



INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

	Page
Guest Editorial Preface: Special Issue from ISCA CAINE 2018	39
<i>Yan Shi and Gongzhu Hu</i>	
Response Time Minimization and Fairness in Distributed Systems with Central-Server Node Model Using Dynamic Load Balancing	40
<i>Satish Penmatsa and Gurdeep S. Hura</i>	
A Declarative Modeling and an Inference Engine to Generate Non-emotional Head-based Conversational Gestures for Human-humanoid Interactions	49
<i>Aditi Singh and Arvind K. Bansal</i>	
Generation of Audiovisual Materials - Considering Semantic and Impressive Harmony Based on Time Change of Music	67
<i>Yuto Shinjo, Teruhisa Hochin, and Hiroki Nomiya</i>	
QoS of Cloud Computing – Application of the JPManager in a Cloud Service	74
<i>Jiang Guo and Yuehong Liao</i>	

* "International Journal of Computers and Their Applications is abstracted and indexed in INSPEC and Scopus."

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

EDITOR-IN-CHIEF

Dr. Gordon Lee, Professor Emeritus
Department of Electrical & Computer Engineering
5500 Campanile Drive
San Diego State University
San Diego, CA 92182-1326, USA
Email: glee@sdsu.edu

CO-EDITOR-IN-CHIEF

Dr. Ziping Liu, Professor
Department of Computer Science
One University Plaza, MS 5950
Southeast Missouri State University
Cape Girardeau, MO 63701
Email: zliu@semo.edu

ASSOCIATE EDITORS

Dr. Hisham Al-Mubaid
University of Houston-
Clear Lake, USA
hisham@uhcl.edu

Dr. Antoine Bossard
Advanced Institute of Industrial
Technology
Tokyo, Japan
abossard@aiit.ac.jp

Dr. Mark Burgin
University of California
Los Angeles, USA
mburgin@math.ucla.edu

Dr. Sergiu Dascalu
University of Nevada,
Reno, USA
dascalus@cse.unr.edu

Dr. Sami Fadali
University of Nevada,
Reno, USA
fadali@ieee.org

Dr. Vic Grout
Glyndŵr University,
Wrexham, UK
v.grout@glyndwr.ac.uk

Dr. Yi Maggie Guo
University of Michigan
Dearborn, USA
magyiguo@umich.edu

Dr. Wen-Chi Hou
Southern Illinois University, USA
hou@cs.siu.edu

Dr. Ramesh K. Karne
Towson University, USA
rkarne@towson.edu

Dr. Bruce M. McMillin
Missouri University of Science
and Technology, USA
ff@mst.edu

Dr. Muhanna Muhanna
Princess Sumaya University for
Technology
Amman, Jordan
m.muhanna@psut.edu.jo

Dr. Mehdi O. Owrang
The American University, USA
owrang@american.edu

Dr. Xing Qiu
University of Rochester, USA
xqiu@bst.rochester.edu

Dr. Juan C. Quiroz
Sunway University, Malaysia
juanq@sunway.edu.my

Dr. Abdelmounaam Rezgui
New Mexico Tech, USA
rezgui@cs.nmt.edu

Dr. James E. Smith
West Virginia University, USA
James.Smith@mail.wvu.edu

Dr. Shamik Sural
Indian Institute of Technology
Kharagpur, India
shamik@cse.iitkgp.ernet.in

Dr. Ramalingam Sridhar
The State University of New York
at Buffalo, USA
rsridhar@buffalo.edu

Dr. Junping Sun
Nova Southeastern University, USA
jps@nsu.nova.edu

Dr. Jianwu Wang
University of California
San Diego, USA
jianwu@sdsc.edu

Dr. Yiu-Kwong Wong
Hong Kong Polytechnic University
Hong Kong
eeykwong@polyu.edu.hk

Dr. Rong Zhao
The State University of New York
at Sony Brook, USA
rong.zhao@stonybrook.edu

ISCA278 Mankato Ave, #220, Winona, MN 55987 USA.....Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: <http://www.isca@isca-hq.org>.

Copyright © 2019 by the International Society for Computers and Their Applications (ISCA)
All rights reserved. Reproduction in any form without the written consent of ISCA is prohibited.

Guest Editorial Preface

Special Issue from ISCA CAINE 2018

This Special Issue of IJCA is a collection of four refereed papers, three of which were selected from the *31st International Conference on Computer Applications in Industry and Engineering (CAINE 2018)*, and the fourth paper was specially solicited for the Special Issue.

Each paper submitted to the conference was reviewed by at least two members of the International Program Committee and additional reviewers judging the originality, technical contribution, significance and quality of presentation. After the conference, a number of high-quality papers were recommended by the Program Chair to be considered for publication in this Special Issue of IJCA. The authors were invited to submit a revised version of their papers. After extensive revisions and a second round of review, three papers from the CAINE 2018 conference and an additional paper were accepted for publication in this issue of the journal.

The papers in this special issue cover a wide range of research interests in areas of computers and applications. The topics and main contributions of the papers are briefly summarized below.

Satish Penmatsa of Framingham State University, USA, and Gurdeep S. Hura of University of Maryland Eastern Shore, USA, in their paper “*Response Time Minimization and Fairness In Distributed Systems With Central-Server Node Model Using Dynamic Load Balancing*” presented a dynamic load balancing scheme whose objectives are to minimize the execution time of jobs in the system and to provide fairness to the users of jobs. Simulations are conducted with various heterogeneous system configurations to evaluate the performance of the presented dynamic load balancing scheme.

Aditi Singh and Arvind K. Bansal of Kent State University, USA, presented a method for declarative modeling a head-based gestures for non-emotional human-robot interaction in their paper “*A Declarative Model and an Inference Engine to Generate Non-emotional Head-based Conversational Gestures for Human-humanoid Interactions.*” The proposed technique declares gestures as a nested-group of coordinated organ-movements and translates organ-movements to a low-level generic library of routines. The paper described 34 gestures and related algorithms with implementation.

Yuto Shinjo, Teruhisa Hochin, and Hiroki Nomiya of Kyoto Institute of Technology, Japan, presented a method to detect change points of the impression of a music piece in the paper “*Generation of Audiovisual Materials Considering Semantic and Impressive Harmony Based on Time Change of Music.*” The method deals with repeatedly changing of the impression of music pieces to improve multimedia data retrieval based on impressions. A trial experiment using the proposed method showed that the accuracy of change point detection of impression of music piece was improved.

Jiang Guo and Yuehong Liao of California State University Los Angeles, in their paper “*QoS of Cloud Computing - Application of the JPManger in a Cloud Service,*” proposed an approach to collect the metrics information of the performance of the Java-based cloud computing services by using Java instrumentation and find the bottlenecks. An agent-based architecture was used to maximize the automation of the bottleneck detection, run time data collection and performance analysis. The Java instrumentation approach was to insert the probe codes at byte code level, that work well for user-developed Java source code, as well as for thirty party byte code.

We hope you enjoy this special issue of the IJCA and we look forward to seeing you at future ISCA conferences. More information about ISCA society can be found at <http://www.isca-hq.org>.

Guest Editors:

Yan Shi, University of Wisconsin-Platteville, USA, CAINE 2018 Program Chair.

Gongzhu Hu, Central Michigan University, USA, CAINE 2018 Conference Chair.

May 2019

Response Time Minimization and Fairness in Distributed Systems with Central-Server Node Model Using Dynamic Load Balancing

Satish Penmatsa*

Framingham State University, Framingham, MA, USA

Gurdeep S. Hura†

University of Maryland Eastern Shore, Princess Anne, MD, USA

Abstract

Distributed computing systems are often comprised of heterogeneous computing resources with varying service rates and workloads, and heterogeneous communications resources with varying bandwidth and network traffic. The performance of these systems can be improved by proper load balancing of the jobs in the system among the computing resources by considering the heterogeneity. In this paper, we present a dynamic load balancing scheme whose objectives are to minimize the execution time of jobs in the system and to provide fairness to the users of jobs. The computing resources are modeled as central-server nodes with one processor and one or more input/output devices. Simulations are conducted with various heterogeneous system configurations to evaluate the performance of the presented dynamic load balancing scheme.

Key Words: Distributed computing; heterogeneous systems; fairness; dynamic load balancing; response time.

1 Introduction

With the ever-increasing complexity of computing applications, distributed computing systems can be an effective alternative over centralized systems. Distributed computing systems comprise of software components spread over multiple computers for improving the efficiency and performance of applications. These distributed systems provide several benefits compared to centralized systems such as scalability and improved availability of services. Since the distributed computing systems may comprise of heterogeneous computing resources with varying service rates and workloads, and heterogeneous communications resources with varying bandwidth and network traffic, the performance of these systems can be impacted if proper load balancing of jobs is not performed. Given a large number of jobs, the load balancing schemes try to find an allocation of jobs to computers in the system optimizing an objective function (*e.g.* total execution time of jobs).

Load balancing schemes can be categorized as being *static*

or *dynamic*. Static schemes base their allocation decisions on collected statistical information about the system (*e.g.* average arrival rate of jobs into the system), whereas, the dynamic (adaptive) schemes base their decision on the current state of the system (*e.g.* number of jobs waiting in the queue to be processed at a computer) [18]. Also, a load balancing scheme may regard all jobs as one group (class) to provide a *system-optimal* solution, or may regard each job being independent to provide a *job-optimal* solution, or may classify the jobs into multiple groups (classes) to provide a *class-optimal* solution [11].

1.1 Related Work

Kameda *et al.* studied static load balancing in distributed systems considering various network configurations (*e.g.* single channel and star network configurations) [7, 16]. Algorithms were devised for providing system-optimal, job-optimal, and class-optimal solutions. Heuristic techniques for static resource allocation in heterogeneous computing environments with tasks having dependencies, priorities, and deadlines were studied in [1]. A study and comparison of job scheduling techniques in a cluster that could be part of a computational grid were made in [19] and a distributed load balancing model for grid computing systems for minimizing the job execution and communication costs was presented in [9].

Zheng *et al.* studied dynamic load balancing for grid computing systems considering communication delays [21]. A grid architecture with computers belonging to dispersed administrative domains or groups connected with heterogeneous communication bandwidths was considered. Dynamic load balancing policies considering single-channel network configuration with central-server node model have been presented in [20]. All jobs in the system were regarded to belong to one group (*e.g.* one user/class). In central-server model, each node consists of one processor and one or more input/output (I/O) devices. Static and dynamic load balancing schemes with the objective of providing a system-optimal solution for multi-class (multi-user) jobs considering a central-server node model were studied in [14] and references there-in.

Fairness of allocation is an important factor in modern distributed systems and we can say that a load balancing

* Department of Computer Science. Email: spenmatsa@framingham.edu.

† Department of Math. & Computer Science. Email: gshura@umes.edu.

scheme is fair if all the users in the system have approximately the same expected response time independent of the computers allocated for the execution of their jobs (of approximately the same size). In [13], we studied dynamic load balancing for minimizing execution costs of user jobs and for providing fairness to users in grid systems. The nodes were modeled as M/M/1 queuing systems [5] where the job inter-arrival times and service times are exponentially distributed and arrive in a single queue to a single computing resource. Static load balancing schemes for providing fairness considering a central-server node model have been presented in [10, 12]. Load balancing schemes based on game theory for providing fairness to users and their jobs have been presented in [2, 8, 3].

A classification of some of the most used load balancing algorithms in distributed systems (including cloud technology, cluster systems, and grid systems) is presented in [4]. Another survey of task allocation and load balancing in distributed systems with respect to aspects such as control models, resource optimization methods, and coordination mechanisms among heterogeneous nodes has been provided in [6]. In [17], performance analysis of greedy load balancing algorithms in heterogeneous distributed computing systems has been made.

1.2 Contribution

In this paper, we present a dynamic load balancing scheme whose objectives are to minimize the execution time of user jobs and to provide fairness to the users in a multi-user heterogeneous distributed system. The computing resources are modeled as central-server nodes with one processor and one or more I/O devices. Simulations are conducted with various heterogeneous system configurations to evaluate the performance of the presented dynamic load balancing scheme. Related preliminary results were presented in [15].

1.3 Organization

The rest of the paper is organized as follows. In Section 2, the system and node models used are presented. In Section 3, the proposed dynamic load balancing scheme is presented. In Section 4, performance analysis of the proposed scheme is made using experimental results. Conclusions are drawn in Section 5.

2 System and Node Models

In this section, we present the distributed system model and the node model used and the assumptions made for the study. We consider a heterogeneous distributed system as shown in Figure 1 with n computers (nodes) connected by a communications network. Each node is modeled as a central-server node (as shown in Figure 1) which consists of one processor and one or more input/output (I/O) devices. Jobs arriving to each node may belong to m different users (classes) (similar to [14, 12]). We assume that μ_i denotes the service rate of node i and ϕ_i^j denotes the external job arrival rate of user j to node i . The total external job arrival rate of user j is denoted by ϕ^j (where $\phi^j = \sum_{i=1}^n \phi_i^j$) and the total external job arrival rate into the system is denoted by Φ (where $\Phi = \sum_{j=1}^m \phi^j$).

We assume that p_0 denotes the probability that a job after departing from the processor finishes and p_1 denotes the probability that a job after departing from the processor requests I/O service. Therefore, $\frac{p_1}{p_0}$ denotes the average number of I/O requests per job. The job flow rate of user j from node r to node s is denoted by x_{rs}^j . The nodes and the communications network are assumed to have an exponential

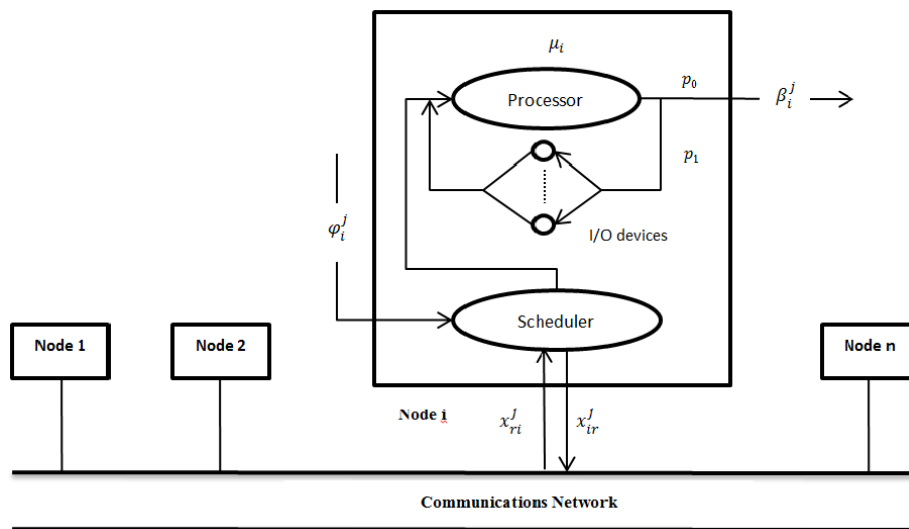


Figure 1: System and node models

service-time distribution and that the job arrivals follow a Poisson distribution [5] (to simulate random, mutually independent job arrivals).

Let β_i^j denote the job processing rate (load) of user j at node i and t_{IO} denote the service time of an I/O device. Based on the above assumptions, the expected (average) response time (node delay) of a user j job processed at node i is given by [5]:

$$F_i^j(\beta_i) = \frac{1}{(\mu_i - \sum_{k=1}^m \beta_i^k)} + \frac{p_1}{p_0} t_{IO} \quad (1)$$

where $\beta_i = [\beta_i^1, \beta_i^2, \dots, \beta_i^m]^T$.

