Table 3). Furthermore, in this phase, it is possible to apply runtime context to modify the structural model. For example, it is possible to locally modify structural model fields based on a given condition such as ignore a field, change field constraints or to expose a new variable/object in the context and make it available as a join point.

Table 3. Subset structural model elements accessible as join points

Extension	Description	Data type	Context variable
Class-level attributes			
-	class name	-	entity, Entity
-	full class name	-	fullClassName
Field-level attributes			
-	field name	-	field, Field
-	field type	-	dataType
Field-level constraints			
1. Persistence profile			
Column	DB table column props.	Any	notNull,required, maxLength,unique..
joinColumn	DB table column props.	Any	notNull,required, unique..
Temporal	Date, Time, TimeStamp	Date	temporal
	..		
2. Validation profile			
Length	Value length in the range	String	minLength, maxLength
Min, Max	Value in the range	Number	min, max
Email	Match email	String	email
Pattern	Matches the reg-exp	String	pattern
Future, Past	Future/Past date	Date	past, future
NotNull	Not null value	Any	required,notNull
NotEmpty	Not empty value	Any	required,notEmpty
	..		
3. Presentation profile			
UiLink	Web link expected	String	link
UiText	Long text expected	String	text, cols, rows
UiParam	Any Param expected	Any	param (name, value)
UiHtml	Html expected	String	html
UiPassword	Secret text expected	String	password
UiType	Type of widget to use	Any	type
UiOrder	Order in view	Any	order
UiTableOrder	Order in table	Any	tableOrder
UiIgnore	Ignore field in UI	Any	ignore
UiPattern	UI Script regular expr	String	uiPattern
UiProfiles	To support grouping	Any	Profiles
	..		
4. Access control profile			
Restrict	Third parti restriction	Any	restrict
UiUserRoles	Values specifies user role	Any	roles
	..		

4.2.3 *Transformation phase*

The transformation phase consists of three parts. This section provides details to build the whole picture. We have introduced how to apply a READ component to the UI DSL, how to receive the data instance, and process the inspection. The result of this is a context-aware structural-model of the data instance. Such a structural model is accessible to the transformation through a EL context using variables described in Table 3. The application context and any third-party elements or properties specified together with the READ component or in its handler are passed to the EL context. For example, consider the entity described in Listing 3 (it uses annotations [10] and [2]) and context specified in Listing 2. The structural model reflects information about the data, its fields and field properties. The settings in Listing 2 modify the model instance, and in our case fields password and notes are ignored. It also exposes the aim to edit the data to the context, as well as specific layout to use for this particular UI page.

The first stage applies transformation rules to data structural model fields. An example of a subset of such rules captured in a DSL is shown in Listing 4. Consider the *first*

name field from the Listing 3. Based on the type, we use the String group of the rules, and since none of the rule pointcuts (expr attribute) apply, we use a default advice a *textTemplate*. For *email*, we pick a *emailTemplate* since the second pointcut applies. The pointcut could use any variable available in EL context (consider context variables in Table 3 from the structural model, or any variable exposed to the component handler). For example, we could ask whether a field of type String is short and required and whether it is Monday and user is from Prague. Such pointcut would look like this:

```
maxLength<100 and required
and timeUtil.getDayName() eq 'Monday'
and locationUtil.city.toLowerCase() eq 'Prague'
```

Each field from the structural model of the given data instance gets advice from the transformation rules. The advice is a DSL template that describes a basic presentation in the target UI language, with integration rules to integrate various concerns, such as binding, help, validation, etc. An example template is shown in Listing 5. Note that this template consists of many references to the structural model through the context variables. These references are part of the integration rules. In order to understand the mechanism, we show three variants of integration rules in Listing 6. It shows (a)-full/ (b)-brief/ (c)-shorten version of integration rules. It integrates join points from a given field. The full version separates the pointcut and advice part; when the pointcut evaluates to true, then the body applies. Brief version provides the same result but needs less code. The shorten version fits to common cases and needs the least code.

The last phase is layout integration. Layout is given by the READ component or deduced dynamically using the handler. This process is similar to XSLT, but it can express anonymous fields and iterate over repeating layout pattern. Consider Listing 7 that shows an HTML table decorating data fields. The layout has a repeating code pattern of two columns for anonymous fields with up to 100 iterations, and a reference to an explicit field called *notes* that spans over two columns in the last row. If fewer fields exist than specified, only the given amount applies. The specific fields take precedence in resolution. The result is a CUI code fragment that the READ handler integrates to the UI.

4.3 Design with READ

Next, we discuss software design with the use of READ. Assuming that we build on the top of an enterprise architecture using 3-layers, the system has a persistence layer that captures its data-model by classes and applies object-relational mapping (ORM). For example Java EE defines standards [10] for the ORM, which extends the class model with additional markup. Similarly validation [2] can be added. Generalization of such extensions and further enhancements are suggested by [8]. READ inspection uses all of this information for the structural model composition and for join points. Besides the data model, READ can also integrate business rules defined in the above layer. Preliminary work in [6] shows that business rules can be inspected and their definitions reused. This can be integrated into the READ context.

Listing 3. Example entity with additional markup

```
@Entity @Table(name = "personInfo")
public class PersonInfo {
   ...
  @UiUserRoles({"Admin","Owner"})
  @UiOrder(1) @Enumerated(EnumType.STRING)
  public Title getTitle() { return title; }

  @UiOrder(2) @NotEmpty @Email
  @Length(max=100) @Column(nullable=false, length=100)
  public String getEmail() { return email; }

  @UiOrder(3) @NotEmpty @Pattern(regex="^[^\\s].*")
  @Length(max=100) @Column(nullable=false, length=100)
  public String getFirstName() { return firstName; }

  @UiOrder(8) @UiProfiles({"US"})
  @NotEmpty @Column(nullable = false)
  public String getHomeState() { return state; }
}
```

Listing 4. Example transformation rules

```
<mapping>
 <type>String</type>
 <default tag="textTemplate" size="20"
   javaPattern="" minLength="0" maxLength="255" />
 <var name="Person.username" tag="emailTemplate"/>
 <cond expr="${email == true}" tag="emailTemplate"/>
 <cond expr="${link == true}" tag="linkTemplate"/>
 <cond expr="${maxLength>255}"tag="textAreaTemplate"/>
</mapping>
```

Listing 5. Example template for inputText widget

```
<x:inputText id="#{prefix}$field$"
     label="#{text[`$entity$.$field$`]}"
      edit="#{empty edit$Field$ ? edit : edit$Field$}"
     value="#{instance.$field$}"   size="$size$"
  required="$required$"          pattern="$pattern$"
 minlength="$minLength$"       maxlength="$maxLength$"
     title="#{text[`title.$entity$.$field$`]}"
  rendered="#{empty render$Field$
        ? 'true' : render$Field$}"/>
```

Considering common development approaches, we only expect data-model entity extension. We refer to such extended entities as rich entities. In the presentation layer, common components can be used together with READ components. READ components take as attributes an entity instance and addition presentation directives and build the presentation for given instance. Such a component can produce a form, table or a report. With READ, the developer does not design a form or a table directly per each page use. Instead, the developer specifies transformation rules that generalize mapping among entity fields and presentation widgets. Transformation rules are generic and can be reused among projects. The developer then designs integration templates that are used by the READ weaver (component). These templates are also generic and can be reused. While developing such templates is time-consuming, we must consider that all these templates are reused by the entire application, thus the initial work amortizes over the size of the software application. Furthermore, developers can design specialized or generic layout templates.

Where can we see the main benefits? First of all, the system presentation reflects the actual state of the software system. All data definitions, runtime contexts, and states are considered in the weaving process, thus the data presentation reflects or adapts to it at runtime. Second, with READ, the size of concern space does not increase the complexity of the system, and described concerns can be reused. Change of

Listing 6. Pointcut strategies for templates

```
(a)   $not empty minlength ; myVar = minlength $
      minlength="$myVar$"
      $$
//-------
(b)   $not empty minlength
       ? "minlength=\"".concat(minlength).concat("\"")
       : "" $
//-------
(c)   minlength="$minlength$"
```

Listing 7. Example layout template

```
<table class="classLayout">
<af:iteration-part maxOccurs="100">
 <tr><td>$af:next$</td><td>$af:next$</td></tr>
</af:iteration-part>
 <tr><td colspan="2" class="foot">$af:notes$</td></tr>
</table>
```

an individual concern is easy to locate and modify. Third, READ reduces errors because the entity becomes a single focal point of information, thus we do not need to restate information multiple times in the UI. Fourth, READ reduces development and maintenance efforts since a new entity presentation does not require any coding. In case a new presentation is needed for a given field, it is possible to define new transformation rule or design a new template. Fifth, READ naturally supports adaptive UI design because it evaluates conditions at runtime and separates concerns. Sixth, READ is open for integration with third party frameworks through the context or data-model extensions. READ templates can integrate any DSL. A more concrete example to this is when we use Java EE and JSF for presentation; it is possible to make templates for various component providers (such as PrimeFaces, RichFaces, Tomahawk, etc.). Seventh, READ does not bind the developer to a single-use approach; alternative approaches can be applied at the same time.

READ can integrate any new concerns in its context and can evaluate them at runtime. Our current approach is evaluated on component-based UIs, although it is not limited to them. The limiting factor can be the runtime integration of READ output to the UI. In some frameworks, this could be complicated, as it requires access to low-level UI compiler libraries. READ does not limit the expressiveness of the UI since designer can adjust the presentation in composition templates. READ can be used with partially rendered pages and AJAX rendered views.

5. EVALUATION

In this section, we provide an evaluation of our approach. First, we consider a UI that provides a single presentation and compare the manual approach with READ. Next we consider UI extensions to support adaptive features such as adjustments to access rights, users location, age, capabilities or screen size. In this evaluation, we compare the development costs for both approaches. We also consider a few maintenance scenarios. In the second part of the evaluation, we consider runtime performance. Third, we evaluate an existing production system that uses READ and provide our evaluation statistics. In the evaluation, we consider an existing application, which is a registration system for the worldwide competitions, the ACM-ICPC registration system (available at http://icpc.baylor.edu)

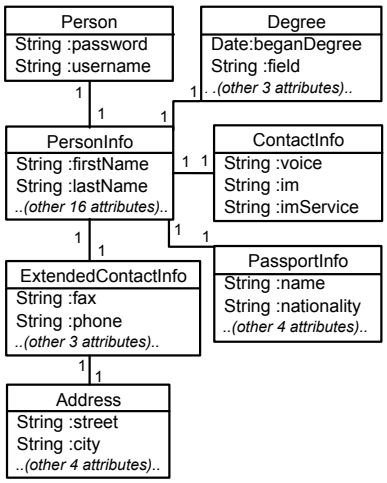


Figure 4. Evaluated application data-model

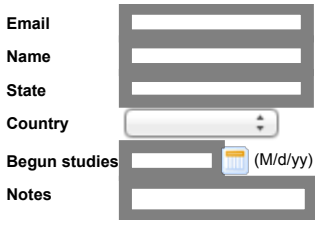


Figure 5. Sample simple UI Form

5.1 Development and maintenance impact

In this section, we consider a subsystem of an existing ACM-ICPC system used for the registration of users and user account management. The considered subsystem has the data-model illustrated in Figure 4. For brevity, the class attributes are abbreviated, and the class model does not list all attributes. The application follows mainstream development with 3-layer Java EE. The lowest layer consists of an object data-model with 7 entities with persistence and validation constraints markup [2, 10]. The business layer contains controllers with business logic, CRUD and search functionality. The presentation layer contains UI implementation using JSF technology (no type-safety).

First, we consider this application with a single UI. The UI part of the application contains search with result listing plus a detail and modification page. The presentation covers the entire data-model in Figure 4. Illustration of a page fragment, a form, is shown in Figure 5. Form submission of data is validated through enforced business constraints upon the submission. The application provides a single data presentation in one layout. In total there are 7 data classes and 46 fields presented in the UI. Excluding development configuration and external libraries, the application consist of 1342 physical lines of code (LOC) of Java, including persistence and business logic, 2221 LOC of XML presentation, and 373 LOC of XML of application configuration. The type-unsafe XML presentation exhibits 564 occurrences of restated information from the data-model and its constraints [2, 10]. Next, we implement the same application using READ. The data instance source code is extended with additional presentation marks [8] extending the field

Table 4. Efforts comparison

User interface	Simple features			Adaptive features		
Approach	Man.	READ	reuse	Man.	READ	reuse
Java LOC	1342	1530	1439	1658	1907	1754
UI XML LOC	2221	1715	1534	13072	5036	4508
Conf. XML LOC	373	442	373	373	649	373
Restated inf.	564	0	0	6768	0	0
UI Conditionals	0	0	0	240	20	20

constraints (see example in Listing 3). The main difference is that READ composes components presenting data. They combine information from data instance inspection, transformation rules and presentation/layout templates. None of the stages involve a direct reference to a particular data field, which leads to 0 occurrences of restated information in XML. This results in 1530 LOC of Java, including the additional data-model marks and a UI handler and 1715 LOC of XML including templates and transformation rules. This shows reasonable code reduction for the presentation part, but at the same time we must consider the maintenance impact. In the manual approach, we are directly responsible for restating information from data model in the UI, where READ handles this for us. With READ we avoid inconsistency and errors, while reducing development time. Even greater code reduction effect can be achieved on larger projects. Note that presentation templates and transformation rules can be reused among projects. In this case, the READ application results in 1439 Java LOC and 1534 XML LOC and equal configuration. The summary can be found in the first part of Table 4 (denoted by simple features). The aspect weaver itself is not included in the evaluation because it is a generic, reusable and external library (reasoning in [16]).

One serious drawback of this application example is that it considers a superset of all possible end users. Thus users with large screen are provided narrow layout, elderly might need to zoom the page, internationals might wonder why they need to fill in a *state*, and non-student registrants need to provide student-specific information.

Next, we consider a more user-friendly presentation supporting adaptability. It provides end-users with a presentation related to their origin using IP geo-location, adjusting to their browsing device screen size, conforming user rights, and fitting user age and capabilities. In total, there are 3 main layouts to conform the screen-size, although multiple data elements follow a custom field order among different layouts. Furthermore, we provide 4 different presentations for children, elderly, adult and experienced users, all possibly combining a given layout (see UI examples in Figs. 6-8).

The application following the mainstream development applies field restrictions, such as user rights or locations awareness, throughout conditionals added to the presentation components. The problem with this approach is that markup languages have limitations in separating layout from the presentation. But also presentation cannot be separated from field binding and property settings. The mainstream approach results with 1658 LOC of Java and 13072 LOC of XML presentation, which includes 240 UI conditionals and 6768 restated information from the data-model. Consider that with this approach, developers follow the implementation in Figure 2 (b).

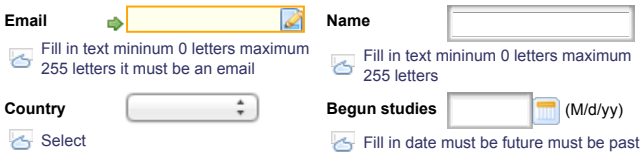


Figure 6. Sample form for confused student

Figure 7. Sample form for child

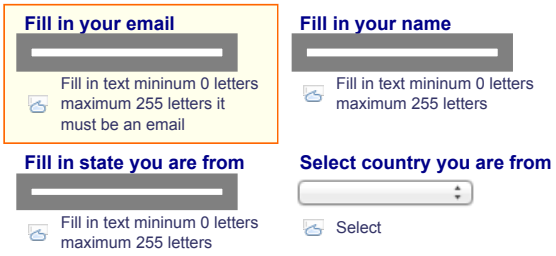


Figure 8. Sample form for elderly

The READ approach allows designers separation of presentation, layout and also of security and location-awareness through various stages within the framework. One of main differences in our approach is that each concern is implemented separately as demonstrated in Figure 2 (a). The READ weaver combines these together. In our study, the application backend Java code includes 1907 LOC, the presentation XML reduces to 5036 LOC, including the presentation and layout templates, and 649 LOC of configuration XML. Conditionals for location and user-right restrictions are captured in the data-model, which reduces them to 20. Furthermore there are no occurrences of restated information in the XML, although field names can be explicitly captured as attributes the UI component Listing 2 to ignoring specified fields. The overall summary of the evaluation is provided in Table 4. Consider that in this second example, individual concerns multiply and their combinations apply. Standard approaches fail to effectively design reusable UI components. The reason is behind the common approaches that fail to capture individual concerns separately, which worsen the code readability, reuse and maintenance. Untangling individual concerns through the AOP approach addresses code readability, reuse and maintenance more effectively.

Next, we evaluate basic maintenance scenarios. With manual development, the UI is fragile because of its coupling to the data-model in the environment with type safety. Changes to a data field, its name or constraints causes inconsistency in all its UI fragments. Such a simple change may lead to 12 locations that need to reflect the change. In type-safe code, this can be easily refactored, but in XML it must be addressed by text search. With READ approach, there are fewer UI references to the data elements; thus it does not require many UI corrections. When we want to globally

Table 5. Performance comparison

	Avg. page load time	Std. deviation
Manual approach	545ms	47
READ approach	539ms	41

change the presentation of a particular widget, in the manual approach all widget occurrences must change; however, with READ such change takes place solely in a template. Changes to user rights manually require to application of conditionals in the UI or at controllers. Since multiple presentations exist for a single field, this can impact a significant amount of UI code. In READ, such change takes place at controllers and then only one time in the data-model, in a single location. The addition of a new form layout may require a new copy of the form with tangled layout, in the common approach. In READ, the layout is a separate fragment, thus only a new layout template is designed.