Let λ^j denote the job traffic through the network of user j and t denote the mean communication time for sending or receiving a job from one node to another for any user. The expected communication delay of a user j job is given by [5]:

$$G^j(\lambda) = \frac{t}{(1 - t \sum_{k=1}^m \lambda^k)}, \quad \sum_{k=1}^m \lambda^k < \frac{1}{t} \quad (2)$$

In the above, it is assumed that $G^j(\lambda)$ is independent of the source-destination pair (r, s) but may depend on the total traffic through the network, λ where $\lambda = \sum_{j=1}^m \lambda^j$.

Hence, the overall average response time of user j is given by:

$$T^j(\beta, \lambda) = \frac{1}{\phi_j} \sum_{i=1}^n \beta_i^j F_i^j(\beta_i) + \frac{\lambda^j}{\phi_j} G^j(\lambda) \quad (3)$$

3 Dynamic Load Balancing

In this section, we present a dynamic load balancing scheme (named DNCOOPC-CS) whose objectives are to dynamically minimize the execution time (response time) of users (jobs) in the system and to provide fairness to all the users. We note that an allocation (of jobs) is said to be fair if all the users experience approximately the same expected response time for the execution of their jobs which are approximately of the same size independent of the computers allocated for their execution. Dynamic load balancing schemes base their decision on the current state of the system.

The DNCOOPC-CS scheme is based on the *static* job allocation scheme NCOOPC-CS presented in [12] whose objective is to improve the performance of E-commerce systems by minimizing the response time of user jobs (or transactions) and by providing fairness to all the users. NCOOPC-CS is based on non-cooperative economic game theory and it was shown that an (Nash) equilibrium solution provides an allocation which is fair to all the users. The following marginal node and marginal communication delay functions are defined in [12] for finding a solution implemented by NCOOPC-CS job allocation scheme.

$$f_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j} [\beta_i^j F_i^j(\beta_i)] = \frac{\mu_i^j}{(\mu_i^j - \beta_i^j)^2} + \frac{p_1}{p_0} t_{IO} \quad (4)$$

where $\mu_i^j = \mu_i - \sum_{k=1, k \neq j}^m \beta_i^k$.

$$g^j(\lambda) = \frac{\partial}{\partial \lambda^j} [\lambda^j G^j(\lambda)] = \frac{t g_{-j}}{(g_{-j} - t \lambda^j)^2} \quad (5)$$

where $g_{-j} = (1 - t \sum_{k=1, k \neq j}^m \lambda^k)$.

$$(f_i^j)^{-1}(\beta_i |_{\beta_i^j = x}) = \begin{cases} \left(\mu_i^j - \sqrt{\frac{\mu_i^j}{x - \frac{p_1}{p_0} t_{IO}}} \right), & \text{if } x > \frac{1}{\mu_i^j} + \frac{p_1}{p_0} t_{IO} \\ 0, & \text{if } x \leq \frac{1}{\mu_i^j} + \frac{p_1}{p_0} t_{IO} \end{cases} \quad (6)$$

The state information that DNCOOPC-CS uses is the number of jobs waiting in queue to be processed (queue length) at the nodes. Each node i ($i = 1, \dots, n$) broadcasts the number of jobs of user j ($j = 1, \dots, m$) in its queue to all the other nodes. This state information exchange is done periodically, say every P time units. Expressions for marginal node and communication delays in terms of current state information (instantaneous variables) are derived below (similar to [14, 13]).

Let r_i ($i = 1, \dots, n$) denote the mean service time of a job at node i , N_i^j ($i = 1, \dots, n; j = 1, \dots, m$) denote the mean number of jobs of user j at node i , ρ denote the utilization of the communications network (where $\rho = t \sum_{k=1}^m \lambda^k$), and ρ^j denote the utilization of the communications network by user j .

Using the relation $r_i = \frac{1}{\mu_i}$ [5] and Little's law ($\sum_{k=1}^m N_i^k = \sum_{k=1}^m \beta_i^k F_i^j(\beta_i^k)$) [5], the marginal node delay of a user j job at node i (i.e. f_i^j in eq. (4)) can be expressed in terms of r_i and N_i^j as:

$$f_i^j(\beta_i) = r_i^j (1 + \sum_{k=1}^m N_i^k)^2 + \frac{p_1}{p_0} t_{IO} \quad (7)$$

where $r_i^j = r_i - \sum_{k=1, k \neq j}^m \beta_i^k$.

Rewriting eq. (5) in terms of ρ , we have

$$g^j(\lambda) = \frac{t(1 - \rho + \rho^j)}{(1 - \rho)^2}, \quad \rho < 1, \rho^j < 1 \quad (8)$$

Let n_i^j denote the number of jobs of user j at node i at a given instant, ρ' denote the utilization of the communications network at a given instant, and $\rho^{j'}$ denote the utilization of the communications network by user j at a given instant.

Expressing f_i^j in eq. (7) and g^j in eq. (8) which use the mean estimates of the system parameters in terms of instantaneous variables, we have, the marginal virtual node delay for user j at node i as:

$$f_i^j = r_i^j (1 + \sum_{k=1}^m n_i^k)^2 + \frac{p_1}{p_0} t_{IO} \quad (9)$$

and the marginal virtual communication delay for user j jobs as:

$$g^j = \frac{t(1-\rho'+\rho^j)}{(1-\rho')^2}, \quad \rho' < 1, \rho^j < 1 \quad (10)$$

While NCOOPC-CS tries to balance the marginal node delays of each user at all the nodes statically, DNCOOPC-CS tries to balance the marginal virtual node delay of each user at all the nodes dynamically. For a user u job arriving at node i that is eligible to transfer, each potential destination node j ($j = 1, \dots, n; j \neq i$) is compared with node i .

If

$$f_i^u > f_j^u + g^u \quad (11)$$

then node i is said to be more heavily loaded than node j for a user u job. A user j job arriving at node i will be eligible to transfer when the number of jobs of user j at node i is greater than some threshold denoted by T_i^j . Else, the job will be processed locally. The optimal loads computed by the static NCOOPC-CS are used to determine the thresholds. These thresholds can be recomputed periodically based on the frequency of variation of the arrival rates to each node.

In order to avoid the scenario where a job will be continually transferred to remote nodes without being processed, we keep track of the number of times that a user j job has been transferred (say, c). If ω ($0 < \omega \leq 1$) is a *weighting factor* to prevent a job from being transferred continuously and Δ ($\Delta > 0$) is a *bias* to protect the system from instability (by forbidding the load balancing policy to react to small load distinctions between the nodes), then the job of user j at node i will be transferred to a remote node only if $(\omega)^c \delta_i^j > \Delta$. Here, δ_i^j is a measure which quantifies the maximum difference between the job queue length at the current node and the job queue lengths at all the remote nodes. If $(\omega)^c \delta_i^j \leq \Delta$, the job will be processed locally.

4 Experimental Results

In the following, we present the experimental results comparing the performance of the proposed DNCOOPC-CS load balancing scheme with that of other existing schemes. We simulated a 32-node heterogeneous system with 20 users to evaluate the performance of DNCOOPC-CS (similar to [14, 13]). The system configuration is shown in Table 1 and presents the service rate of the computers (nodes) ($\mu_i, i = 1, \dots, 32$), their relative service rates, and the number of computers of each type. The service rate of the fastest computers is 10 times that of the slowest computers.

Table 1: System configuration

Service rate (jobs/sec)	10	30	60	100
Relative service rate	1	3	6	10
Number of computers	8	8	8	8

For comparison, the following load balancing schemes were also implemented: NCOOPC-CS [12] - a static scheme with the objective of minimizing the total execution time of users jobs and provide fairness to the users; DynamicGOS [14] - a dynamic load balancing scheme with the objective of providing a system optimal solution (but not fairness).

The total job arrival rate into the system is determined by the system utilization (load) and the total service rate of the system. *System utilization* (ψ) represents the amount of load on the system. It is defined as the ratio of the total arrival rate to the aggregate service rate of the system:

$$\psi = \frac{\Phi}{\sum_{i=1}^n \mu_i} \quad (12)$$

The total job arrival rate into the system is divided among the 20 users unevenly to simulate heterogeneous user job arrival rates. The mean service time of each computer i , $i = 1, \dots, 32$ and the mean inter-arrival time of a job of each user j , $j = 1, \dots, 20$ to each computer i are calculated from the mean service rate of each computer i and the mean arrival rate of a job of each user j to each computer i respectively. The mean communication time is assumed to 1 *millisecond*. The overhead (OV) for job transfer we use in the following is defined as the percentage of service time that a computer has to spend to send or receive a job.

4.1 Effect of System Utilization:

The average response times for executing the jobs of all users in the system for system utilizations (system loads) ranging from 10% to 90% are presented in Figure 2. We assume that there is no overhead for job transfer in this case. The bias for job transfer (Δ) is set to 0.4 and the weighting factor for job transfer (ω) is set to 0.9 for both DNCOOPC-CS and DynamicGOS. It can be observed that the average response time achieved by all the schemes is close for low system utilizations. As the system load increases, the average

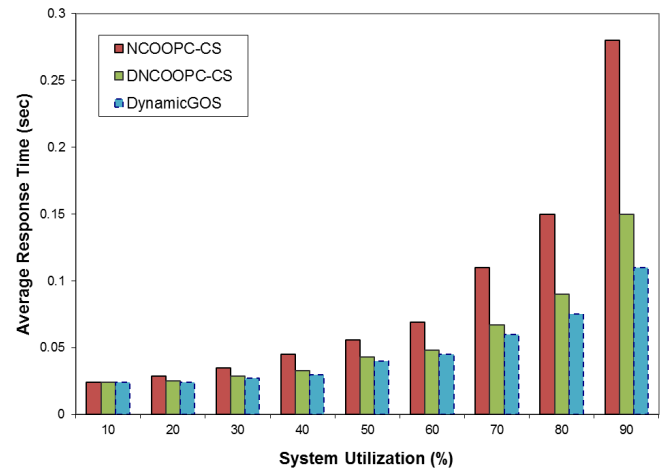


Figure 2: Average response time vs system utilization (OV = 0)

response time increases, and the average response times achieved by the dynamic schemes, which use the instantaneous state information, are substantially lower (by at least about 40%) than that of the static NCOOPC-CS. DynamicGOS, whose objective is to provide a system optimal solution achieves a lower response time compared to DNCOOPC-CS, whose objective is not only to reduce the response time of jobs but also provide fairness to the users.

We use Fairness Index (FI) [5] as the metric to quantify the fairness of the load balancing schemes. If all the users have the same average response time, then $FI = 1$ and the system is 100% fair to all users and it is load-balanced. If FI decreases, then the load balancing scheme favors only some users. The

Fairness Index (FI) of the load balancing schemes under consideration is presented in Figure 3 for various system utilizations. It can be observed that the FI of DynamicGOS falls from 1 at low system loads to about 0.93 at high system loads while the FI of DNCOOPC-CS is in the range $\{0.98, 1\}$.

Figure 4 presents the average response times achieved by the load balancing schemes when the overhead for sending and receiving a job is set to 5% of the mean job service time at a node. It can be observed that the response times of DNCOOPC-CS are substantially lower than that of the static NCOOPC-CS for medium and high system loads. This is because of the use of current job queue lengths at the nodes by the dynamic scheme compared with the use of average state

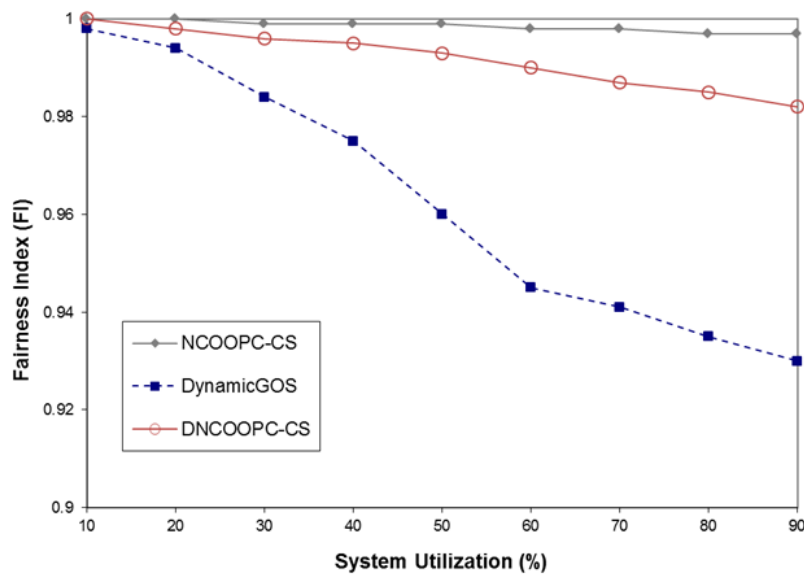


Figure 3: Fairness index vs system utilization (OV = 0)

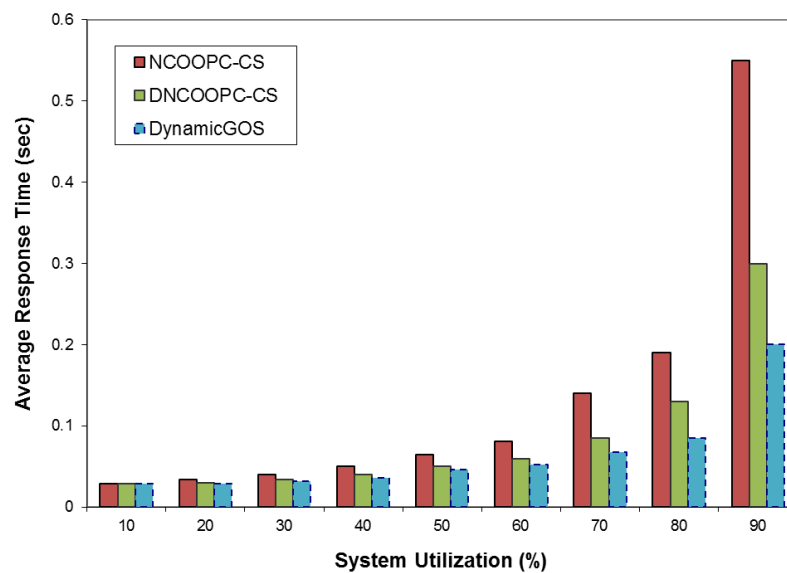


Figure 4: Average response time vs system utilization (OV = 5%)

information by the static scheme. DynamicGOS again achieves a lower response time (at high system loads) than DNCOOPC-CS due to its allocation in order to achieve a near optimal solution. Figure 5 presents the Fairness Index (FI) achieved by the schemes when the overhead for sending and receiving a job is 5% of the mean job service time at a node. It can be observed that the Fairness Index of DNCOOPC-CS is in the range {0.97, 1} and the Fairness Index of DynamicGOS is in the range {0.92, 0.99}. This is because the allocation computed by DNCOOPC-CS is not only to lower the response time of

jobs but also to provide a fair solution.

Figure 6 presents the average response times achieved by the load balancing schemes when the overhead for job transfer is 10%. Although the response times increase considerably, it can be again observed that the response times of DNCOOPC-CS are lower than that of the static NCOOPC-CS for medium and high system loads. Figure 7 presents the Fairness Index (FI) when the overhead for job transfer is 10%. It can be observed that the FI of DNCOOPC-CS is in the range {0.97, 1} and the FI of DynamicGOS is in the range {0.89, 0.99}.