For the performance evaluation, we consider 5 forms with total of 21 fields. We evaluate the time needed for the page to render using both the manual and READ approach as shown in Table 5. The load times for a page containing the forms, averaged over 250 samples were 545ms (std. dev. 47) for manual approach and 539ms (std. dev. 41) for READ. The measurement shows that the page load time is similar for both approaches.

5.2 Case Study : Production Experience

In order to demonstrate a large scenario, we provide a study that applies the READ framework in production use. The entire user registration and contest management system described in Section 5 is used. The goal of this study is to show applicability of READ to production environment, its impact on development and maintenance, statistics resulting from the approach, and generalization of its impact.

A subset of this system is evaluated in Section 5.1, using a prototype applying various adaptive features. In the production system we only consider single presentation and multiple screen layouts. The entire application is complex and builds on a large data-model (70 data entities). UI development and maintenance makes up a significant portion of the overall development effort. Recently, the application migrated to a new version that includes changes of the presentation framework. Thus we changed all the UI components in the entire application. Since both versions apply READ for the UI forms, only a few changes were required to support new form components. For the entire application, only 25 integration templates existed; these were reused for all forms in the application. Changes to support new form widgets took place in these templates. There were no changes needed for the other concerns (e.g., layout templates, or transformation rules). This migration was done in a very short time, compare to what it would be in the manual case. If had used the manual approach, each form would combine multiple concerns and thus the change would impact up to 21451 LOC of XHTML. Instead with READ, we could solely focus on a single concern (a presentation), which is a change in the UI templates, and this impacts only 288 LOC. While porting forms required little time and effort, the migration of the UI tables, which did not apply our READ approach, took considerably longer because it entangled multiple concerns that were reused in many locations.

Table 6. Case study summary

Application	Java	77394 LOC	
	XML	2380 LOC	
	XHTML	41473 LOC	
	Generated UI	equiv. to 21451 LOC (XHTML)	
Estimate	Savings on restated inf. in UI		15592
UI	Data entities	63 (70 total in the application)	
	Data fields	473	
Average	Entity	7.5 fields	
	Entity in UI	82.5 restated inf.	per UI form
	Entity in UI	113 LOC	and layout
	Field in UI	15 LOC	

Our code measurement in the production system provides the following results. Out of the 70 entities in the data-model, 63 of them are referenced in the UI as forms. All of these forms are generated at runtime based on data inspection. These forms are rendered in three different layout widths according to the user's screen size. In order to apply the READ approach, we must define 28 transformation rules (only 101 LOC), integration templates for UI components (288 LOC) and layout templates (367 LOC). We also need to apply additional 545 annotations to the Java classes. The view part of the application, including XHTML and XML, consists of 41473 LOC and 2380 LOC. The entire Java code has 77394 LOC. The approach brings the reduction in UI forms for 63 entities in three different layouts, which represents up to 21451 LOC of XHTML code. This represents approximately 32% of the entire UI code for the application.

The following is the summary on these measurements. There are 63 entities, with total 473 fields, that are represented in the UI. Each field may have multiple constraints defined by field annotations for object-relational mapping, validation, security or presentation (see Listing 3). This represents 9-13 references per field in the UI component (see the Listing 5); the exact number depends on a particular widget type and the field. We also measure the average number of fields in the UI form per a given entity. The result shows that our system has the mean value 7.5 fields per entity (median 6) with a standard deviation of 4.85. When counting that an UI widget has approximately 11 references to the data-model, it results in 82.5 occurrences of restated information in the XHTML per the average data entity in a single layout form. Consequently, READ prevented an estimated 15592 occurrences of restated information in the application UI. The measured statistics to render data entity in a UI form in a single layout results in 113 LOC with standard deviation 63.77. This is caused by the deviation of class fields. Thus each time we use READ in the UI we save around 113 LOC. On average this represents 15 LOC for an individual field in UI for a single layout.

The measurements are summarized in Table 6; it shows that the use of READ framework reduces the amount of UI code. Significant portion of the UI code is generated. Doing this manually require us to handle significant coupling and many occurrences of restated information; therefore, future evolution management would result in high maintenance efforts.

Table 6 gives our estimate on an average entity its field count, UI references, and LOC required for UI presentation with contemporary approach. The project statistics shows that the UI part of the system is significant, which confirms the estimate from [14]. Regarding the performance, there is no performance reduction exhibited and the community has reported no performance issue. The broader variety of adaptivity of the system to the international audience is currently under development.

6. CONCLUSION

Despite many benefits of CUIs, few production systems support employing them. The reasons behind this include the excessive cost of CUI development and maintenance as shown in our case study. We provide an approach that considers existing standards for application frameworks, aspect-oriented programming and employs code-inspection to face the complexity and efforts related to CUI design. Our READ technique considerably reduces the cost involved in the development of CUIs. While time-consuming to initially develop, all the work related to templates and transformation rules amortizes over the size of the software application. Our approach is implemented in a production-level library, called AspectFaces. It is currently used in production at the ACM-ICPC system.

In the future, we plan to generalize our approach to inspect and reuse application business rules. Our preliminary results show that such an approach will provide more options and variety of adaptivity and further code-reduction for business rules-aware UI.

7. ACKNOWLEDGMENTS

This research was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS12/ 147/OHK3/2T/13.

8. REFERENCES

[1] Java Unified Expression Language, Aug. 2013. http://juel.sourceforge.net.

[2] E. Bernard. JSR 303: Bean validation, Nov. 2009.

[3] A. Blouin, B. Morin, O. Beaudoux, G. Nain, P. Albers, and J.-M. Jézéquel. Combining aspect-oriented modeling with property-based reasoning to improve user interface adaptation. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '11, pages 85–94, New York, NY, USA, 2011. ACM.

[4] M. Blumendorf, G. Lehmann, and S. Albayrak. Bridging models and systems at runtime to build adaptive user interfaces. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '10, pages 9–18, New York, NY, USA, 2010. ACM.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[6] K. Cemus and T. Cerny. Aspect-driven design of information systems. In *SOFSEM 2014: Theory and Practice of Computer Science*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Novy Smokovec, High Tatras, Slovakia, 25, January 2014.

[7] T. Cerny, V. Chalupa, and M. Donahoo. Towards smart user interface design. In *Information Science and Applications (ICISA), 2012 International Conference on*, pages 1 –6, may 2012.

[8] T. Cerny and E. Song. Model-driven Rich Form Generation. *Information: An International Interdisciplinary Journal*, 15(7, SI):2695–2714, JUL 2012.

[9] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[10] L. DeMichiel. JSR 317: JavaTM persistence API, version 2.0, November 2009.

[11] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, Inc., Oct. 1976.

[12] I. R. Forman and N. Forman. *Java Reflection in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.

[13] M. Karu. A textual domain specific language for user interface modelling. In T. Sobh and K. Elleithy, editors, *Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering*, volume 151 of *Lecture Notes in Electrical Engineering*, pages 985–996. Springer New York, 2013.

[14] R. Kennard and J. Leaney. Towards a general purpose architecture for ui generation. *Journal of Systems and Software*, 83(10):1896 – 1906, 2010.

[15] R. Kennard and S. Robert. Application of software mining to automatic user interface generation. In *SoMeT'08*, pages 244–254, 2008.

[16] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar. Aspect-oriented programming. In *In ECOOP'97-Object-Oriented Programming, 11th European Conference*, volume 1241, pages 220–242. Springer, June 1997.

[17] A. G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[18] R. Laddad. *AspectJ in Action: Enterprise AOP with Spring Applications*. Manning Publications Co., Greenwich, CT, USA, 2nd edition, 2009.

[19] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero. USIXML: A Language Supporting Multi-path Development of User Interfaces Engineering Human Computer Interaction and Interactive Systems. volume 3425 of *Lecture Notes in Computer Science*, chapter 12, pages 134–135. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2005.

[20] V. López-Jaquero, F. Montero, and F. Real. Designing user interface adaptation rules with t: Xml. In *Proceedings of the 14th international conference on*

Intelligent user interfaces, IUI '09, pages 383–388, New York, NY, USA, 2009. ACM.

[21] M. Macik, T. Cerny, J. Basek, and P. Slavik. Platform-aware rich-form generation for adaptive systems through code-inspection. In *Human Factors in Computing and Informatics*, pages 768–784. Springer Berlin Heidelberg, 2013.

[22] M. Macik, M. Klima, and P. Slavik. Ui generation for data visualisation in heterogenous environment. In *Proceedings of the 7th international conference on Advances in visual computing - Volume Part II*, ISVC'11, pages 647–658, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, Dec. 2005.