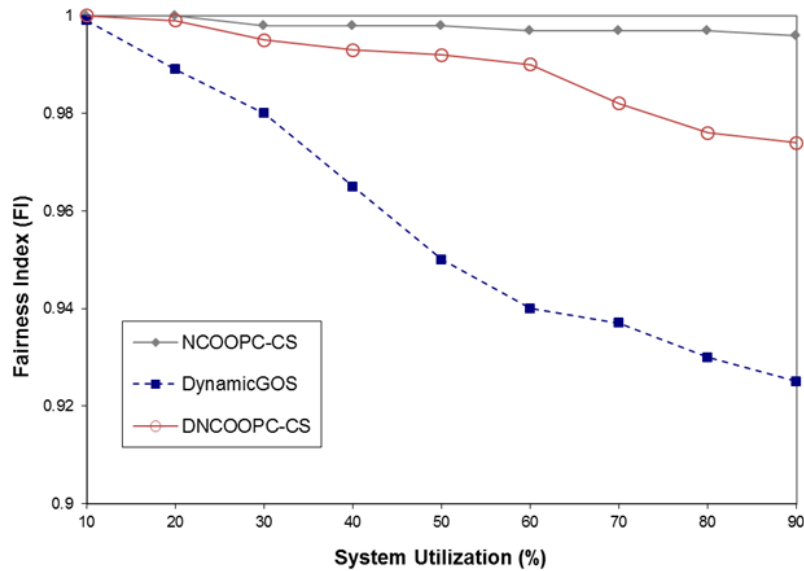


Figure 5: Fairness index vs system utilization (OV = 5%)

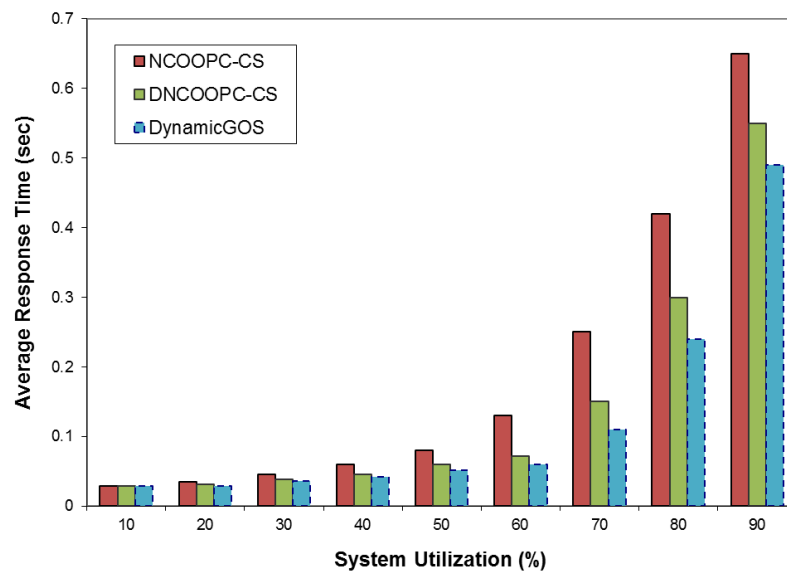


Figure 6: Average response time vs system utilization (OV = 10%)

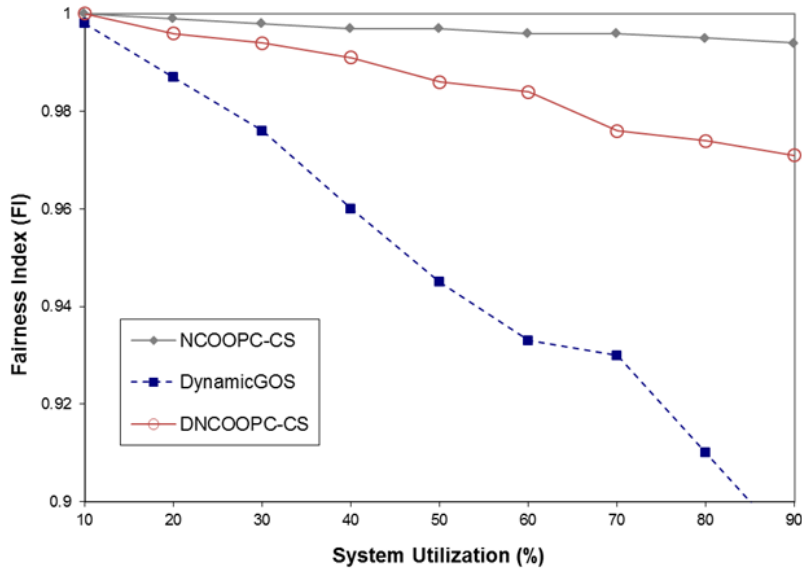


Figure 7: Fairness index vs system utilization (OV = 10%)

4.2 Effect of Bias (Δ)

In the following, we present the effect of *bias* ($\Delta, (\Delta > 0)$) on DNCOOPC-CS which is used to protect the system from instability by forbidding the load balancing scheme to react to small load distinctions between the nodes. Figure 8 presents the variation of average response time with system utilization of DNCOOPC-CS for various biases. The overhead is assumed to be 5% and the other parameters are fixed as in Figure 2. It can be observed that as the bias decreases, the average response time of DNCOOPC-CS decreases. This is

because, as the bias goes lower, it will be relatively easy for the load balancing scheme to find a remote node with a value of δ_i^j that satisfies $(\omega)^c \delta_i^j > \Delta$ (see Section 3) for remote processing. When such a remote node is not found, the job will be forced to be processed locally, which can lead to load imbalance.

Figure 9 presents the variation of Fairness Index (FI) with system utilization of DNCOOPC-CS for various biases. When the bias is set to 0.8, the FI of DNCOOPC-CS falls from 0.99 at low system loads to about 0.94 at high system loads and when the bias is set to 0.6, the FI of DNCOOPC-CS falls from

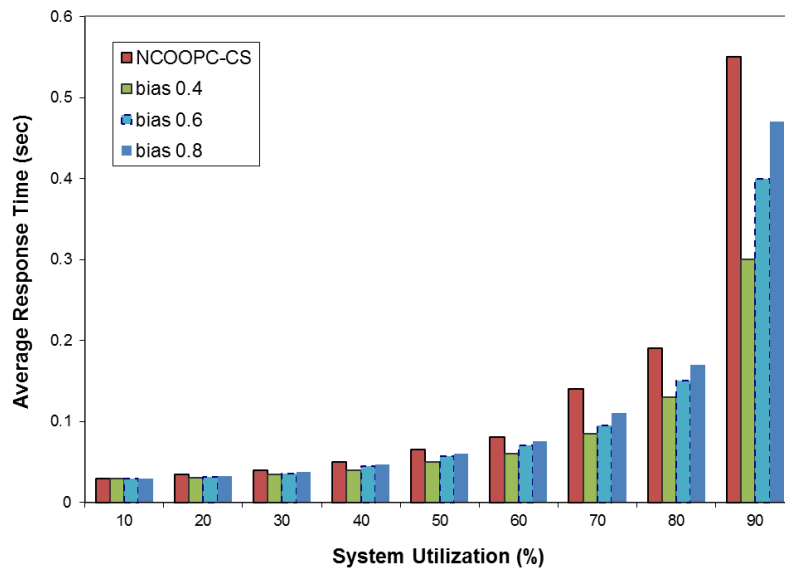


Figure 8: Average response time vs system utilization for various biases of DNCOOPC-CS (OV = 5%)

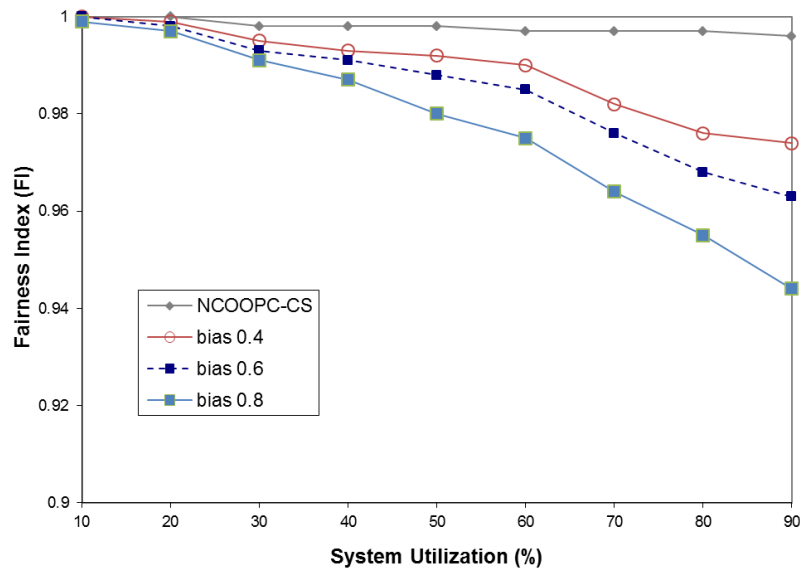


Figure 9: Fairness index vs system utilization for various biases of DNCOOPC-CS (OV = 5%)

1 at low system loads to about 0.96 at high system loads. As the bias reduces to 0.4, the FI of DNCOOPC-CS at high system loads is about 0.98.

Based on the above results, it can be observed that the performance of DNCOOPC-CS is not only close to that of DynamicGOS in terms of the average response time but also provides fairness to all the users (in terms of their experienced response times).

5 Conclusions

In this paper, a dynamic load balancing scheme (DNCOOPC-CS) for heterogeneous distributed systems is presented. The objectives of DNCOOPC-CS are to minimize the average response time of users jobs and to provide fairness to the users. The computers were modeled as central-server nodes. Based on experimental results with various system loads, it was observed that the Fairness Index achieved by DNCOOPC-CS is close to 1 and the average response times achieved by it were considerably lower than its static counterpart and comparable to a system-optimal dynamic scheme.

In future work, we plan to evaluate the performance of DNCOOPC-CS by varying the heterogeneity and the size of the distributed system.

References

- [1] T. D. Braun, H. J. Siegel, A. A. Maciejewski, and Y. Hong, "Static Resource Allocation for Heterogeneous Computing Environments with Tasks having Dependencies, Priorities, Deadlines, and Multiple Versions," *Journal of Parallel and Distributed Computing*, 68(11):1504-1516, Nov. 2008.
- [2] D. Grosu and A. T. Chronopoulos, "Noncooperative Load Balancing in Distributed Systems," *Journal of Parallel and Distributed Computing*, 65(9):1022-1034, Sep. 2005.
- [3] D. Grosu, A. T. Chronopoulos, and M.Y. Leung, "Load Balancing in Distributed Systems: An Approach using Cooperative Games," *Proc. of the 16th IEEE Intl. Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, USA, pp 501-510, April 14-15, 2002.
- [4] I. N. Ivanisenko and T. A. Radivilova, "Survey of Major Load Balancing Algorithms in Distributed System," *Information Technologies in Innovation Business Conference (ITIB)*, pp 89-92, 2015.
- [5] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley-Interscience, 1991.
- [6] Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, 27(2):585-599, 2016.
- [7] H. Kameda, J. Li, C. Kim, and Y. Zhang, *Optimal Load Balancing in Distributed Computer Systems*, Springer Verlag, London, 1997.
- [8] S. Nouri and S. Parsa, "A Non-Cooperative Approach for Load Balancing in heterogeneous Distributed Computing Platform," *Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology*, pp 756-761, 2009.
- [9] J. C. Patni and M. S. Aswal, "Distributed Load Balancing Model for Grid Computing Environment," *Proc. of the*

- International Conference on Next Generation Computing Technologies*, pp 123-126, 2015.
- [10] S. Penmatsa, "Load Balancing for Providing Fairness in Utility Computing Systems with Central-Server Model," *Proceedings of the 30th International Conference on Computers and their Applications*, pp 387-392, March 9-11, 2015.
- [11] S. Penmatsa and A. T. Chronopoulos, "Game-Theoretic Static Load Balancing for Distributed Systems," *Journal of Parallel and Distributed Computing*, 71(4):537-555, 2011.
- [12] S. Penmatsa and G. S. Hura, "Job Allocation in e-Commerce Systems Involving Self-Interested Agents," *Journal of Global Information Technology*, 5(1):1-11, 2010.
- [13] S. Penmatsa and G. S. Hura, "Adaptive Cost Optimization and Fair Resource Allocation in Computational Grid Systems," *Proceedings of the 29th International Conference on Computer Applications in Industry and Engineering*, pp 79-84, 2016.
- [14] S. Penmatsa and G. S. Hura, "Performance Evaluation of System Optimal Load Balancing Schemes for Multi-User Job Distributed Systems," *Proceedings of the 23rd International Conference on Parallel and Distributed Processing Techniques and Applications*, pp 68-73, 2017.
- [15] S. Penmatsa and G. S. Hura, "Dynamic Load Balancing in Heterogeneous Computing Systems with Central-Server Node Model," *Proc. of the 31st International Conference on Computer Applications in Industry and Engineering*, pp 14-19, 2018.
- [16] K. W. Ross and D. D. Yao, "Optimal Load Balancing and Scheduling in a Distributed Computer System," *Journal of the ACM*, 38(3):676-690, July 1991.
- [17] B. Sahoo, D. Kumar, and S. K. Jena, "Performance Analysis of Greedy Load Balancing Algorithms in Heterogeneous Distributed Computing System," *Proc. of the International Conference on High Performance Computing and Applications*, pp 1-7, 2014.
- [18] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *Comput.*, 25(12): 33-44, 1992.
- [19] K. Skenteridou and H. D. Karatza, "Job Scheduling in a Grid Cluster," *Proc. of the International Conference on Computer, Information and Telecommunication Systems*, pp 1-5, 2015.
- [20] Y. Zhang, H. Kameda, and S. L. Hung, "Comparison of Dynamic and Static Load-Balancing Strategies in Heterogeneous Distributed Systems," *IEE Proc. Computers and Digital Techniques*, 144(2):100-106, March 1997.
- [21] Q. Zheng, C-K. Tham, and B. Veeravalli, "Dynamic Load Balancing and Pricing in Grid Computing with Communication Delay," *Journal of Grid Computing*, 6(3):239-253, Sept. 2008.



Satish Penmatsa received his M.S. and Ph.D. in Computer Science from the University of Texas at San Antonio in 2003 and 2007, respectively. He is currently with the Department of Computer Science at Framingham State University, Framingham, Massachusetts. His research interests are in the areas of parallel and distributed

systems, high performance computing, grid computing, wireless networks, game theory, science and engineering applications. He is a member of IEEE, IEEE Computer Society, ISCA, and the ACM.



Gurdeep S. Hura received his M. E. and Ph.D. in Computer Science from the University of Roorkee (India) in 1975 and 1984 respectively. He is currently with the Department of Mathematics and Computer Science at the University of Maryland Eastern Shore, Princess Anne, Maryland. His research interests include petri nets, performance modeling and analysis, and cyber security. He is a senior member of IEEE, a Fellow of Society

for Design and Process Science, and a CAC ABET National and International Program evaluator.

A Declarative Modeling and an Inference Engine to Generate Non-emotional Head-based Conversational Gestures for Human-humanoid Interactions

Aditi Singh* and Arvind K. Bansal*
Kent State University, Kent, Ohio, 44240, USA

Abstract

This paper describes an approach for declarative modeling and an implementation of a major subset of head-based gestures for non-emotional conversational interactions. Conversational interaction between human and robot includes verbal and non-verbal communication. Non-verbal conversations depend on gestures. Gestures require postures, head-motions, hand-motions, jaw-motions and eye-motion, including gaze, and the coordination of various movements. The proposed technique declares gestures as a nested-group of coordinated organ-movements and translates organ-movements to a low-level generic library of routines for the programmatically coordinated rotations of stepper and servo motors. The library of gestures is easily adaptable to individuals' variability and speech variability due to declarative modeling. We describe declarations for 34 gestures, and describer algorithms for an inference engine. We have implemented the approach on an artificial human skeleton using Python programming language, Adafruit driver-library and speech rendering software running on a Raspberry PI 3B.

Key Words: Human-robot interaction; conversational gesture; gesture generation; robotics; social robotics; declarative modeling.

1 Introduction

The human-robot interaction has multiple expected applications in the medical industry, education and entertainment industry [6, 11]. Humanoid robots will act like a companion of people because of the aging population and the lack of availability of caretakers and educators [5, 15, 31, 43]. However, a major concern goes to the physical body of a robot. The focus is towards making robot anthropomorphic.

Anthropomorphism has roots in the Greek words "*Anthropos*" for "*human*" and "*morph*" for "form/shape/structure". Anthropomorphism attributes human-traits to objects to make social robots human-friendly and acceptable by mimicking form factor and behavioral characteristics of humans. In robotics, "anthropomorphic design" refers to three parts: a robot's structure, conduct, and interaction with the human [13]. In the last few years, researchers have developed many android models with limited interaction capabilities [17, 34, 38].

Gesture is a non-verbal communication language that uses postures and coordinated motions of mainly upper body parts [21, 24]. Gestures are essential for interaction and communication to convey the intent [2]. Conversational gesture is an important aspect of human-robot interactions [42]. Robots' interactions with humans are non-emotional. However, robot should be able to exhibit human-traits such as acceptance, affirmation, encouragement, agreement, subordination, backchanneling, along with a meaningful speech for better acceptability and improved communication functionality.

A non-emotional conversational gesture is generated using a coordinated combination of posture and motion mainly involving head, eyes, jaw, hands and shoulders. Human muscles involved in gesture generations have four major functions: 1) movement of organs associated with gestures; 2) graceful dampening of motions; 3) smoothen jerky motion of the organs; and 4) stabilization of the organs involved in the movement in specific postures.

This research describes a declarative and scalable modeling technique and an implementation of a prototype using an artificial human skeleton for generating non-emotional conversational gestures. The implemented gestures involve postures and coordinated movements of head-motion, jaw movements and eyes-movements. This research will augment robot-human conversational interaction efforts [17, 34, 38] that have limited conversational gesture generation capabilities. Although, our scheme has been demonstrated for head-based gestures, it can be extended to more gestures involving hand and body postures [4, 34].

The major contributions in this research are:

1. Development of a scalable declarative model of specifying conversational gestures that are interpreted and actuated using a common engine;
2. Modeling the generation of non-emotional conversational gestures based upon behavioral psychology research [2-3, 16, 35];
3. Gesture-time is automatically scaled to match the speech-time of a phrase using a syllable dictionary for better synchronization between speech-phrase and the associated gesture.

The overall paper is organized as follows. Section 2 describes the related works. Section 3 described a modeling of the postures and coordinated movements using stepper and servo motors.

*Department of Computer Science. Email: asingh37@kent.edu and akbansal@kent.edu.

Section 4 describes the modeling of non-emotional conversational gestures using coordinated movements. Section 5 describes the algorithms. Section 6 describes the implementation. Section 7 describes the limitations and future work.

2 Related Works

Studies [12] have shown the role of physical embodiment in human-robot interaction. The term *embodiment* refers to the physical existence of an entity, i.e. a robot in a physical environment. A robot represents a physical body with actuators and sensors, perception, cognitive and interactive capabilities. There is a huge impact on human perception, reaction and behavior based upon robot-size, human-like behavior, and interaction style towards human-humanoid interactions [41]. Along with human-like appearance, studies [2-3, 12, 32] show the significance of head-motion, speech, gaze and other human-like conversational gestures in human-robot interaction. Studies [9, 16, 27] have also shown the importance of timed gaze during conversation, turn-taking, intent, role-playing, submissions and dominance. Researchers in social robotics have shown a limited amount of gestures in virtual agents (avatars) using 2D or 3D simulations of human behavior [20, 30] and humanoid [10, 17, 19, 34, 38].

Researchers have studied the maximum likelihood of specific body-postures and gestures for different personality types using Bayesian model [4]. Production and synchronization of speech and gesture during a conversation are important for humanoid-human interactions [37]. Researchers have also analyzed relationships between dialogue data and hand gestures [25].

In recent years, researchers have developed many humanoids with limited interaction capabilities with humans. Most notables are ERICA [19], HUBO [34], Nao [7, 29], and Sophia [38]. Roman [23] and Nao [7, 29] exhibit limited built-in gestures, and focus on behavior patterns to interact with the surrounding.

Nao, a humanoid robot, imitates the human arm gesture by capturing the skeleton position data and translating to angle data for motor-movements [17]. A behavior-oriented software framework has been developed that uses: 1) similarity-based color-matching of surrounding entities; and 2) behavior templates based upon a psychological classification of non-verbal part of human behavior. However, conversational gestures such as agreement, encouragement, and rejection, subordination, backchanneling (see Subsection 4.1 for the complete list) have not been addressed. In comparison, our research is about modeling conversational gestures.

ERICA [19] and Sophia [38] model conversational interactions. ERICA supports 44 degrees of freedom, speech synthesis, jaw movement, limited facial expressions, limited lip-movements, limited lip-syncing, blinking and breathing. It also provides limited speech response capability. However, only a set of eighteen gestures has been coded. Sophia [38] supports the features of ERICA, and mimics 62 human facial expressions due to the use of the material frubber [22] – a layered combination of EAP (Electro Active Polymer) that simulates human-skin. Both the androids have limited natural language processing

capabilities besides limited vision processing and directional speech recognition capabilities. A texture-changing skin has been designed for expressive social robots [24] to express fear or excitement. Researchers have also captured segmented gestures and have performed the feature-extraction of gesture-motions during human-robot interaction in a social setting [40].

Compared to ERICA and Sophia, we have implemented 34 gestures (see Subsection 4.1). Our gestures are declarative and are interpreted using a general-purpose engine. Our definition of gestures is influenced by the research of behavior psychologists [2-3, 16, 35]. Modeling a gesture in our platform requires declaration as described in Figure 3 (see Section 4.3) that avoids coding. We can model combinations of complex gestures with ease by combining the declared gestures stored in a library. Our focus in this research is to provide a new method and capability for automatically generating modifiable and learnable gestures by the humanoids.

We are interested in creating a platform where complex conversational interactions are automatically learnt and mimicked by observing human gestures. Our system combines a sequence of declared conversational gestures and archived human phrases. We believe that our methodology will augment other efforts of modeling conversational interaction. Our implementation lacks human-like facial expressions, lip movement independent of jaw movement, tongue movement and blinking, which has been partially achieved using elastomers-based frubber in Sophia [22, 38].

3 Postures and Movements

A posture is modeled as a tuple of rotational angles of organs. Forward looking position with the horizontally leveled face is the *neutral position*. Rotational angles are measured with respect to the neutral position as a reference. A gesture is generated using a combination of coordinated organ-movements that are *sequential*, *concurrent* without synchronization, *synchronized* at the motion-beginnings and/or at the motion-endings, or one motion occurring during another motion.

A motor-rotation is used to model an organ movement $\mu_{\text{motion-type}}^{\text{organ-rotation}}$. An organ-rotation is a *head-tilt*; a *head-nod*; a *head-shake*; *jaw-open*; *jaw-close*; *vergence and tracking* generated using synchronized movement of eyes; and their combinations. A rotation can occur in both positive and negative directions. We express left and downward rotations as negative motion; right and upward rotations as positive motions. *Vergence* requires synchronized motions of eye-motors in the opposite directions, and *tracking* requires synchronized motions of eyes in the same direction.

We abbreviate head-tilt as *tilt*, head-nod as *nod*, neck-rotate as a *shake*; jaw motion as *jaw*; left-eye-rotation as *le* and right-eye-rotation as *re*. Negative direction rotation such as lowering the head is denoted by “– *nod*”; tilt on the left-hand side is denoted by “– *tilt*”; shake in the left direction is denoted by “– *shake*”; and downwards jaw motion (opening a jaw) is denoted by “– *jaw*”. Positive-direction motion such as upward jaw

motion (closing a jaw) is denoted by “+ *jaw*”. Positive-direction motions are usually not prefixed by ‘+’ unless the motion includes a combination of positive and negative motions. A positive motion preceded by a negative motion is denoted by “+−”, and a negative motion preceded by a positive motion is denoted by “−+”. The motion in either direction is denoted by ‘±’.

A motion-type is: 1) *absolute* (denoted as ‘abs’) with respect to the neutral position; 2) *relative* (denoted as ‘rel’) with respect to the current position; 3) move to the neutral (or previous) position using the knowledge of the absolute current position (or the last absolute position); or 4) kernel motor-movements as described in Table 2 in Section 4.3. For example, an absolute motor-movement corresponding to left-eye is denoted as μ_{abs}^{le} ; a relative motor-movement for head-tilt is denoted as μ_{rel}^{tilt} ; a motor-movement for neck rotation to the neutral position is denoted as $\mu_{neutral}^{shake}$. A kernel motor-movement that is applicable to all organ-rotations, and motion-types are denoted as μ_i .

3.1 Synchronized Movements

Two or more movements in one or more threads are synchronized if the start, and the end of the thread are associated with temporal constraints to satisfy a barrier that cannot be violated [1]. For example, motor movements μ_i and μ_j are sequential if μ_j is always executed after μ_i ends.

Out of Allen’s [1] thirteen synchronization types, we have used five types to generate various gestures: 1) *sequential*: motor movements μ_i and μ_j are sequential if μ_j is executed after μ_i ends; 2) *start-synchronized*: motor movements μ_i and μ_j start at the same time; 3) *end-synchronized*: motor movements μ_i and μ_j end at the same time; 4) *strictly synchronized*: satisfies *start-synchronization* and *end-synchronization*; 5) *during*: remaining movements occur during the interval of the first movement.

The conditions for synchronization are summarized in Table 1. The functions *start*, *end*, and *ready* are intuitively clear. The symbol ‘||’ denotes concurrent execution. The symbol ‘ \rightarrow ’ denotes implication. The function *succ*(μ_i) denotes the task executed after the movement μ_i . Sequential motion is denoted as



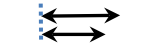

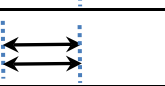
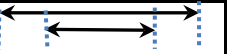
‘ \rightsquigarrow ’ where the left-hand-side event precedes the right-hand-side event. Start-synchronization is denoted as \parallel^{ss} . End-synchronization is denoted as \parallel^{es} . Strict synchronization is denoted as \parallel^{sync} . Inclusion of one or more events during the execution of the first movement is denoted as \parallel^{during} .

Limited implicit synchronized motion can be achieved by relatively expensive motors such as *Dynamixel series* [14] using low-level instruction broadcast to multiple connected motors. However, our scheme is a general-purpose scheme, and works on any shape and size of motors – an important constraint in availability of limited space in the eyeballs.

The software-based synchronization of multiple motor movements, involving different threads, is done using locks. However, synchronization at the thread level does not ensure the synchronized movements of motors due to 1) larger time-periodicity of PWM (Pulse Width Modulation) signals that control the motor movements; 2) lack of positional feedback in stepper and servo-motors; 3) inherent inertia between the start and the stop of consecutive movements in the same servo-motor. To overcome these limitations, small delays are introduced before and after every motor movement. A simple motor movement is a triple of the form $\langle \delta_i^s, \mu_i, \delta_i^e \rangle$ where δ_i^s is the delay before the motor-movement starts, μ_i is the rotation, and δ_i^e is the delay after the motor-movement ends.

Example 1: Modeling synchronized eye motion: An example of the synchronized movement is vergence or tracking which are modeled as (sync, $[(\delta_1^s, \mu_i^{+le}, \delta_1^e), (\delta_2^s, \mu_j^{-re}, \delta_2^e)]$) where sync denotes the strict synchronization of motor actions μ_i^{+le} (left-eye motor moving to right) and μ_j^{-re} (right-eye motor moving to the left). This is realized using a lock that checks the values of Boolean variables: *startLeftEye* and *startRightEye* to be true. After the two Boolean flags become true, the parent thread uses another lock to wait until the values of the Boolean variables *endLeftEye* and *endRightEye* become true before moving to the next activity. Without this synchronization, uncoordinated eye-motions will cause perceptual inconsistency for humans interacting with the robot; eye movement will look unnatural, and images acquired by the robot-cameras will be inconsistent.

Table 1: Synchronization types

Type	Abbreviation	Denotation	Logical Description	Temporal Model
Sequential	<i>seq</i>	\rightsquigarrow	$end(\mu_i) \rightarrow start(\mu_j)$	
no synchronization	<i>conc</i>	\parallel^{conc}	$start(\mu_i) \vee start(\mu_j)$	
Start synchronization	<i>startsync</i>	\parallel^{ss}	$(ready(\mu_i) \wedge ready(\mu_j)) \rightarrow start(\mu_i \parallel \mu_j)$	
End synchronization	<i>endsync</i>	\parallel^{es}	$(end(\mu_i) \wedge end(\mu_j)) \rightarrow start(succ(\mu_i) \parallel succ(\mu_j)) \vee start(\mu_k)$	
Strict synchronization	<i>sync</i>	\parallel^{sync}	$(ready(\mu_i) \wedge ready(\mu_j)) \rightarrow start(\mu_i \parallel \mu_j) \rightarrow (end(\mu_i) \wedge end(\mu_j)) \rightarrow start(succ(\mu_i) \parallel succ(\mu_j)) \vee start(\mu_k)$	
During	<i>during</i>	\parallel^{during}	$(start(\mu_i) \rightarrow start(\mu_j) \rightarrow end(\mu_j) \rightarrow end(\mu_i))$	

3.2 Postures and Movements Declaration

During a conversation, a humanoid takes a posture for a limited time before making the next movement. These limited-time postures are included in the end-delay in the last coordinated movement. We model head-gestures as a nested group of coordinated movements. These group motions (or tasks) are sequential or concurrent, including synchronized tasks as described in Subsection 3.1. Speech rendering involves synchronization of jaw movement with syllables and words in speech. Group-motions are declared as a quadruple of the form (*synchronization-type*, *start-delay*, *a set of coordinated movements*, *end-delay*) where *start-delay* for the *i*th group is denoted by δ_i^{sg} , and the corresponding *end-delay* is denoted by δ_i^{eg} . In case of a sequential motion-group, a set of coordinated movements is represented as a sequence of movements.

The *gesture-time* depends on the sentence being uttered during the gesture. Uttered sentence dilates the gesture-delays accordingly. *Scale-ratio* is derived as the ratio of time taken to generate and render the speech, and the sum of default delays in the coordinated movements involved in a gesture action. The *scale-ratio* dilates the default gesture-delays based upon the time taken by the corresponding sentence being spoken during a gesture. *Gesture-time* is derived as a product of *scale-ratio* and the sum of *default delays*.

Vergence requires both the eyes to focus by moving the pupil in opposite directions: right-eye moving to left-direction and left-eye moving to the right-direction in strict synchronization. *Vergence* is modeled as (*sync*, $[(\delta_1^s, \mu_i^{+le}, \delta_1^e), (\delta_2^s, \mu_j^{-re}, \delta_2^e)]$). *Divergence* is the inverse of *vergence* and is denoted as “-*vergence*”. During *divergence*, right-eye rotates right, and left-eye rotates left. *Divergence* is modeled as (*sync*, $[(\delta_1^s, \mu_i^{-le}, \delta_1^e), (\delta_1^s, \mu_i^{+re}, \delta_1^e)]$). In some gestures, *vergence* is followed by *divergence* after a user-defined delay δ , and is denoted by *vergence-then-divergence*. To simplify the notions, we will denote *vergence* as $(\delta_1^s, \mu_i^{+vergence}, \delta_1^e)$, *divergence* as $(\delta_1^s, \mu_i^{-vergence}, \delta_1^e)$ and *vergence-then-divergence* by $(\delta_1^s, \mu_i^{+vergence}, \delta_1^e)$. *Tracking* is another coordinated eye-motion where both the eyes move in the same direction. *Tracking* is modeled as (*sync*, $[(\delta_1^s, \mu_i^{\pm le}, \delta_1^e), (\delta_1^s, \mu_i^{\pm re}, \delta_1^e)]$). We denote tracking in the right direction as $(\delta_1^s, \mu_i^{+tracking}, \delta_1^e)$ and tracking in the left direction as $(\delta_1^s, \mu_i^{-tracking}, \delta_1^e)$. Tracking in either direction or continuous tracking is denoted as $(\delta_1^s, \mu_i^{\pm tracking}, \delta_1^e)$. *Gaze* is defined as wait(δ) combined with *vergence* and gesture-specific user-defined delay.