[24] G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520, Aug. 2004.

[25] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@ run.time to support dynamic adaptation. *Computer*, 42(10):44–51, Oct. 2009.

[26] J.-M. Oh, Y. S. Lee, and N. Moon. Towards cultural user interface generator principles. In *Proceedings of the 2011 Fifth FTRA International Conference on Multimedia and Ubiquitous Engineering*, MUE '11, pages 143–148, Washington, DC, USA, 2011. IEEE Computer Society.

[27] M. Schlee and J. Vanderdonckt. Generative programming of graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '04, pages 403–406, New York, NY, USA, 2004. ACM.

[28] V. Schwartze, S. Feuerstack, and S. Albayrak. Behavior-sensitive user interfaces for smart environments. In *Proceedings of the 2nd International Conference on Digital Human Modeling: Held as Part of HCI International 2009*, ICDHM '09, pages 305–314, Berlin, Heidelberg, 2009. Springer-Verlag.

[29] J.-S. Sottet, G. Calvary, J. Coutaz, and J.-M. Favre. A model-driven engineering approach for the usability of plastic user interfaces. In *Engineering Interactive Systems*, pages 140–157. Springer, 2008.

[30] J.-S. Sottet, G. Calvary, and J.-M. Favre. Models at runtime for sustaining user interface plasticity. In *Models@ run. time workshop (in conjunction with MoDELS/UML 2006 conference)*, 2006.

[31] C. Stephanidis, A. Paramythis, D. Akoumianakis, and M. Sfyrakis. Self-adapting web-based systems: Towards universal accessibility. In Waern, editor, *In 4th ERCIM Workshop on "User Interfaces for All"*, 1998.

[32] M. Stoerzer and S. Hanenberg. A classification of pointcut language constructs. In *Workshop on Software-engineering Properties of Languages and Aspect Technologies (SPLAT) held in conjunction with AOSD*, 2005.

ABOUT THE AUTHORS:

Tomas Cerny received his Bachelor's and Master's degrees from the Faculty of Electrical Engineering at the Czech Technical University in Prague, and M.S. degree from Baylor University. He is a Ph.D. student in Prague. His area of research is software engineering, aspect-driven development, user interface design, enterprise application development and networking.

Michael “Jeff” Donahoo received his B.S. and M.S. degrees from Baylor University, and Ph.D. in the Computer Science from Georgia Institute of Technology. Jeff is currently a Professor of Computer Science at Baylor University where he conducts research on networking, security, and enterprise application development.

Eunjee Song is currently an Associate Professor in the Department of Computer Science at Baylor University. Her main research interests include applying aspect-oriented modeling (AOM) and model-driven engineering techniques to specifying and analyzing complex systems. She received her Ph.D. and M.S. in Computer Science degrees from Colorado State University in 2007 and 2001 respectively. Prior to that, she worked for IBM Korea for more than five years after receiving her B.S. in Computer Engineering and B.S. in Architecture degrees from Seoul National University in Korea.

Karel Cemus is a Ph.D. student at Faculty of Electrical Engineering of Czech Technical University in Prague, where he also received his Bachelor's and Master's degree. His research focuses on enterprise information systems, their adaptivity, efficient design and maintenance.

In Situ Affect Detection in Mobile Devices: A Multimodal Approach for Advertisement Using Social Network

Mohammad Adibuzzaman, Niharika Jain
Math, Statistics and Computer Science
Marquette University
Milwaukee, WI, USA
{mohammad.adibuzzaman, niharika.jain}@marquette.edu

Nicholas Steinhafel
Computer Science
Marquette University
nsteinhafel@gmail.com

Munir Haque
Computer and Information Sciences
University of Alabama
Birmingham, Alabama, USA
mhaque@uab.edu

Ferdaus Ahmed
Math, Statistics and Computer Science
Marquette University
Milwaukee, WI, USA
ferdaus.ahmed@gmail.com

Sheikh Ahamed
Math, Statistics and Computer Science
Marquette University
Milwaukee, WI, USA
iq@mscs.mu.edu

Richard Love
International Breast Cancer Research Foundation
Madison, WI, USA
richard@ibcrf.org

ABSTRACT

Affect detection has been widely advocated to be implemented in a natural environment. But due to constraints such as correct labeling and lack of usable sensors in natural environment most of the research in multi-modal affect detection has been done in laboratory environment. In this paper, we investigate affect detection in natural environment using sensors available in smart phones. We use facial expression and energy expenditure of a person to classify a person's affective state by continuously recording accelerometer data for energy and camera image for facial expression and measure the performance of the system. We have deployed our system in a natural environment and have provided special attention on annotation for the training data to validate the 'ground truth'. We have found important relationship between valence and arousal space for better accuracy of affect detection by using facial image and energy. This validates Russell's two dimensional theory of emotion using arousal and valence space. In this paper, we have presented initial findings in multi-modal affect detection. Using the multimodal technique, we propose a system that can be used in social networks for affect sensitive advertisement. [1]

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine System; I.4 [**Image Processing and Computer Vision**]: Feature Measurement

General Terms

Performance

Keywords

Algorithms, Experimentation, Human Factors

1. INTRODUCTION

Affective computing finds its application in several domains such as advertising, robotics, health care, Human Computer Interaction (HCI) and gaming [11]. The prior works in affective computing follow a unimodal approach for affect detection, wherein a single feature is used for affect recognition. This single feature could correspond to facial expressions, speech, body gestures or physiological features like heart rate, skin conductivity, blood pressure etc.

Human emotion is a multi-modal phenomenon. The emotional or affective states such as happiness, sorrow, fear etc. experienced by humans are usually characterized by changes in more than one feature. For example, an expression of happiness can be understood from facial expressions and speech, or, anger can be evident from speech, body movement or facial expressions. For an affect recognition system to be more natural, reliable and accurate, using a multimodal approach is better than a unimodal one. In the field of affective computing, multimodal real time implementation is widely advocated but rarely implemented [11]. Research has been done for affect detection from facial expression, speech data, body gesture, heart rate, skin conductance, pressure sensor and other inputs.

Facial expressions can be decoded by several means. One of the most popular approaches, FACS (Facial Action Coding System) proposed by Ekman [24], uses Action Units (AUs) to recognize the facial expression. The focus of region based approaches is on specific regions of the face like eyebrows, eyes and lips and hence becomes a region specific approach. Facial features can also be recognized using pattern recognition approaches [17]. There are also works for facial feature extraction using geometric features and appearance features [28].

Several research studies have demonstrated the use of machine learning techniques in understanding the patterns associated with physiological activity in order to detect affect [29]. Fusion of the data obtained from facial expressions and speech has been done to attain benefits of multimodal approach. In another study, motion and audio data from mobile-device was used to assess the mental and physical well-being and a high correlation was found with gold-standard survey metrics [20]. To the best of our knowledge, we

have found very few studies which include physical activity as one of the modalities in affect detection [12].

Although, research using multimodal approach for affect detection has gained popularity in the last decade [29], the use of hand-held devices with in-built sensors in collecting information has been rare. In a laboratory setting, a person is usually equipped with several devices or medical equipment to collect relevant data. The use of a single hand-held device, equipped with in-built sensors can negate such need. Smartphone is one such example which has in-built sensors such as camera, microphone, GPS and accelerometer. These sensors can help collect useful information in relation to affective states like facial expressions (from camera), speech (from microphone), location (from GPS) etc. Likewise, accelerometer data can be used to estimate the energy exerted by a person over a period of time. It can be realized that the power of such hand-held devices knows no bounds. Just like facial features, information about energy expenditure can also give us affective cues. The presence of camera and accelerometer in the smartphones can hence play a vital role in affect detection using mobile or hand-held devices.

For multimodal affect detection, we used Naïve Bayes fusion technique to integrate the data from camera (facial expressions) and accelerometer (physical activity). We evaluated and compared the system performance for unimodal (only facial expression data) and multimodal approaches for affect detection. It was found that the multimodal system was significantly better than the unimodal system.

We also build a Facebook application for detecting affective state and which can be used for advertisement. The Facebook application uses images of a person collected from smart phone and uses eigenface for training and detection of facial expressions into six basic emotional categories: anger, fear, happiness, disgust, sadness and surprise.

1.1 Contributions

Decision making capabilities in humans are largely governed by their emotions [3]. Hence, to understand human behavior, it becomes important to gain knowledge about affective states and their recognition. In this paper, we present a system which uses the in-built sensors of a smartphone to give information about affective state of the user. The contributions of this research work are as follows:

- We provide a natural setting for data collection without interfering in daily routine of participants
- We claim that a multimodal (facial expressions and energy expenditure) design provides more accuracy in affect recognition as compared to unimodal (only facial expressions) design.
- The affect detection technology can be used for advertisement in social network. We propose such a system and build a prototype for affect aware advertisement.