Besides intentional motion, one or more organs are involved in random motions when a person is relaxed or during some gestures such as *being unsure* (see Figure 3). We denote such motions by prefixing the movement by the word “random” such as *random-nod*, *random-shake*, and *random-eye*. For a small amount of motion, we prefix the motion by the fuzzy-word “slight” such as *slight-nod*, *slight-tilt*, *slight-shake*, *slight-vergence*.

An utterance or speaking a phrase is strictly synchronized with the jaw motion. In the neutral position, jaw is slightly open, and lips are closed. A jaw opens when a person starts speaking, and returns to the normal relaxed state (slightly open) after the speech is over. A speech is modeled as $(\mu_i^{-jaw} \parallel^{sync} \text{render_speech}(<phrase_file>)) \rightsquigarrow \mu_{neutral}^{jaw}$. For the convenience, we abbreviate the whole action as *speech-phrase* (*<phrase>*).

3.3 Motion Parameterization

A motion is modeled using five types of parameters: *angle*, *speed*, *delay*, *dampening-factor*, and *dampening-count*. *Delay* is used to coordinate and synchronize two motor movements. *Dampening-factor* ($0.0 < \text{dampening-factor} \leq 1.0$) is a multiplicative factor to express a repeating organ movement with progressive reduction of displacements in every iteration. *Dampening-count* (≥ 1) is the number of times a motion repeats back-and-forth before stops. A repeated movement of the same organ is modeled as a sequence $\langle \theta, \eta\theta, \eta^2\theta, \dots, \eta^m\theta \rangle$ where θ is the initial angle of rotation, η is the dampening-factor, and m is the dampening-count. The value of an angle can be positive or negative and is limited by the extent an organ can move in either direction.

The parameters are given as a combination of fuzzy values and concrete values for the convenience to an application programmer while modeling a complex gesture or a sequence of gestures. The fuzzy parameters are defuzzified into concrete values based on the motor’s actual allowed ranges and body movement constraints before storing the corresponding concrete values into a dynamic parameter dictionary \mathcal{DP} .

4 Non-Emotional Interactions

Non-emotional interactions exhibit intentions, reactions, attitude, roles (possibly dynamic) of the two actors involved in a conversation [32]. They express reaction to other parties’ actions without exhibiting intensity variations of an emotional interaction.

4.1 Gestures

Conversational gestures are *acceptance*, *admiration*, *agreement*, *affirmation*, *appreciation*, *argument*, *avoidance*, *backchannel*, *confidence*, *confusion*, *discourage*, *defensive*, *defiance*, *denial*, *depression*, *dominate*, *disagreement*, *encouragement*, *expectation*, *frustrated*, *greet*, *inclusion*, *interested*, *interject*, *interrogate*, *permit*, *persuade*, *plead*, *question*, *reject*, *relaxed*, *request*, *ridicule*, *seek-attention*, *submit*, *unsure*, and *veiled disagreement*.

During a non-emotional conversation, nodding the head is inferred as *acceptance*. Humans show admiration when they like to listen to the speaker. The gesture involves tilt and jaw-movement to the neutral position. *Agreement/Affirmation* is confirmation to convey ‘yes’ or ‘no’. It involves one or more nods. *Appreciation* varies from individual to individual. One such variation of appreciation is tilt followed by a speak-phrase

followed by return to the neutral position.

Humans argue when they want to prove they are right. The motion involves vergence with multiple speak-phrases. *Avoidance* involves the head down with a slight tilt. *Backchannel* shows listening with acceptance or agreement. It involves periodic nodding of head and gaze to show the participation in a conversation.

Humans show *confidence* by moving the head up and gaze. A person shows *confusion* when they cannot understand something. This is modeled by random eye-motions and shaking the head. *Discourage* is shown by an authority to their subordinate by moving the head up and maintaining eye-contact. An individual shows *defensiveness* to protect himself through argumentation when found guilty or trying to prove his/her point. It involves repeated combinations of head shakes and speak-phrase.

Defiance is a form of daring or bold resistance to authority or to any opposing force. *Disagreement* is expressed by a repeated head-shake. *Denial* is disagreement where a speak-phrase occurs during the shake. *Depression* is expressed by lowering the head for a long time. *Rejection* is denial associated with an utterance “no” and a speak-phrase. *Ridicule* (making fun of others) is exhibited by concurrently tilting and slightly lowering the head followed by a speak-phrase.

Domination is a dynamic role that may change between conversing actors depending upon the context of the conversation. It is modeled by moving the head up with a slight head-rotation, a slight tilt and eye-contact accompanied by vergence and a possible speak-phrase. *Encourage* is shown by an actor to another conversing actor by moving up the head and maintaining eye contact. *A person* shows an *expectation* by slightly tilting the head and tracking. *Frustration* is an active reaction that comes with irritation when a person is not interested. In such a situation, the head shakes, and eyes track randomly.

Greeting is culture specific. In eastern cultures, greeting involves lowering of the head associated with greeting word. In contrast, greeting an equal or friend as in western culture is associated with head moving up slightly associated with a greeting-word. *Inclusion* is a behavior shown by a person to do something together. The person tilts the head moderately to either direction with one or more nod. *Interest* is shown by a conversing actor to show active participation by nodding one or more times and tilt head while maintaining an eye-contact. *Interjection* is an interruption during a communication. A person interjects by moving up the head associated speak-phrase and eye-contact. A person shows *permission* by bending the head down slightly along with an utterance.

Persuasion shows that a person is attentive and earnestly encouraging a person to take up a task. He/she tilts and moves up the head slightly along with an appropriate speak-phrase. *Pleading* is requisition when requesting a person is not in control over the situation and wants something desperately. A person *pleads* by moving the head down along with an associated speak-phrase. *Request* is associated with lowered and tilted head along with a speak-phrase. An actor may ask a *question*, possibly interrupting another actor(s) during a conversation, by tilting or

moving the head slightly up along with the speak-phrase. *Relaxed* is shown by after a gesture or after a speech. *Relaxation* is random head nod, shake and eye tracking.

In *seeking attention* gesture, the speaker exhibits varying head orientation and gaze fixed to the listener. *Submission* is exhibited by lowering the head associated with a speak-phrase. *Unsure* gesture is exhibited by a slight tilt in either direction, gaze or jaw closing. *Veiled disagreement* is not openly expressed [28], and is exhibited by slight head-tilt and jaw-closing.

An actor exhibits the conversational gestures in one of the three roles: *listener*, *speaker* or *short intermittent conversation*. Many gestures are not accompanied by any speech-phrase. For example, *arrogance*, *confusion*, *depression*, *disagreement*, *dominance*, *denial*, *interested*, *relaxed* and veiled disagreement mode require no speech. At most, they are associated short phrase or a single word utterance. Speech along with voice modulation reduces the ambiguity. The gestures *acceptance*, *admiration*, *affirmation*, *agreement*, *avoidance*, *backchannel* involve short speech-phrases. The gestures *argument*, *confidence*, *encouragement*, *greet*, *permission*, *persuade*, *plead*, *request*, *ridicule* (including sarcasm) and *seek-attention* are usually accompanied with a longer speech-phrases. All gestures associated with attention such as *backchannel*, *interest*, *plead*, *request* and *seeking attention* requires gaze. Various motions to generate gestures are summarized in Figure 3 under Section 4.3.

4.2 Kernel Movements

The gesture motions are mapped to eleven categories of kernel movements for each organ. These kernel movements are combined to form complex motions to model the gestures. The kernel movements are implemented using basic motor actions: 1) absolute rotation; 2) relative rotation; 3) returning to the previous or neutral position; and 4) performing repeated motions for a finite time with dampening. The eleven kernel movements are: 1) *rot_abs* (denotation μ_{abs}) – rotation by an absolute user-defined angle; 2) *rot_rel* (denotation μ_{rel}) – rotation by a relative user-defined angle; 3) *rot_neutral* (denotation $\mu_{neutral}$) – rotate to the neutral position from the current position; 4) *rot_abs_return* (denotation μ_1) – rotate to an absolute user-defined angle and return to original position after some gesture-specific time-delay; 5) *rot_rel_return* (denotation μ_2) – rotate by a relative user-defined angle with respect to current position and return to the original position after some gesture-specific delay; 6) *rot_abs_neutral* (denotation μ_3) rotate to an absolute user-defined angle, and return to the neutral position; 7) *rot_rel_neutral* (denotation μ_4) – rotate relative to the current position and return to the neutral position; 8) *rot_abs_bi* (denotation μ_5) – rotate sideways by absolute user-defined angles on both directions, and return to the original position; 9) *rot_rel_bi* (denotation μ_6) – rotate sideways by absolute user-defined angles on both directions, and return to the original position; 10) *rep_rot_abs_bi_neutral* (denotation μ_7) – repeatedly perform the motor-action *rot_abs_bi* and then return to the neutral position; 11) *rep_rot_rel_bi_neutral* (denotation μ_8) – repeatedly perform the motor-action μ_5 and return to the original position. *Dampening-factor* η is associated with the

motion's *rep_rot_abs_bi_neutral* and *rep_rot_rel_bi_neutral*.

The kernel motions are padded with delays on both sides. The notation ' \bullet ' describes the composition of motions. $Y\bullet X$ denotes X followed by Y. Table 2 describes the kernel movements as a composition of simpler kernel movements and their applications in modeling sample gestures and interaction scenarios.

4.3 Abstract Grammar for Movement Generation

An abstract grammar (represented in an extended BNF form) for the modeling gestures is described in Figure 1. The bold symbols are terminal symbols; a pair of the angular brackets contains non-terminal symbols. A gesture is a *simple-motion*, or a *motion-group* associated with zero or more *speech-files*. A *simple-motion* is a triple of the form (*start-delay*, *motionTuple*, *end-delay*).

Each organ motion can be: 1) a single motion; 2) a group of simple-motions; 3) a nested group of motions. A *motion-tuple* is a 4-tuple of the form (*kernel-motion*, *motion-type*, *direction*, *extent*). A *motion-type* is an element of the set {*nod*, *shake*, *tilt*, *jaw*, *vergence*, *tracking*}. A *direction* is an element of the set {'+', '-', '+ -', '- +', '±'}; the *extent* is a fuzzy value, or a concrete value constrained between $-60^\circ - +60^\circ$. A *fuzzy value* is an element in the set {'- high', '- moderate', '- slight', 'neutral', 'slight', 'moderate', 'high'}.

At least one movement in a nested group-of-motion is a group-of-motions. A *motion-group* is a group of concurrent (or sequential) motions with five types of synchronization. *Synchronization-type* is an element of the abbreviated set {*seq*, *conc*, *ss*, *es*, *sync*, *during*}. A *motion* within a *motion-group* can be a *simple-motion* or a *motion-group*. The delays can be of multiple types: simple-motion start-delay denoted as δ_i^s ; simple-motion end-delay denoted as δ_i^e ; motion-group start-delay denoted as δ_i^{sg} , motion-group end-delay denoted as δ_i^{eg} . A *motion-group* can have embedded optional *speech-phrases*.

Figure 2 describes a schematic to compute the time taken to utter a phrase. The scheme comprises two associated dictionaries: *word* \rightarrow *syllable-list* and *syllable* \rightarrow *timing*. Summing up the syllable-timings gives the times taken to utter a word. Time taken to utter a phrase is derived by summing the time taken to utter the included words and silence-time between the words. For example, the word "hello" has two syllables: 'hel' and 'lō' with utterance time of 0.8 and 1.2 seconds, and a cumulative time of 2.0 seconds.

4.4 Gesture Modeling and Generation

Figure 3 shows organ movements and the corresponding motor movements encoding for 34 implemented gestures (see Subsection 4.1). A *Kernel-motion* is one of the eleven motions described in Table 2. A *motion-group* is modeled as (*synchronization-type*, *initial-delay*, [*set of motions*], *final delay*). As described earlier. Vergence, tracking, and speak-phrase are coordinated group-actions with synchronization. However, they have been labeled like an atomic motion for convenience. Speak-phrases are user-defined, or a default file

associated with the gestures. Default-files for gestures are stored in a database and are automatically picked in the absence of a user-defined file.

A motion-group is executed based upon the synchronization type. For the sequential group, the movements are executed left to right. With the nested group of motion, the descendant group is executed before the parent-group. The last group of motions in many gestures returns the organs to the neutral positions either sequentially or in a synchronized manner depending upon the gesture. If a motion is a nested group motion, the nesting level is incremented by one.

Example 2: Let us take the gesture of *defiance* encoding in Figure 3. The corresponding motion is modeled as synchronized head tilt and vergence, followed by raising the head slightly followed by speaking a phrase such as "No, I will not do it" followed by head returning to neutral position. Vergence and speak-phrase form a composite synchronized group-motion. The motion is described as [*head-tilt* \parallel^{sync} *vergence*] \rightsquigarrow *raise the head slightly* \rightsquigarrow *speak phrase* \rightsquigarrow *return head to the neutral position* concurrently performing the motor actions.

5 Gesture Interpretation Engine

The major algorithms involved in the implementation are: 1) translating conversation gestures as nested concurrent movements invoked by concurrent threads; and 2) implementing synchronization using locks. The intermediate steps require 1) defuzzification of parameters, 2) storing and retrieving the parameter-values in a dynamic parameter dictionary \mathcal{D}^p , 3) spawning concurrent threads based upon the associated synchronization type in a group-motion, 4) updating the dynamic dictionary after every motor movement, and 5) translation of complex motor instructions to basic Adafruit motor rotations.

5.1 Dictionaries

The dictionaries are: 1) a *gesture-dictionary* \mathcal{D}^g , 2) *organ-motions* \mathcal{D}^m , 3) a dynamic parameter-dictionary \mathcal{D}^p , and 4) a dynamic execution-environment dictionary \mathcal{D}^e . Each tuple of the gesture-dictionary \mathcal{D}^g contains three types of information: a) gesture-name, b) a gesture encoding as described in Figure 3, and c) associated speech-files. A tuple in the organ-motions dictionary \mathcal{D}^m carries the three types of information: 1) organ-motion name, and 2) the set of associated constraint, and 3) the set of default parameter values associated to the motion. The parameter-dictionary \mathcal{D}^p carries information such as current position, speed, delay, dampening-factor, and damp-count for each motor.

The *dynamic execution-environment dictionary* \mathcal{D}^e contains two types of information: *environment-dump* and *Boolean flags* for the synchronization of motions in a motion-group. Each environment-tuple of \mathcal{D}^e carries three information: 1) nested-path of the parent, 2) the index of the task in the current coordinated-movement, and 3) number of sibling motions concurrent. Boolean flags are used to test various

synchronization conditions for synchronization types and set up spin-lock to block the parent-threads until the conditions needed for synchronization of child-tasks are satisfied.

5.2 Fuzzification/Defuzzification

The user-defined parameters for the motor movement are a combination of actual values and fuzzy values. The concept of fuzzy logic is important to model a gesture. The initial step is to set up the fuzzy values for the organs. The algorithm requires the inputs: *organ name*; *motor attributes*; and *fuzzy values*. The fuzzy set contains values in both the directions (positive and negative) with the relaxed position as the neutral position. The fuzzy set is defined as $\{-high, -moderate, -slight,$

neutral, *slight*, *moderate*, *high* $\}$. The defuzzification of fuzzy values is done using a linear scale before actuating a motor movement. It is mapped to the range of human allowable angles for different motions.

5.3 Notations

The variable names suggest their roles in the algorithm. Tuples are denoted using a pair of parentheses. The selection of the i th field of a tuple τ is denoted by the symbol $\Pi_i(\tau)$. For example, $\Pi_2((4, \mu_{abs}, 6))$ derives μ_{abs} . Union of two sets is denoted by 'U'; membership in a set is tested by the Greek symbol '∈'; complement of a predicate is denoted by the

Table 2: Kernel motions and their application in modeling gestures

Motion-name/Denotation		Description and Applications
rot_abs	μ_{abs}	<i>Operation</i> : rotate by an absolute angle. <i>Application</i> : It is a basic rotation used in other kernel-movements. Used in many gestures such as accept; admiration; agreement; affirmation; appreciate; avoid; discourage; defensive; defiance; depressed; frustrated; interject; persuade; plead; question; request; ridicule; seek-attention; submit.
rot_rel	μ_{rel}	<i>Operation</i> : rotate by a relative angle denoted by O_{rel} . <i>Application</i> : It is a basic rotation used in other kernel-movements
rot_neutral	$\mu_{neutral}$	<i>Operation</i> : rotate to the neutral position. <i>Application</i> : It is a basic rotation used in other kernel-movements. Used in many gestures to return the organs back to the relaxed position such as backchannel; confident; confusion; discourage; defiance; denial; dominate; encourage; greet; include; interested; permit; persuade; plead; question; reject; request; ridicule; seek attention; and submit.
rot_abs_return	μ_1	<i>Operation</i> : store the current position ϕ in the parameter dictionary \mathbb{D}^P as the previous_position. Rotate by the user-defined absolute angle θ . Wait for the gesture-specific delay τ . Retrieve the previously stored angle ϕ from the dictionary \mathbb{D}^P and perform the operation <i>rotate_abs</i> (ϕ). $rotate_abs(\phi) \bullet retrieve(prev, \mathbb{D}^P, \phi) \bullet wait(\delta) \bullet rotate_abs(\theta) \bullet store(prev, \mathbb{D}^P, cur_pos(\mathbb{D}^P))$ <i>Application</i> : Used in gestures such as backchannel; confusion; defiance; dominate; persuade; unsure.
rot_rel_return	μ_2	<i>Operation</i> : $rot_rel(-\theta) \bullet wait(\delta) \bullet rot_rel(\theta)$ <i>Application</i> : shifting attention; gaze in a multi-party conversation
rot_abs_neutral	μ_3	<i>Operation</i> : $rot_neutral \bullet wait(\delta) \bullet rot_abs(\theta)$. <i>Application</i> : avoid; confident; defiance; defensive; denial; frustrated; veiled disagreement; unsure
rot_rel_neutral	μ_4	<i>Operation</i> : $rot_neutral \bullet wait(\delta) \bullet rot_rel(\theta)$. <i>Application</i> : open and close a jaw during speak-phrase.
rot_abs_bi	μ_5	<i>Operation</i> : $rotate_abs(\phi) \bullet retrieve(prev, \mathbb{D}^P, \phi) \bullet wait(\delta) \bullet rot_abs(-\theta) \bullet wait(\delta) \bullet rot_abs(\theta) \bullet store(prev, \mathbb{D}^P, cur_pos(\mathbb{D}^P))$ <i>Application</i> : soft agreement; denial.
rot_rel_bi	μ_6	<i>Operation</i> : $rot_rel(\theta) \bullet wait(\delta) \bullet rot_rel(-2\theta) \bullet delay(\tau) \bullet rot_rel(\theta)$. <i>Application</i> : It is used in multi-party interaction.
rep_rot_abs_bi_neutral	μ_7	<i>Operation</i> : It repeatedly rotates a motor n times, and returns to the neutral position. $rot_neutral \bullet wait(\delta) \bullet (rot_abs_bi)^n$. <i>Application</i> : It is used for strong agreement and denial with single person.
rep_rot_rel_bi_neutral	μ_8	<i>Operation</i> : It repeatedly rotates a motor n times, and returns to the neutral position. $(rot_rel_bi)^n$. <i>Application</i> : It is used in multi-party interaction. accept; admiration; agreement; affirmation; argument; for strong agreement; greet; denial; reject; request; relax

```

<interaction> :: {<gesture>}+
<gesture> :: '(' (<simple-motion >|<motion-group>) ',' (<speechFile>|'*'|ε)* ')'
<motion-group > :: '(' (<sync-type> ',' <delay> ',' {(motion-group >| <motion> | <speech-phrase>)}+ ',' <delay> ')'
<simple-motion> :: '(' <delay> ',' <motion-tuple> ',' <delay> ')'
<motion-tuple> :: '(' <kernel-motion> ',' <motion-type> ',' <direction> ',' <extent> ')'
<extent> :: <concrete-value> | <fuzzy-value>
<sync-type> :: (seq | conc | ss | es | sync | during)
<kernel-motion> :: (μabs | μrel | μneutral | μ1 | ... | μ8)
<motion-type> :: (nod | shake | tilt | jaw | vergence | tracking)
<direction> :: ('+'| '-' | '+ -'| '- +' | '±')
<fuzzy-value> :: (- high | - moderate | - slight | neutral | slight | moderate | high)

```

Figure 1: Phrase-time lookup dictionary

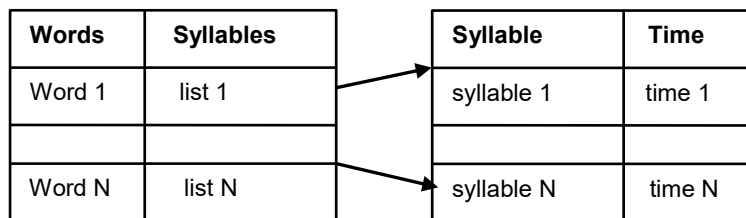


Figure 2: A schematic for computing speech-duration

symbol ‘→’, insertion in a set or a dictionary is denoted using an overloaded symbol ‘+’, insertion in a sequence using the symbol ‘+’, a sequence is denoted within a pair of angular brackets < ... >, and a set or dictionary is denoted using a pair of curly brackets { ... }.

5.4 Algorithms for Gesture Execution

To execute a gesture, a user specifies the gesture along with the user-defined parameters that can be current, previous and default values of *angle*, *speed*, *delay*, *dampening-factor*, and *dampening-count* for each motion. We select the default speech-file from the gesture-dictionary D^G .

The execution of a gesture is modeled as a motion-tree (see Figure 4). The execution pattern is obtained using the gesture-name from the dictionary D^G . Execution pattern includes a gesture-tuple *gestureTuple* that contains the information about gesture-encoding *gestureEncoding* and defuzzified user-defined parameters along with the user-defined speech-file. In the absence of user-defined speech-file, we select randomly a default speech-file from the gesture dictionary D^G . Two indices, *nestingLevel* and *motionIndex* uniquely identify the current motion. The variable *nestingLevel* stores the depth of the motion-tree, and the variable *motionIndex* identifies the position of a motion within the same group. Every time a new motion-group is started the current environment $env = (syncType, motionPath, motionCount)$ is stored in the environment dictionary D^E .

The variable *syncType* describes the synchronization type of the motion-group, and the variable *motionCount* describes the total number of motions in the motion-group. The variable *motionPath* is a sequence of the pair (*nestingLevel*, *motionIndex*), and uniquely identifies any motion or motion-group within a gesture.

A gesture can invoke a simple motion as in the *depression* or a motion-group. A simple motion actuation requires two parameters: *gestureTuple* to invoke the motion and the corresponding *speech-phrase*. The motion-group activation requires two additional parameters: *gestureEncoding* and *motionPath* to extract the current motion-group. We describe an algorithm in Figure 5.

A simple motion is executed by creating and spawning a thread with the motion-tuple *motionTuple* and corresponding motion-parameters. The *motionTuple* for a simple motion is a triple of the form (*motionType*, *motionName*, *extent*) where *motionType* is one of the eleven kernel motions described in Table 2, *motionName* is an element of the set (*nod*, *tilt*, *shake*, *jaw*, *left-eye*, *right-eye*, *jaw*, *vergence*, *tracking*), and *extent* is the magnitude of the angle. We describe an algorithm in Figure 6.

We describe an algorithm for executing motion-groups in Figure 7. A motion-group uses D^E to store the dump of previous environments (*synchronization-type*, *motionPath*, *motionCount*), builds a new environment by incrementing the *nestingLevel* by one, resetting the *motionIndex* to 1, and resetting the *motionCount* to the number of motions in the

Accept/Admiration/Agreement/Affirmation

Motion: head tilt \rightsquigarrow [nod \parallel^{ss} speak-phrase] \rightsquigarrow return to original position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{tilt}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_8^{-nod}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg}), (\delta_3^s, \mu_{abs}^{+nod}, \delta_3^e), (\delta_4^s, \mu_{abs}^{-tilt}, \delta_4^e)], \delta_0^{eg})$

Appreciate

Motion: head tilt \rightsquigarrow speak-phrase \rightsquigarrow return to original position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{+tilt}, \delta_1^e), speakphrase(< phrase >)], (\delta_2^s, \mu_{abs}^{-tilt}, \delta_2^e)], \delta_0^{eg})$

Argument

Motion: focus \rightsquigarrow [nod \parallel^{ss} speak-phrase] \rightsquigarrow tilt \rightsquigarrow [nod \parallel^{ss} speak-phrase]⁺ \rightsquigarrow return head

Encoding: $\left((seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{vergence}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{-nod}, \delta_2^e), speakphrase(< utterance >)], \delta_1^{eg}), (\delta_3^s, \mu_{abs}^{-tilt}, \delta_3^e)], \right. \\ \left. (ss, \delta_1^{sg}, [(\delta_2^s, \mu_8^{-nod}, \delta_2^e), speakphrase(< utterance >)], \delta_1^{eg})^*, (\delta_4^s, \mu_{neutral}^{-nod}, \delta_4^e)], \delta_0^{eg} \right)$

Avoid

Motion: [lower head (negative) \parallel^{ss} rotate head in either direction]

Encoding: $(ss, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{-nod}, \delta_1^e), (\delta_2^s, \mu_3^{-vergence}, \delta_2^e)], \delta_0^{eg})$

Backchannel

Motion: vergence \rightsquigarrow (listen \rightsquigarrow [nod \parallel^{ss} speak-phrase]⁺) \rightsquigarrow diverge to neutral position \rightsquigarrow return head to neutral position

Encoding: $\left((seq, \delta_0^{sg}, [(\delta_1^s, \mu_1^{vergence}, \delta_1^e), (listen, (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{nod}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg})^+], \right. \\ \left. (\delta_3^s, \mu_{neutral}^{-vergence}, \delta_3^e), (\delta_4^s, \mu_{neutral}^{-nod}, \delta_4^e)], \delta_0^{eg} \right)$

Confident

Motion: vergence \rightsquigarrow [head-nod slightly \parallel^{ss} vergence] \rightsquigarrow speak-phrase \rightsquigarrow [return jaw to neutral \parallel^{ss} return head to neutral position]

Encoding: $\left((seq, \delta_0^{sg}, (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{nod}, \delta_2^e), (\delta_3^s, \mu_1^{vergence}, \delta_3^e)], \delta_1^{eg}), speakphrase(< phrase >)], \right. \\ \left. (ss, \delta_2^{sg}, [(\delta_4^s, \mu_{neutral}^{-jaw}, \delta_4^e), (\delta_5^s, \mu_{neutral}^{-nod}, \delta_5^e)], \delta_2^{eg}), \delta_0^{eg} \right)$

Confusion

Motion description: vergence \rightsquigarrow (listen \rightsquigarrow [shake head \parallel^{ss} speak-phrase]⁺) \rightsquigarrow diverge to neutral position \rightsquigarrow return head to neutral position

Encoding: $\left((seq, \delta_0^{sg}, [(\delta_1^s, \mu_1^{vergence}, \delta_1^e), (listen, (ss, \delta_1^{sg}, [(\delta_2^s, \mu_8^{shake}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg})^+], \right. \\ \left. (\delta_3^s, \mu_{neutral}^{-vergence}, \delta_3^e), (\delta_4^s, \mu_{neutral}^{-shake}, \delta_4^e)], \delta_0^{eg} \right)$

Discourage

Motion: vergence \rightsquigarrow [head-shake \parallel^{ss} speak-phrase] \rightsquigarrow return head to neutral

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{vergence}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_8^{shake}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg}), (\delta_3^s, \mu_{neutral}^{-shake}, \delta_3^e)], \delta_0^{eg})$

Disagreement

Motion: head-shake \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_8^{shake}, \delta_1^e), (\delta_2^s, \mu_{neutral}^{-shake}, \delta_2^e)], \delta_0^{eg})$

Defensive

Motion: vergence \rightsquigarrow [head-shake \parallel^{ss} speak-phrase] \rightsquigarrow return head

Encoding: $\left((seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{vergence}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{shake}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg}), \right. \\ \left. (ss, \delta_2^{sg}, [(\delta_3^s, \mu_{neutral}^{-vergence}, \delta_3^e), (\delta_3^s, \mu_{neutral}^{-shake}, \delta_3^e)], \delta_2^{eg}), \delta_0^{eg} \right)$

Defiance

Motion: [head-tilt \parallel^{sync} vergence] \rightsquigarrow raise head slightly \rightsquigarrow speak-phrase \rightsquigarrow return to neutral position

Encoding: $\left((seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{sync}, \delta_1^e), [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (\delta_2^s, \mu_1^{vergence}, \delta_2^e)], \delta_1^{eg}), (\delta_3^s, \mu_{abs}^{nod}, \delta_3^e), speakphrase(< phrase >)], \right. \\ \left. (ss, \delta_4^{sg}, [(\delta_5^s, \mu_{neutral}^{-nod}, \delta_5^e), (\delta_6^s, \mu_{neutral}^{-tilt}, \delta_6^e)], \delta_4^{eg}), \delta_0^{eg} \right)$

Denial

Motion: head-shake \rightsquigarrow speak-phrase \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(during, \delta_1^{sg}, [(\delta_2^s, \mu_8^{shake}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{sg}), (\delta_3^s, \mu_{neutral}^{-shake}, \delta_3^e)], \delta_0^{eg})$

Depressed

Motion: Lowering the head for a longer period

Encoding: $(\delta_1^s, \mu_{abs}^{-nod}, \delta_1^e)$

Dominate

Motion: head-tilt \rightsquigarrow (head-nod \parallel^{ss} vergence) \rightsquigarrow speak-phrase \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{nod}, \delta_2^e), (\delta_3^s, \mu_1^{vergence}, \delta_3^e)], \delta_1^{eg}), speakphrase(< phrase >)], (\delta_4^s, \delta_4^{sg}, [(\delta_5^s, \mu_{neutral}^{-nod}, \delta_5^e), (\delta_6^s, \mu_{neutral}^{-tilt}, \delta_6^e)], \delta_4^{eg}), \delta_0^{eg})$

Encourage

Motion: (head-nod $\parallel^{include}$ vergence) \rightsquigarrow speak-phrase \rightsquigarrow return to upright position \rightsquigarrow diverge eyes to neutral position