2. STATE OF THE ART

Even though there has been lot of work in the area of affective computing, it suffers from challenges and limitations [19]. Expression of emotion varies from person to person and is expressed in numerous ways. It is difficult to incorporate each and every parameter in data collection and analyses. Moreover, the data is usually collected in a laboratory environment which introduces the bias of observation for the emotional expression. These lab settings are devoid from the information corresponding to participantâĂŹs environment, nearby objects, location etc. which otherwise play a crucial role in his/her affective state.

In an attempt to overcome some of these challenges, the data in this study[8] was collected through several means. It included a mobile phone journaling system, wireless sensors for measuring galvanic skin response, heart rate and physical activity. These 'in situ' ratings were combined with contextual ratings provided by third-party raters to increase the affect recognition rates. The data collected from these different sources were then triangulated so as to achieve a data set containing mutually agreeing information. This information when fed to a J48 decision tree returned highly accurate results (100% accuracy) in terms of high or low activation states.

For multimodal affect detection in natural settings, 'ground truth data' were collected through self-reports, and audio and physiological data were collected through sensors [9]. Self-report helped in the development of inference algorithms. It was observed that having a customizable time window is important because of the usual delay in annotating emotional experience. Having this feature definitely improved the accuracy of the affect recognition system. It was also realized that using wireless sensors can be challenging because of positioning and connectivity issues.

The power of mobile devices has been further explored for affect annotation by incorporating the multi-modal technique for assessing physical as well as mental well-being [20]. A study was conducted wherein the subjects were given a device consisting of various sensors which could help collecting data required for the above mentioned assessment. The classification of collected data as speech and activity was done using two-state hidden Markov Models (HMM) and decision-stump classifiers respectively. A correlation between automated assessment of mental, and/or physical health and the result of gold-standard surveys was found so as to stress upon the accuracy of sensor-based measurements.

Healey et al. tries to standardize the affective data annotation in [8] and [9]. But their approach used sensors not only that are available in mobile devices, but also external sensors. Again, a comprehensive study about the classification algorithm was not present. [20] shows the correlation of gold standard surveys with sensor data capture, but that does not provide a study only for mobile device. The participants had to wear other sensor devices for affective data annotation. In our study we overcome both of these problem by using only one smart phone for the user. We also present a comparative study of the result about multimodal versus unimodal system. We also summarize a list of related works that use different modalities for affect detection in Table 1. All of these works discussed in this section provide the techniques and results of using different modalities. But none of the research study uses only smart phones for collecting data. Also, a comparative study between unimodal system versus multimodal system was not present.

3. OUR APPROACH

To capture the arousal and valence space, we used facial expression and energy exertion of a person. From the field of psychology,

Table 1. Summary of related works for different modalities

Multimodalities	Features
Facial Features (Eye, nose and mouth) [2]	- Skin Detection Algorithm for identifying the skin pixels - Z-based erosion and standard erosion algorithms for refining the pixels - Range imagery to locate eyes, mouth and nose - Geometric-based confidence measure to select the best feature group
Heart activity and facial expressions [10]	- Augsburg Matlab toolbox for extracting features from ECG signals - eMotion software for extracting features from facial expressions - Chi-square feature selection algorithm for selecting equal features from each channel - Feature-level fusion to append the selected features KNN, SVM and Decision tree algorithms for classification Vote classifier for average probability
Conversational cues, body language and facial features [4]	- F-ratio from a univariate ANOVA to select equal features - Feature-level fusion to merge the features obtained from each channel - Linear discriminant analyses to classify the data
Facial features and speech [27]	- Nearest neighbor method to classify video data - HMM to classify audio data - Rule-based algorithm for combining the information

arousal space can be captured by heart rate, pupil size or energy expenditure; all of them can be captured from the sensors available in smart phones. In this paper, we used facial expression for valence and energy expenditure for the arousal space. We used eigenface algorithm for detecting facial expression and later used mean of different affective states as the second feature for multimodal affect detection using Naïve Bayes fusion. In this section we describe the methods we used for the annotation of in situ affective data and the reason for using facial expression and energy expenditure. In the next section, we describe the details of our classifier.

3.1 Selecting Modalities

Emotion labeling is moderately less work in laboratory environment where the researcher can control the environment, recreate the situation, recording can be done accurately and the person can be interviewed for his/her annotations. However, the emotion labeling can still be flawed since in controlled environment people might act differently, both physiologically and cognitively. In situ capturing of affective state captures the natural data, but it needs more methodical approach for emotion journaling. The participants need to be trained well and the data labeled needs to be verified later. Our goal is to minimize the error for establishing the 'ground truth', which defines true affective state for the given data in machine learning algorithm for classification.

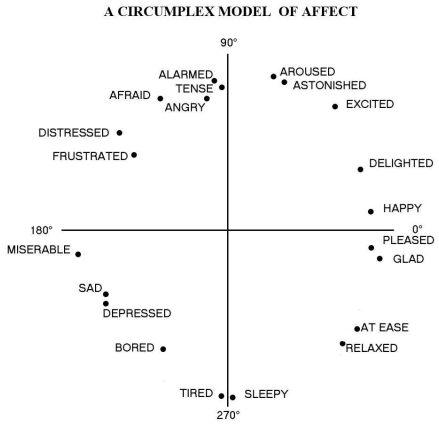


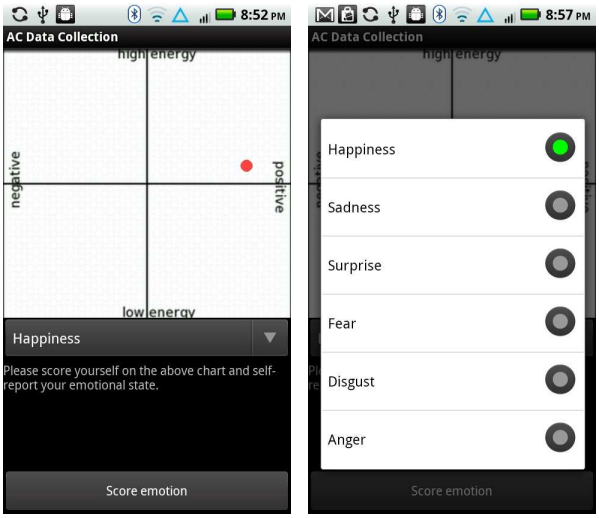
Figure 1. Russell's circumplex model of emotion [23].

According to the Russell's circumplex model of emotion, each affective state can be represented in 2D space [23]. The horizontal axis represents the valence and the vertical axis represents the arousal space. Valence represents how good or bad a person is feeling, and arousal represents how much a person is aroused. Therefore, we hypothesize that if we could capture the arousal space data from accelerometer, we could better classify the affective states. For example, for the happy state, this is a positive feeling and a person might have some kind of excitement. On the other hand, for sad feeling, it is a negative feeling and the person may have less movement, which corresponds to less energy expenditure.

3.2 Emotion Journaling

We used smart phones for emotion journaling. Smart phones give us the opportunity for labeling emotion as soon as it occurs with real time sending and storage capability. Eight participants were recruited for the study, aged between 21-33 all of whom are students. Participants were asked to carry the smart phones and annotate the data for at least 5 times a day for a seven day period. The participants will be referred as PA in this work.

For the journaling of emotional data, we used camera and accelerometer data of smart phones for facial expression and activity. Also location data was stored using GPS of the smart phones that might give us the context information. Three android phones were used, two Droid X with android operating system of 2.2 and one Samsung Galaxy Nexus with android operating system of 3. PAs were asked to take a facial picture with the smart phones and then label the data. The labeling was done using two sources for capturing the natural feeling. One is using the *Mood-map* which corresponds to Russell's circumplex model and the other is radio button from



(a) (b)

Figure 2. Annotation of emotion data: (a) Annotation of affective state using Russell's 2D emotional space. (b) Annotation of affective state using radio button.

which the user can pick one from the six basic emotions. Figure 2 shows the interface for mood-map as well as the radio buttons. Continuous and fine grained accelerometer data for fifteen minutes before taking the picture and location data were also recorded and then sent to the server using the phones' internet.

3.3 Journaling Training

Each participant was asked to keep the smart phone for one week. Before handing over the smart phone, they were trained on how to use the application for emotional data annotation. During that period, PAs were asked to carry the smart phones six to eight hours a day and label the data whenever any emotional event occurred. They were trained to use the touch based application as well as how to take the picture, use the mood-map and upload the data. Furthermore, there was constant communication between the PAs and the researcher for any question from the participant.

3.4 Algorithm Design

The algorithm for affect detection from facial expression and accelerometer data can be discussed in different components;face detection, affective state from facial image, energy expenditure from body movement and fusion using Naïve Bayes. Each of the components are discussed here.