Encoding: $(seq, \delta_0^{sg}, (during, \delta_1^{sg}, [(\delta_1^s, \mu_3^{nod}, \delta_1^e), (\delta_2^s, \mu_1^{vergence}, \delta_2^e)], \delta_1^{eg}), speakphrase(< phrase >), (\delta_3^s, \mu_{neutral}^{-nod}, \delta_3^e), (\delta_2^s, \mu_{neutral}^{-vergence}, \delta_2^e), \delta_0^{eg})$

Expect

Motion: head-tilt \parallel^{ss} track

Encoding: $(ss, \delta_0^{sg}, [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (\delta_2^s, \mu_8^{\pm track}, \delta_2^e)], \delta_0^{eg})$

Frustrated

Motion: (head-shake randomly \parallel^{ss} vergence) \rightsquigarrow open mouth slightly

Encoding: $(seq, \delta_0^{sg}, (ss, \delta_1^{sg}, [(\delta_2^s, \mu_8^{random_shake}, \delta_2^e), (\delta_3^s, \mu_{abs}^{vergence}, \delta_3^e)], \delta_1^{sg}), (\delta_3^s, \mu_3^{-jaw}, \delta_3^e), \delta_0^{eg})$

Greet

Motion: head-tilt slightly \rightsquigarrow (head-nod \parallel^{ss} speak-phrase) \rightsquigarrow return to neutral position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{nod}, \delta_2^e), (\delta_3^s, \mu_{abs}^{vergence}, \delta_3^e)], \delta_1^{eg}), speakphrase(< phrase >)], (ss, \delta_4^{sg}, [(\delta_5^s, \mu_{neutral}^{-vergence}, \delta_5^e), (\delta_6^s, \mu_{neutral}^{-nod}, \delta_6^e)], \delta_4^{eg}), (\delta_7^s, \mu_{neutral}^{-tilt}, \delta_7^e), \delta_0^{eg})$

Include

Motion: head-tilt \rightsquigarrow (head-nod \parallel^{ss} speak-phrase) \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{-nod}, \delta_2^e), (\delta_3^s, \mu_1^{vergence}, \delta_3^e)], \delta_1^{eg}), speakphrase(< phrase >)], (\delta_4^s, \mu_{neutral}^{+nod}, \delta_4^e), (\delta_5^s, \mu_{neutral}^{-tilt}, \delta_5^e)], \delta_0^{eg})$

Interested

Motion: head-tilt \rightsquigarrow (head-up \parallel^{sync} speak-phrase) \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (sync, \delta_1^{sg}, [(\delta_2^s, \mu_3^{-nod}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg}), (\delta_3^s, \mu_{neutral}^{+nod}, \delta_3^e)], (\delta_4^s, \mu_{neutral}^{-tilt}, \delta_4^e)], \delta_0^{eg})$

Interject

Motion: head-up \rightsquigarrow speak-phrase \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_{abs}^{nod}, \delta_1^e), speakphrase(< phrase >)], (\delta_2^s, \mu_{neutral}^{-nod}, \delta_2^e)], \delta_0^{eg})$

Interrogate

Motion: head-tilt \rightsquigarrow (head-nod \parallel^{ss} speak-phrase) \rightsquigarrow return to upright position

Encoding: $(seq, \delta_0^{sg}, [(\delta_1^s, \mu_3^{tilt}, \delta_1^e), (ss, \delta_1^{sg}, [(\delta_2^s, \mu_3^{nod}, \delta_2^e), speakphrase(< phrase >)], \delta_1^{eg}), (\delta_3^s, \mu_{neutral}^{-nod}, \delta_3^e), (\delta_4^s, \mu_{neutral}^{-tilt}, \delta_4^e)], \delta_0^{eg})$

Permit

Motion: (slight lowering head \parallel^{ss} speak-phrase) \rightsquigarrow return to upright position

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(ss, \delta_1^{sg}, \left[\left(\delta_2^s, \mu_3^{nod}, \delta_2^e \right), speakphrase(< phrase > \right), \delta_1^{eg} \right], \left(\delta_3^s, \mu_{neutral}^{-nod}, \delta_3^e \right) \right], \delta_0^{eg} \right)$$

Persuade

Motion: (head-tilt \parallel^{sync} vergence) \rightsquigarrow (head-nod \parallel^{ss} speak-phrase) \rightsquigarrow return to neutral position

$$\text{Encoding: } \left(\left(seq, \delta_0^{sg}, \left[\left(sync, \delta_1^{sg}, \left[\left(\delta_2^s, \mu_3^{tilt}, \delta_2^e \right), \left(\delta_3^s, \mu_1^{vergence}, \delta_3^e \right) \right], \delta_1^{eg} \right), \left(ss, \delta_4^{sg}, \left[\left(\delta_5^s, \mu_{abs}^{nod}, \delta_5^e \right), \right. \right. \right. \right. \right. \\ \left. \left. \left. \left. \left. speakphrase(< phrase > \right), \delta_4^{eg} \right], \left(\delta_5^s, \mu_{neutral}^{-nod}, \delta_5^e \right), \left(\delta_6^s, \mu_{neutral}^{-tilt}, \delta_6^e \right) \right], \delta_0^{eg} \right) \right)$$

Plead

Motion: (head-lowered \parallel^{ss} speak-phrase) \rightsquigarrow return to neutral position

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(ss, \delta_1^{sg}, \left[\left(\delta_1^s, \mu_{abs}^{-nod}, \delta_1^e \right), speakphrase(< phrase > \right), \delta_1^{eg} \right], \left(\delta_2^s, \mu_{neutral}^{+nod}, \delta_2^e \right) \right], \delta_0^{eg} \right)$$

Question

Motion: head-tilt \rightsquigarrow (head-nod \parallel^{ss} speak-phrase) \rightsquigarrow return to upright position

$$\text{Encoding: } \left(\left(seq, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_{abs}^{tilt}, \delta_1^e \right), \left(ss, \delta_2^{sg}, \left[\left(\delta_3^s, \mu_{abs}^{nod}, \delta_3^e \right), speakphrase(< phrase > \right), \delta_1^{eg} \right], \delta_0^{eg} \right), \right. \right. \\ \left. \left. \left(ss, \delta_4^{sg}, \left[\left(\delta_5^s, \mu_{neutral}^{-nod}, \delta_5^e \right), \left(\delta_6^s, \mu_{neutral}^{-tilt}, \delta_6^e \right) \right], \delta_4^{eg} \right), \delta_0^{eg} \right) \right)$$

Reject

Motion: (repeated head-shake \parallel^{during} speak-phrase)

$$\text{Encoding: } \left(during, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_8^{shake}, \delta_1^e \right), speakphrase(< phrase > \right), \delta_0^{eg} \right)$$

Relax

Motion: (random head-nod \parallel^{conc} random head-shake \parallel^{conc} random-eyes)

$$\text{Encoding: } \left(conc, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_8^{random-nod}, \delta_1^e \right), \left(\delta_2^s, \mu_8^{random-shake}, \delta_2^e \right), \left(\delta_3^s, \mu_8^{random-eyes}, \delta_3^e \right) \right], \delta_0^{eg} \right)$$

Request

Motion: head-tilt \rightsquigarrow (repeated dampened nod \parallel^{sync} speak-phrase) \rightsquigarrow return to upright position

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_{abs}^{tilt}, \delta_1^e \right), \left(sync, \delta_2^{sg}, \left[\left(\delta_3^s, \mu_8^{nod}, \delta_3^e \right), speakphrase(< phrase > \right), \delta_2^{eg} \right], \left(\delta_4^s, \mu_{neutral}^{-tilt}, \delta_4^e \right) \right], \delta_0^{eg} \right)$$

Ridicule

Motion: (head-tilt \rightsquigarrow (repeated dampened nod \parallel^{ss} speak-phrase) \rightsquigarrow return head to upright position)

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_{abs}^{tilt}, \delta_1^e \right), \left(ss, \delta_2^{sg}, \left[\left(\delta_3^s, \mu_8^{nod}, \delta_3^e \right), speakphrase(< phrase > \right), \delta_2^{eg} \right], \left(\delta_4^s, \mu_{neutral}^{-tilt}, \delta_4^e \right) \right], \delta_0^{eg} \right)$$

Seek attention

Motion: (slight vergence \parallel^{ss} speak-phrase) \rightsquigarrow return to neutral position

$$\text{Encoding: } \left(\left(seq, \delta_0^{sg}, \left[\left(ss, \delta_1^{sg}, \left[\left(\delta_2^s, \mu_{abs}^{vergence}, \delta_2^e \right), speakphrase(< phrase > \right), \delta_1^{eg} \right], \left(ss, \delta_3^{sg}, \left[\left(\delta_4^s, \mu_{neutral}^{-vergence}, \delta_4^e \right), \right. \right. \right. \right. \right. \\ \left. \left. \left. \left. \left. \left(\delta_5^s, \mu_{neutral}^{-nod}, \delta_5^e \right), \delta_3^{eg} \right), \delta_0^{eg} \right) \right] \right) \right)$$

Submit

Motion: head-lowered \rightsquigarrow speak-phrase \rightsquigarrow return to neutral position

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_{abs}^{nod}, \delta_1^e \right), speakphrase(< phrase > \right), \left(\delta_2^s, \mu_{neutral}^{-nod}, \delta_2^e \right) \right], \delta_0^{eg} \right)$$

Unsure

Motion: head-tilt \rightsquigarrow (random eyes \parallel^{conc} open jaw slightly)

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(\delta_1^s, \mu_3^{tilt}, \delta_1^e \right), \left(conc, \delta_1^{sg}, \left[\left(\delta_2^s, \mu_1^{random-eye}, \delta_2^e \right), \left(\delta_3^s, \mu_1^{-jaw}, \delta_3^e \right) \right], \delta_1^{eg} \right), \delta_0^{eg} \right]$$

Veiled disagreement

Motion: (head-tilt \parallel^{ss} head-down) \rightsquigarrow close jaw

$$\text{Encoding: } \left(seq, \delta_0^{sg}, \left[\left(ss, \delta_1^{sg}, \left[\left(\delta_1^s, \mu_3^{tilt}, \delta_1^e \right), \left(\delta_2^s, \mu_3^{-nod}, \delta_2^e \right) \right], \delta_1^{eg} \right), \left(\delta_3^s, \mu_3^{-jaw}, \delta_3^e \right) \right], \delta_0^{eg} \right)$$

Figure 3: Declarative encoding of 34 gestures

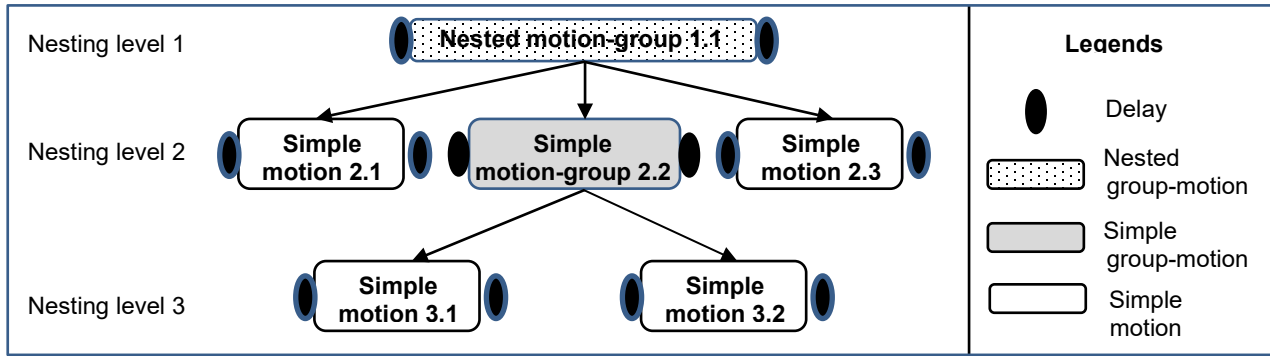


Figure 4: An illustration of a motion-tree for a gesture

newly actuated motion-group. The children threads are invoked based upon the synchronization-type. The parent thread is suspended using a spin-lock. The count of currently spawned children is kept in the dictionary D^E . Every time a child-thread is spawned, the corresponding count is incremented. The control returns to the parent-thread after all the children-threads have ended.

An algorithm for synchronization of motions is described in Figure 8. We achieve the synchronization of motions using semaphore locks. Each thread has the following Boolean flags: *startedMotion*, *endedMotion*, *duringEnded*, *startedFirst*, *startedRest*, *endedFirst*, and *endedRest*. The

suffix “First” denotes the first motion and the suffix “Rest” denotes the other motions in the group. Otherwise, the names are intuitive.

The motion flags are stored in the D^E using the key (*flag*, *MotionPath*). Initially, all the flags are set to *false*. The *startedMotion* flag is set to *true* after spawning the corresponding child-thread. To check that every thread spawning different motion has started, the values of the *startedMotion* flags are logically-ANDed under a critical section using the lock. To check that all the children threads have successfully terminated, the *endedMotion* flags of children-threads are logically-ANDed under a critical section using a lock.

```

Algorithm execute_gesture;
Input: 1. A gesture  $G = (gestureName, gestureTuple, speechFileName)$ 
Global: 1. A static dictionary  $D^G$  containing (gesture name, (gesture encoding, default speech files));
          2. A static dictionary  $D^M$  containing (motionName, (set of constraints, default values));
          3. A dynamic execution environment dictionary  $D^E$ ;
          4. A dynamic parameter dictionary  $D^P$  of the form (motion-name, motionPath, parameter-values);
          5. A speech-file SpeechFile;
          6. A lock threadLock for executing atomic operations among concurrently threads;

{  $D^E = \{ \}$ ; nestingLevel = 0; motionIndex = 1; motionCount = 1; syncType = null ;
  env = [(syncType, nestingLevel, motionIndex, motionCount)]; % initialize the environment
  gestureEncoding =  $\Pi_1(D^G(gestureName))$  % get the gesture encoding from the dictionary of gestures
  if ( speechFileName= 'default') speechFileName = pick_randomly( $\Pi_2(D^G(G))$ ); % pick default speech file
  cumulativeDelay = add_delays(gestureEncoding); speechTime = add_speech-delays(speechFile);
  scaleRatio = speechTime/ cumulativeDelay;
  gestureTuple = update_delays(gestureTuple, scaleRatio);
  /* start processing the gesture motions */
  if (length(gestureTuple) == 3)
    actuate_simple_motion(gestureTuple, gestureEncoding, motionPath, speechFile)
  elseif (length(gestureTuple) == 4) {
    push_env(env,  $D^E$ ); actuate_group_motion(gestureTuple, gestureEncoding, motionPath, speechFile);}
  else return failure % there is error in the gesture declaration;
}

```

Figure 5: Top level algorithm to actuate a gesture motion


```

function actuate_simple_motion % executes one organ motion with initial and final delays
Input: 1. motionTuple; 2. gestureEncoding; 3. motionPath; 4. speechFile
{
  initialDelay =  $\Pi_1$ (motionTuple) ; motionName =  $\Pi_2$ (motionTuple); finalDelay =  $\Pi_3$ (motionTuple);
  motionParameters = build_parameters(motionTuple, gestureEncoding, motionPath);
  sleep(initialDelay);
  thread motionThread = build_thread(motionName, motionPath, motionParameters); % build a thread
  spawn_thread(motionThread); % start the motion
  sleep(finalDelay);
  update_parameters(motionParameters, DP);
}
function build_parameters
input: 1. motionTuple; 2. gestureEncoding; 3. motionPath
output: motionParameters
{
  userParameters = get_parameters(motionTuple, gestureEncoding, motionPath);
  defuzzifiedParameters = defuzzify(userParameters); % defuzzify the parameters
  motionName =  $\Pi_2$ (motionTuple);
  defaultParameters = get_parameters(motionName, DM); % extract the default parameters
  motionParameters = defuzzifiedParameters  $\cup$  defaultParameters;
  return(motionParameters)
}

```

Figure 6: An algorithm to execute simple motion

6 Implementation

A prototype has been implemented on an upper body-part of an artificial human skeleton using Python language and Adafruit motor-drivers running on a Raspberry Pi 3B. Head-movements have been realized using a combination of stepper and servo motors as illustrated in Figures 9 and 10.