3.4.1 Face Detection

Pixels corresponding to skin are different from other pixels in an image. [17] has shown the clustering of skin pixels in a specific region for Skin color modelling in chromatic color space. Though the skin color of persons vary widely based on different ethnicity, research [24] shows that the still form a cluster in the chromatic color space. After taking the image of the subject we first crop the image and take only the head portion of the image. Then we use skin color modeling for extracting the required facial portion from the head image.

3.4.2 Affective State from Facial Image

For this part we use a combination of Eigenfaces, Eigeneyes, and Eigenlips methods based on Principal Component Analysis (PCA) [28][29]. This analysis method includes only the characteristic features of the face corresponding to a specific facial expression and leaves other features. This strategy reduces the amount of training sample and helps us make our system computationally inexpensive which is one of our prime goals. These resultant images are used as samples for training Eigenfaces method and M Eigenfaces with highest Eigenvalues. We generate the Eigenspace as follows:

- The first step is to obtain a set S with M face images. Each image is transformed into a vector of size N^2 and placed into the set, $S = \gamma_1, \gamma_2, \gamma_3, ..., \gamma_M$
- Second step is to obtain the mean image ψ

$$\psi = \frac{1}{M}\sum_{n=1}^{M}\gamma_n$$

- We find the difference ψ between the input image ϕ and the mean image,$\phi_i = \gamma_i - \psi$
- Next we seek a set of M orthonormal vectors, μ_M, which best describes the distribution of the data. The k^{th} vector, μ_k, is chosen such that

$$\psi = \frac{1}{M}\sum_{n=1}^{M}(\mu_k^T\phi_n)^2$$

- λ_k is a maximum, subject to

$$\mu_l^T\mu_k = \begin{cases} 1, & \text{if } l == k. \\ 0, & \text{otherwise.} \end{cases}$$

  where μ_k and λ_k are the eigenvalues and eigenvectors of the covariance matrix C.

- The covariance matrix C has been obtained in the following manner

$$\psi = \frac{1}{M}\sum_{n=1}^{M}(\phi_n\phi_n^T)^2 = AA^T$$

  where $A = [\phi_1, \phi_2, \phi_3, ..., \phi_m]$.

- To find eigenvectors from the covariance matrix is a huge computational task. Since $M$is far less than N^2 by N^2, we can construct the M by M matrix,

$$L = A^TA$$

  where $L_{mn} = \phi_m^T\phi_n$

- We find the M Eigenvectors, v_l of L.These vectors (v_l) determine linear combinations of the M training set face images to form the Eigenfaces u_l.

$$\mu_l = \sum_{k=1}^{M} v_{lk}\phi_k$$

  where $l = 1, 2, 3, ..., M$

- After computing the Eigenvectors and Eigenvalues on the covariance matrix of the training images

  - M eigenvectors are sorted by Eigenvalues
  - Top eigenvectors represent Eigenspace

- Project each of the original images into Eigenspace to find a vector of weights representing the contribution of each Eigenface to the reconstruction of the given image.

When detecting a new face, the facial image is projected in the Eigenspace and the Euclidian distance between the new face and all the faces in the Eigenspace is measured. The face that represents the closest distance will be considered as a match for the new image. Similar process is followed for Eigenlips and Eigeneyes methods. The mathematical steps are as follows:

- Any new image is projected into Eigenspace and we find the face-key by
  $\omega_k = \mu_k^T$ and $\omega^T = [\omega_1, \omega_2, \omega_3, ..., \omega_M]$
  where, u_k is the k^{th} eigenvector and ω_k is the k^{th} weight in the weight vector $\omega^T = [\omega_1, \omega_2, \omega_3, ..., \omega_M]$

- The M weights represent the contribution of each respective Eigenfaces. The vector Ω, is taken as the 'face-key' for a face's image projected into Eigenspace.

- We compare any two 'face-keys' by a simple Euclidean distance measure
$$\epsilon = ||\Omega_a - \Omega_b||^2$$

- An acceptance (the two face images match) or rejection (the two images do not match) is determined by applying a threshold.

3.4.3 Energy Expenditure from Body Movement

There exists a significant correlation between accelerometer data and the work done by a person. It is found that the energy measured by ADInstrument Exercise Phsyiology Kit is highly correlated with accelerometer energy when the phone is positioned at the waist [5].

Droid X uses the STMicroelectronics LIS331DL accelerometer. In our study, 2 Droid X 3G devices running Android OS 2.2 and one Samsung Galaxy Nexus with Android OS 3 were used as acceleration measurement platforms.

Since this is a piezo-resistive accelerometer, low pass filtering is required to acquire the true activity-component. We applied low-pass filtering on the raw accelerometer data, as its output includes a DC gravitational contribution. In the literature, the ideal cut-off frequency or the filter ranges from 0.1 Hz to 0.5 Hz. We used 0.5 Hz filter in Matlab to exclude the gravitational contribution. After testing the varying frequency in this range, we found good result preserving the activity contribution.
To correlate accelerometer data with energy expenditure of a person, the accelerometer's three dimensional vector needs to be summarized as one scalar value that represents physical activity intensity over small time periods [5]. This scalar value is considered accelerometer energy spent by the user. To calculate accelerometer energy, several different methods have been proposed, but the most used one is the summation of time integrals of accelerometer output over the three spatial axes [5]. We adopted this method. The accelerometer energy is calculated according to the following formula:
Accelerometer energy $= \int_{t_0}^{t_0+T} |a_x| + |a_y| + |a_z| dt$

Here a_x,a_y and a_z are low-pass filtered accelerometer data corresponding to the x, y, and z axes. For calculating the values of this equation, we found the accelerometer input data on each of the axes. Then low pass filtering was used on each axis input. Next, we calculated the absolute value of the accelerometer inputs and found the integration during fifteen minutes time before taking the user image.

3.4.4 Fusion Using Naïve Bayes

We found the mean of the energy data for different affective states and those means were used as a separate feature for the fusion. Table 2 summarizes the mean of the energy for different affective states. Those means were used as the additional feature for our fusion.

Table 2. Mean of energy for different affective states

Affective State	Energy(mean)
Anger	8.56E+00
Disgust	2.24E+01
Fear	4.12E+01
Happy	1.51E+01
Sad	4.28E+00
Surprise	3.56E+01

It is argued that human behaviour is close to that predicted by Bayesian decision theory [13]. Different probabilistic graphical model algorithms are used in the literature like Hidden Markov Model (HMM) and Support Vector Machine(SVM).

In our fusion, we used Bayesian classifier. Since we are working only on two modalities, we argue Naïve Bayes algorithm would be a better fit, which performs better with small number of features and potentially large data for fusion . Fusing the modalities of facial expression and energy data at decision level enables us to gain the knowledge about the relationship between these two modalities for a particular affective state [14].
The Bayesian fusion framework that we apply is proposed in [25]. It uses the conditional error distributions of each classifier to approximate uncertainty about that classifier's decision. The combined decision is the weighted sum of the individual decisions. Given a problem with K classes and C different classifiers, λ_i, $i = 1, ..., C$ we like to infer the true class label ω, given the observation x. Assuming that for each classifier λ_i we have a predicted class label ω_k, where $k = 1, ..., K$ then the true class label can be derived as follows:
$$P(\omega|x) \approx P(\omega|\omega_k, \lambda_i)P(\omega_k|\lambda_i, x)P(\lambda_i|x)$$
Probabilities $P(\omega|\omega_k, \lambda_i$ and $P(\lambda_i|x)$ are used to weight the combined decision and can be approximated from the confusion matrix of classifier λ_i.

We used the energy expenditure data of the same persons from our facial expression database. When the users simulated affective state, their energy data was also collected for the last 15 minutes before taking the photograph.

4. EVALUATION

We evaluated the system in four different ways. Validating the ground truth, performance of unimodal system with only facial expression,validating energy data, and performance of the multimodal system.

4.1 Validating The Ground truth

After the data collection, each day the participants were interviewed and asked about their labeling. We found that some data were not properly labeled. Due to ambiguity of the context, some data were also discarded. For example, on one occasion PA2 said, *'I was feeling very good with my grade, but did not have much movement since I was sitting on my desk. So I labeled the emotion as positive in valence but negative in arousal and did not know which one to pick from radio button. So I selected sad.*' We only incorporated the data that the researcher and the PA we agreed on to be of any particular affective state.

4.2 Unimodal System With Facial Expression

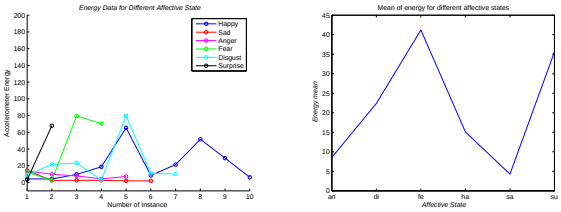
We trained our database with the pictures taken by the camera of the smart phones. Then for each image in the training database, we used our classifier for facial expression and found 89% accuracy. The confusion matrix for facial expression is given in Table 3. We found that the pictures taken by the camera for which the environment was dark, the system gave inaccurate results and the image was not properly classified. We got one inaccurate results for each of the expression anger, sad, disgust, fear and surprise. From the result, we conclude that the classifier works well with the training database as long as the image is taken properly with proper lighting. It does not depend on any particular expression.

Table 3. Confusion matrix for facial expression classifier

a	b	c	d	e	f	←Classified as
8	0	0	0	0	0	| a=happy
0	7	0	0	1	0	| b=anger
1	0	7	0	0	0	| c=sad
1	0	0	7	0	0	| d=disgust
1	0	0	0	7	0	| e=fear
1	0	0	0	0	7	| f=surprise

4.3 Validating Energy Data

We used the mean of the energy data for different affective states as the second feature for our Naïve Bayes fusion. We found an interesting relationship between the energy and the different categories. Figure 3(a) shows the energy mean for different annotations by different PAs. Each point represents a particular annotation by any PA. It was difficult to visually distinguish the energy for the different categories. However, for the three categories, namely happy,



(a) (b)

Figure 3. Accelerometer Energy for Different Affective State: (a) Accelerometer energy for six basic emotions. (b) Mean of energy for different affective state.

sad and anger; we found an important relationship. The mean of energy of sad is much lower than the mean of the energy of happy and that of anger. On the other hand, the mean of energy of anger is not very high where in Russell's two dimensional space it is considered higher than the happy state. We conclude, from our data that happiness usually has a high energy expenditure relative to sadness, which is in line with Russell's theory.

This relationship is best shown in Figure 3(b), where we plot only the mean for different affective states. The horizontal axis represents different emotional states and the vertical axis represents the energy mean for the corresponding emotion. We find that sadness has much lower mean of energy than that of happiness. Also, fear has high value in arousal space and we found that the mean to be much higher than happy and sad. This is also in line with the Russell's circumplex model where fear is phrased as afraid [Figure 1].

4.4 Performance of Multimodal System

The last part of the discussion addresses the performance of the multimodal system. We see how our system performs with the Naïve Bayes fusion. We find that the system performance for correctly classifying the instances for our training database increases from 89% to 93%. Out of 48 total instances, 45 were classified correctly.

A close analysis from the confusion matrix of the multimodal system from Table 4 gives us the reason of the improvement.

First, the image previously misclassified as fear instead of anger is classified correctly now. The reason is that the mean of energy for anger (8.56E+00) is much lower than that of fear (4.12E+01).As a result, even if the image was not clear enough, it is correctly classified. The same reasoning is also true for the data that were previously misclassified as anger instead of fear.

This data is now also classified correctly. Another interesting observation is the confusion matrix entry for disgust. One disgust entry was misclassified in the unimodal system as anger.
In the multimodal system it still is misclassified, but to a different class, fear. We observe that the mean of energy for disgust is equidistant from both anger and fear. As a result, the system could not find a close match for this annotation.

5. APPLICATION

Table 4. Confusion matrix using Naïve Bayes classifier

a	b	c	d	e	f	←Classified as
8	0	0	0	0	0	| a=happy
0	8	0	0	0	0	| b=anger
1	0	7	0	0	0	| c=sad
0	0	0	7	0	1	| d=disgust
0	0	0	0	8	0	| e=fear
0	0	0	1	0	7	| f=surprise

The application of the affect detection technology in natural everyday settings includes a broad range of area. It could be used for affect sensitive user interface, learning and also for advertisement. We built a prototype for affect sensitive advertisement in Social networking site, Facebook, for a desktop version. Our ultimate goal is to use the sensors available in smart phones and build a prototype for affect sensitive advertisement.

Neuropsychologists have looked at human decision making and found that emotions play a role with simple as well as with complex choices. This is relatively a new view on human decision making.

Greene et. el. used functional Magnetic Resonance Imaging (fMRI) on 100 respondents to show how emotions influence what they call 'Moral Judgements' [6]. Among the 60 moral judgements studied in [7], cases have found such as choice between different coupons to use in a store and the choice between bus and train for travelling depending on different affective states. Also, according to a study from eMarketer (October 2009), one-half of Beresford respondents said they considered information shared on their networks when making a decision. The proportion was higher among users aged between 18 and 24, at 65%. This indicates consumers put great trust in their social networks [1]. For these reasons, we propose an advertisement model for smart phones using social network that would capture the in situ affective state of the user and will show advertisement accordingly. Figure 8 shows a screen shot of the Facebook application.

6. FINDINGS AND DISCUSSION

6.1 Inherent Theory of Emotion is Not Established

The theory of emotion is not established yet. Psychologists have different approaches to identify different emotions. Research in the field of affective computing is about finding the features that are most likely related to emotion-oriented computing. Understanding those ideas and adapting those to any computational methods is still in progress. Furthermore, expression of emotion greatly varies from person to person, man and women, and also among different age groups and races.

Figure 4. Prototype of Facebook application for affect sensitive advertisement.

Paul Ekman has identified six basic emotions for psychologists to identify from video sequence using Facial Action Coding System (FACS). Those are happiness, sadness, disgust, anger, fear and surprise. There are other emotions important for automatic detection of emotions like boredom, frustration, excitement and many more. Even Ekman expanded his list of basic emotions to include other emotions like amusement, contempt, embarrassment, excitement, guilt, satisfaction etc.

There are also different approaches in computing for different theories in psychology. For Ekman's FACS to be implemented; feature extraction is needed from facial image and then it needs to be classified. However, there are different emotions with overlapping Coding Schemes which makes the implementation complicated.

For the holistic approach different machine learning algorithms are used. We have used such an approach.

6.2 Multimodal System Needs More Modalities

In our approach, we have argued that multimodal emotion recognition will contribute to the more accurate affective classification. For that we might have to put different weights for different modalities. Also, in person to person communication, we may or may not look for the same features in multiple channels like facial expression, speech and body movements. More importance might be needed for finding same emotional cues in multiple modalities. Again, this varies a lot among person to person. People tend to understand about others emotion from facial expression, tone, body movement, gestures and most importantly context. Depending upon context, the interpretation of a message could be quite different from another. A combination of low level features, high level reasoning, and natural language processing is likely to provide best multimodal affect recognition. But very few systems have been developed in a natural environment considering multiple modalities. Even if they were developed, their performance is measured in a laboratory environment, which might be quite different than in a natural environment.

6.3 Privacy

We have argued that affect detection is important but that also comes with increasing concerns about privacy awareness of the people. However, this argument can be contrasted with the fact that in our system, detected affective state is shared only by the permission of

the user. Nevertheless, there remains significant scope for research regarding privacy issues and different levels of anonymization techniques to be dealt with.

7. CONCLUSION

In this paper we investigate if the sensors of the smart phone can be used for multimodal affect detection. We also showed an application which can be used for advertisement in a social networking site like facebook. We found that some emotional states are ambiguous and in many cases people have 'mixed feeling'. In many situations even human can not identify the emotional states properly. This is because human might have mixed emotions at a particular time. There are no borders with different emotional states. However, still we emphasized on the labeling of the emotion by the PAs. The success of such systems largely depends on the emotional self awareness of the PAs. We also suggest that because of overlapping conditions of affective states, the classification approach should use probabilistic approaches. Instead of classifying one particular instance to a particular affective state, different probabilities should be assigned for each instance. We also find arousal to be easily captured than valence. One such approach might be using pupil size to capture the information in arousal space, which is an important indication of arousal in psychology. We believe the large sensor data captured by the smart phones can be used for machine interpretation of human affective states and machines can understand part of larger human intelligence. Also, affect sensitive applications should be developed targeting the application scenario. For example, the application for advertisement in smart phones may not be feasible for detecting boredom in a learning environment. With the continuous advancement of sensor technologies in smart phones, we can predict human affective states more accurately and the application of such affect detection technique might be huge.

8. REFERENCES

[1] http://www.bazaarvoice.com/resources/stats.

[2] Boehnen, C., Russ, T. A Fast Multi-Modal Approach to Facial Feature Detection. In *Proc. of the Seventh IEEE Workshop on Applications of Computer Vision*, 2005.

[3] Bradley, M. M., Miccoli, L., Escrig, M. A., Lang, P. J. The pupil as a measure of emotional arousal and autonomic activation. Center for the Study of Emotion and Attention, University of Florida, 2008.

[4] DâĂŹMello, S.K., Graesser, A. Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features. In *User Modeling and User-Adapted Interaction archive*, volume 20, June 2010.

[5] Fujiki, Y. iPhone as a Physical Activity Measurement Platform. In *CHI 2010(Student Research Competition)*, 2010.

[6] Greene, J. D., Sommerville, R.B., Nystrom, L. E., Darley, J. M., Cohen, J. D. . An fMRI Investigation of Emotional Engagement in Moral Judgement. In *Science*, volume 293, September 2001.

[7] Hansen, F., Christensen, S. R. Emotions, Advertising and Consumer Choice. Copenhagen Business School Press, 2007.

[8] Healey, J. Recording Affect in The Field: Towards Methods and Metrics for Improving Ground Truth Labels. In *Proc. of the Affective computing and intelligent interactions*, 2011.

[9] Healey, J., Nachman, L., Subramanian, S., Shahabdeen, J., Morris, M. Out of the Lab and into the Fray: Towards Modeling Emotion in Everyday Life. In *Proc. 8th International Conference on Pervasive Computing*, 2010.

[10] Hussain, M. S., Calvo, R. A. . Multimodal Affect Detection from Physiological and Facial Features during ITS Interaction. In *Proc. 15th International Conference on Artificial Intelligence in Education (AIED)*, June 2011.

[11] James, A., Sebe, N. Multimodal Human Computer Interaction: A Survey. In *Computer Vision and Image Understanding*, volume 108, October 2007.

[12] Kapoor, A., Picard, R. Multimodal Affect recognition in learning environments. pages 677–682, 2005.

[13] Kording, K.P., Wolpert, D.M. Bayesian decision theory in sensorimotor control. In *Trends in Cognitive Sciences*, pages 319–326, 2006.

[14] Metallinou, A., Narayanan, S., Lee, S. Decision Level Combination of Multiple Modalities for recognition and analysis of emotional expression. In *Proc. of the International Conference on Acoustics, Speech, and Signal Processing*, 2010.

[15] Monwar, M., Prkachin, K., Rezaei, S. Eigenimage Based Pain Expression Recognition. In *International Journal of Applied Mathematics*, May 2007.

[16] Nicolaou, A., Pantic, M., Gunes, H. Continuous prediction of spontaneous affect from multiple cues and modalities in valence-arousal space. In *IEEE Transactions On Affective Computing*, volume 2, 2011.

[17] Pantic, M., Roisman, G., Huang, T., Zeng, Z. A survey of Affect Recognition Methods: Audio, Visual and Spontaneous Expressions. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 31, January 2009.

[18] Picard, R. W. Affective Computing. The MIT Press, 1997.

[19] Picard, R. W. Affective Computing: Challenges. In *International Journal of Human-Computer Studies*, 2003.

[20] Rabbi, M., Ali, S., Choudhury, T., Berke, E. Passive and In-situ Assessment of Mental and Physical Well-being using Mobile Sensors. In *Proc. of the 13th international conference on Ubiquitous computing*, 2011.

[21] Rime, B., Mesquita, B., Philipot, P. Long lasting cognitive and social consequences of emotion: Social sharing and Rumination. In *European Review of Social Psychology*, 1992.

[22] Rime, B., Philippot, P., Zech, E., Luminet, O., Finkenauer, C. Social sharing of emotion: New evidence and new questions. In *European review of social psychology*, volume 9, pages 145–189.

[23] Russell, J. A. A circumplex model of affect. In *Journal of Personality and Social Psychology*, volume 39, 1980.

[24] Scherer, K.R., Ekman, P. Methods For Measuring Facial Action. In *Handbook of methods in nonverval behavior research*, pages 45–135. Cambridge University Press, 1982.

[25] Serre, T., Bouvrie, J., Ivanov, Y. . Error weighted Classifier Combination for Multimodal Human Identification. In *Tech. Rep., MIT, Cambridge, MA*, 2005.

[26] Sharma, R., Huang, T., Pavlovic, V. Toward Multimodal Human Computer Interface. In *Proc. IEEE*, volume 86, pages 853–869, 1998.

[27] Silva, L. C. D., Ng, P. C. Bimodal Emotion Recognition. In *Fourth IEEE International Conference on Automatic Face*

and Gesture Recognition, 2000.

[28] Tian, Y., Cohn, J., Kanade, T. Facial Expression Analysis. In *Handbook of Face Recognition*. Springer, 2005.

[29] Tian, Y., Cohn, J., Kanade, T. Affect Detection: An Interdisciplinary Review of Models, Methods, and Their Applications. In *IEEE Transactions on Affective Computing*, volume 1, 2010.

[30] A. E. K. N. Wagner, J. From Physiological Signals to Emotions Implementing and Comparing Selected Methods for Feature Extraction and Classification. In *IEEE Int'l Conf. Multimedia and Expo*, pages 940–943, 2005.

[31] M. Weiser. Some computer science issues in ubiquitous computing. In *Communications of the ACM*, pages 74–83, 1993.

ABOUT THE AUTHORS:

Mohammad Adibuzzaman is a Doctoral Candidate at the department of Math, Statistics and Computer Science at Marquette University, Milwaukee Wisconsin. His PhD thesis is titled 'Smart Monitoring of Health Parameters' for which he is working on algorithm development, system design and mathematical modeling for innovative healthcare solutions. He received his Masters in Computational Sciences in 2012 from the same University. Before coming to Marquette, he worked as a Junior Research Assistant at the Human Computer Interaction Laboratory of National University of Singapore and as a Software Engineer at a software company in Bangladesh. He also worked as an Oak Ridge Institute of Science and Engineering (ORISE) fellow at the U.S. Food and Drug Administration in 2013 by an appointment to the Research Participation Program at the Center for Devices and Radiological Health administered by the Oak Ridge Institute for Science and Education.

Niharika Jain is a Doctoral Student in the Ubicomp Lab at Marquette University. She received her Master's degree in Computational Sciences from Marquette University in 2012. Her interest lies in applying computational approaches in the area of affective computing. Her research is focused on studying anxiety in children with Autism Spectrum Disorders. Before joining Marquette, Niharika was working as a software engineer in one of the leading IT companies in India.

Nick Steinhafel briefly attended Marquette University where he studied Computer Science and Psychology. He has since relocated to California where he has launched several successful companies. Nick is currently employed at a startup focusing on disrupting the health care claim industry.

Dr. Md Munirul Haque is a Postdoctoral Fellow at the Department of Computer and Information Sciences at the University of Alabama at Birmingham. He holds a Ph.D. in Computational Sciences and a M.S. in Computer Science from Marquette University, USA and B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology. Dr. Haque has published 20+ peer reviewed journal, conference and workshop papers. He is the recipient of the Ross Fellowship Award for outstanding Ph.D. student and 4 best paper/poster awards (COMPSAC 2007, CHI 2012). His research interest includes mobile security, m-health, and HCI. He is especially interested in mobile phone based applications to improve the quality of life of people with special needs.

Ferdaus Kawsar is a Doctoral candidate at the department of Mathematics, Statistics and Computer Science at Marquette University, Milwaukee Wisconsin. He received his Master's degree in Computational Sciences from the same university in 2012. His interest includes 'Remote Activity Monitoring' for which he is working on the design and development of the system as well as the development and evaluation of algorithms. Other areas if interest include Time Series Data Analysis, Pervasive Healthcare and Machine Learning. Before joining Marquette, Ferdaus worked as a Lecturer in different universities in Bangladesh.

Sheikh Iqbal Ahamed is a professor and director of Ubicomp Lab in the department of Math, Statistics, and Computer Science at Marquette University, USA. He is also a faculty member of Medical college of Wisconsin, USA. He is a senior member of the IEEE Computer Society and ACM. He completed his Ph.D. in Computer Science from Arizona State University, USA in 2003. His research interests include mHealth, security and privacy in pervasive computing and middleware for ubiquitous/pervasive computing. He has published 100+ peer reviewed journal, conference and workshop papers including seven best paper/posters. Dr. Ahamed serves regularly on international conference program committees in software engineering and pervasive computing such as COMPSAC 13, COMPSAC 12, PERCOM 08, and SAC 08. He is the Guest Editor of Computer Communications Journal, Elsevier.

Dr. Richard Love is now retired from positions as a professor of medicine (in medical oncology) at the University of Wisconsin (1976-2005), and then a professor of medicine and epidemiology/biostatistics at The Ohio State University (2005-2011) and as a senior adviser at the National Cancer Institute (2007-2011). He is currently scientific director of the International Breast Cancer Research Foundation, an organization he began in the early 1990s, and a leader in health and a social entrepreneur for Amader Gram, a non-governmental organization in Bangladesh. He has suggested the term "Public health oncology" in writing about our need to focus more on populations in our global cancer activities. He is currently developing detailed plans and raising international support for the Amader Gram Cancer Care Initiative—a Bangladeshi rural population-cancer health-directed series of 8 social businesses based in 5 new buildings.