The reduction in jerky motions of motors and the stability of the organs' postures has been realized using additional coiled springs inspired by deep muscles in a neck, jaws and back of the head, and are illustrated in Figure 11. Table 3 shows the correspondence between springs and human muscles along with their functions. We have used coiled springs to 1) achieve graceful dampening effect to minimize jerky motion, and 2) simulate the function of the deep neck muscles to stabilize the head. The following muscles of a human body [28, 33] were emulated to achieve the movements: 1) *sternocleidomastoids* to balance a head during tilt using the springs S_1 and S_2 ; 2) *semispinalis capitis* to pull the head back during neutral position using springs S_3 and S_4 ; 3) *sub occipitals* to extend and rotate the head emulated by the spring S_5 ; 4) *masseters* to move the jaw during talking or chewing using the springs S_6 and S_7 .

The placement of eyes in the eye-sockets is shown in Figure 12(a). The schematics for servo motor-placement needed for the synchronized eye-movements is shown in Figure 12(b). The motors are fixed to the skull such that the eyes rotate around the x-axis in the sockets. Eye-balls were connected from motor-stem using a small coiled spring to get realistic flexible movements as

shown in Figure 12(b). Rolling of the eyes, a minor movement around the y-axis, is not provided due to limited space in the socket. However, it will be emulated using flexinol in the future.

We generate speech using a Python library function for Raspberry Pi that renders a .wav file. The .wav files are indexed on the speech file name given in the gestures. In the absence of any specific speech file, a default file associated with the corresponding gesture is picked randomly and rendered.

7 Future Work

We are extending our system to integrate: 1) gestures based upon coordinated movements of hand, torso and spine; and 2) automated prediction of gestures and head-motion based upon speech data and dialog response. We are also developing machine learning tools for a humanoid to fine-tune archived gesture timings and actions by observing a human repeating the gestures.

References

- [1] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," *Communications of the ACM*, 26(11):832-843, November 1983.
- [2] S. Andrist, X. Z. Tan, M. Gleicher and B. Mutlu, "Conversational Gaze Aversion for Humanlike Robots," *Proceedings of the 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Bielefeld, Germany, pp 25-32, March 2014.

```

function actuate_group_motion
Input: 1. motionTuple; 2. gestureEncoding; 3. groupPath; 4. speechFile;
{ syncType =  $\Pi_1$ (MotionTuple); initialDelay =  $\Pi_2$ (motionTuple) ; motionGroup =  $\Pi_3$ (motionTuple);
  finalDelay =  $\Pi_4$ (motionTuple); motionCount = length(motionGroup); endedMotions = false;
  nestingLevel =  $\Pi_1$ (first(groupPath)); % extract the nested level from the first element of the path
  motionIndex =  $\Pi_2$ (first(groupPath)); % get the motion-index of the current motion
  sleep(initialDelay); % sleep for initial motion-group delay
  acquire(threadLock); nestingLevel = nestingLevel + 1; release(threadLock);
  If (syncType  $\in$  {sync, ss, es, conc}) % execute threads concurrently
  { threadList = < >; index = 0;
    while (index < motionCount){
      motionPath = concatenate((nestingLevel, index), groupPath);
      if ( $\Pi_3$ (motionTuple[index]) == 3){ % if it is a simple motion
        motionParameters = build_parameters(motionName, gestureEncoding, motionPath);
        thread newThread = build_motion_thread(motionTuple, motionParameters);
        acquire(threadLock)
          threadList = threadList + newThread; index += 1; store_thread( $D^E$ , newThread);
        release (threadLock);
      else { % the motion is an embedded motion-group
        newMotionTuple =  $\Pi_3$ (motionTuple)[index];
        acquire(threadLock) {
          env = (syncType, nestingLevel, index, motionCount);
          push_env(  $D^E$ , env); % dump the current environment in execution-environment dictionary
          release(threadLock);}
        actuate_motion_group(newMotionTuple, motionPath, gestureEncoding, speechFile);}
      actuate_synchronized_concurrent_movements(syncType, groupPath, threadList); %
      if (syncType  $\in$  {sync, ss}) while ( $\neg$  started_all_threads(groupPath)) snooze() % spinlock parent
      if (syncType  $\in$  {sync, es}) while ( $\neg$  ended_all_threads(groupPath)) snooze() % spinlock parent
      if ( syncType  $\in$  {during}) {
        while ( $\neg$  ended_included_threads(groupPath)) snooze(snoozeTime); % spinlock for first thread
        while( $\neg$  ended_first_thread(groupPath)) snooze(snoozeTime);} % spinlock for the parent-thread
        acquire(threadLock) pop_env( $D^E$ , env); release(threadLock);}
    else {% execute threads sequentially
      forall (index < motionCount) {
        newMotionTuple =  $\Pi_3$ (motionTuple)[index];
        if (length(motionTuple) == 3){ % if it is a simple motion
          motionParameters = build_parameters(motionName, gestureTuple, motionPath);
          thread motionThread = build_motion_thread(motionTuple, motionParameters);
          spawn(motionThread);}
        else{ push_env( $D^E$ , env);
          actuate_motion_group(newMotionTuple, env, gestureTuple, speechFile);
          (syncType, nestingLevel, index, motionCount) = pop_env( $D^E$ );} }}
      acquire(threadLock); endedMotionFlag[(nestingLevel, motionIndex)] = True; release(threadLock); }
    sleep(finalDelay);}

```

Figure 7: An algorithm for executing a motion-group

```

function actuate_synchronized_concurrent_movements
Input: 1. Synchronization type syncType; 2. threadPairList as a list of pairs (motionPath, motionThread);
{
  i = 1; n = length(threadList); startedMotions = false; endedMotions = false;
  duringEnded = True; startedFirst = false; startedRest = false; endedFirst = false; endedRest = false;
  snoozeTime = 1 millisecond;
  acquire(threadLock); forall (1 ≤ i ≤ n) {motionStartedFlags[i] = false; motionEndedFlags[i] = false;
  release(threadLock);
  i = 1;
  while (¬ empty(threadInfoList)) {
    nextThreadPair = get_next_pair(threadPairList); motionPath = Π1(nextThread);
    curThread = Π1(nextThread); threadInfoList = rest(motionList);
    if ( (i == 1) ∨ ((syncType == 'during') ∧ startedFirst) ∨ (syncType ∈ { ss, es, sync})){
      spawn_thread(curThread);
      acquire(threadLock)
      set_startedMotionFlag(MotionPath, True); % set the startedmotionFlag in  $D^E$  as true
      if (i == 1) startedFirst = true;
      release(threadLock);} }
  if (syncType ∈ {'ss', 'sync'}) {
    while (¬ startFlags){ % spinlock checking for started threads to be true
      acquire(threadLock)
      startedFlags = false; startedFlags = and_flags(started, groupPath);
      release(threadLock);
      snooze(snoozeTime); } % snooze before starting again
    if (syncType == 'ss') return;
  }
  if (syncType ∈ {'es', 'sync'}) {
    while (¬ endedFlags) {
      acquire(threadLock)
      endedFlags = false; endedFlags = and_flags(started, groupPath);
      release(threadLock)
      snooze(snoozeTime) ;} % snooze before starting again
    return ;}
  if (syncType == 'during' ∧ startedFirst)
    while (¬ startedRest) { % spinlock checking for started threads to be true
      acquire(threadLock)
      startedRest = false; startedRest = and_flags(during, groupPath) ;}
      release(threadLock);
      snooze(snoozeTime);} % snooze before checking again
    while (¬ endedRest){ % spinlock checking for started threads to be true
      acquire(threadLock)
      endedRest = false; startedRest = and_flags(during, groupPath);
      release(threadLock)
      snooze(snoozeTime);} % snooze before checking again
    while (¬ endedfirst) snooze(snoozeTime); % snooze before checking again
    return;
  }
}

```

Figure 8: Algorithm for spawning synchronized motions

- [3] O. Aran, H. Hung, and D. Gatica-Perez, "A Multimodal Corpus for Studying Dominance in Small Group Conversations," *Proceedings of the Workshop on Multimedia Corpora: Advances in Capturing, Coding and Analyzing Multimodality*, Valetta, Malta, pp 22 -26, May 2010.
- [4] G. Ball and J. Breese, "Relating Personality and Behavior: Posture and Gestures," *Proceedings of the International Workshop on Affective Interactions (IWAI 1999)*, (Editor: A. Paiva), Springer, Heidelberg, Germany, LNCS 1814:196-203, 2000.
- [5] A. Bartl, S. Bosch, M. Brandt, M. Dittrich, and B. Lugrin, "The Influence of a Social Robot's Persona on How it is Perceived and Accepted by Elderly Users," *Proceedings of the 8th International Conference on Social Robotics*, Kansas City, MO, USA, pp 681-691, November 2016.
- [6] C. Breazeal, "Social Robots: From Research to Commercialization," *Proceedings of the 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Vienna, Austria, pp 1, March 2017.

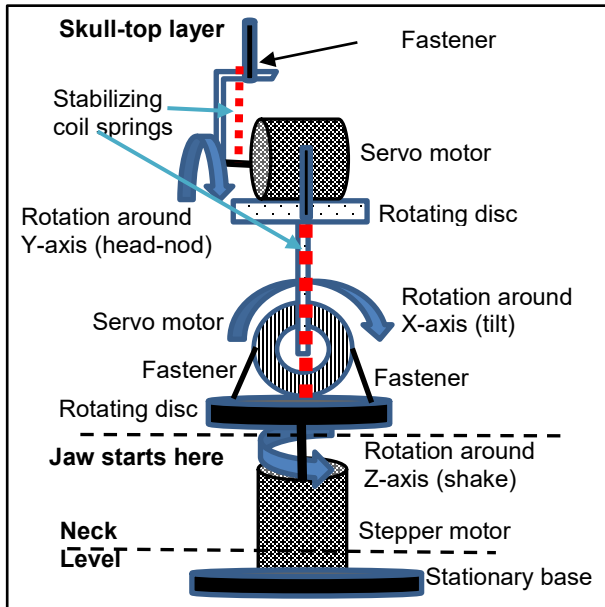


Figure 9: A schematic of head-motion

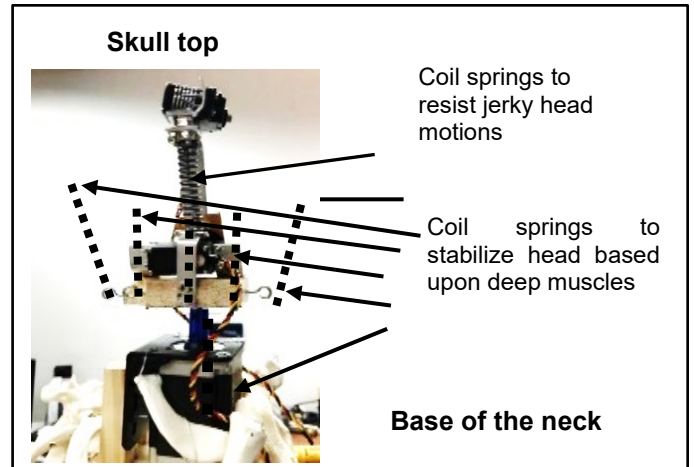


Figure 10: Implementation of head motion

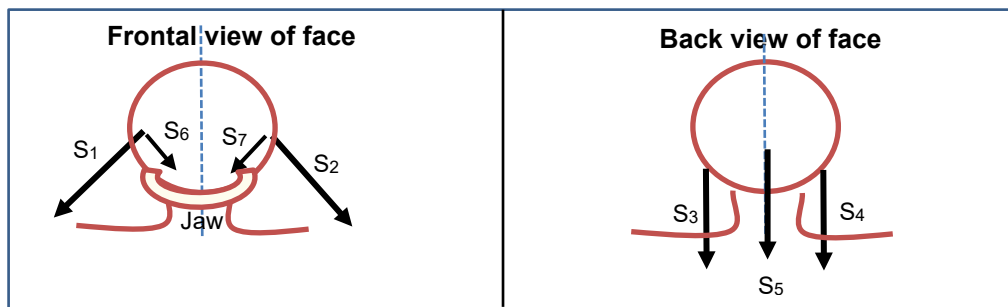


Figure 11: Implementation of head motion

Table 3: Muscles emulated for head and jaw movement and stability

Spring id	Muscle	Function
S ₁ and S ₂	sternocleidomastoid	Stabilizes tilt movement. Rotates head to the opposite side
S ₃ and S ₄	semispinalis capitis	Pulls head back. Rotates head in the same direction
S ₅	sub occipitals	Extending and rotating the head
S ₆ and S ₇	masseter	Moves jaw for talking or chewing

[7] V. Berenz, F. Tanaka, K. Suzuki and M. Herink, "TDM: A Software Framework for Elegant and Rapid Development of Autonomous Behaviors for Humanoid Robots," *Proceedings of the 11th IEEE-RAS International Conference on Humanoid Robots*, Bled, Slovenia, pp 179-186, October 2011.

[8] P. Bremner, A. Pipe, C. Melhuish, M. Fraser, and S. Subramanian, "Conversational Gestures in Human-Robot

Interaction," *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, TX, USA, pp 1645-1649, October 2009.

[9] F. Broz and H. Lehmann, "A Gaze Controller for Coordinating Mutual Gaze during Conversational Turn-taking in Human-Robot Interaction," *Extended Abstracts in ACM Conference on Human Robot Interaction (HRI)*, Portland, Oregon, USA, pp 131-132, March 2015.

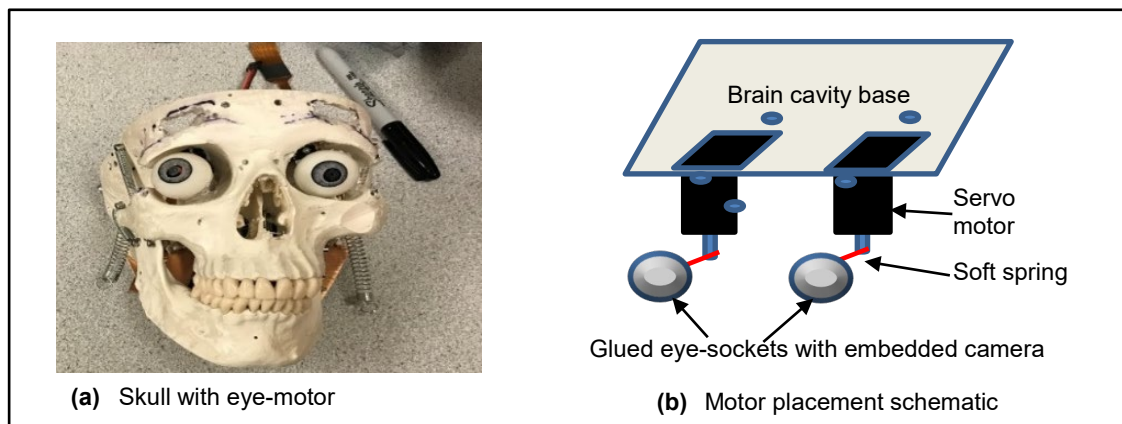


Figure 12: Implementation of eye motion

- [10] J. Cabibihan, W. So and S. Pramanik, "Human-Recognizable Robotic Gestures," *IEEE Transactions on Autonomous Mental Development*, 4(4):305-314, December 2012.
- [11] B. Chandrasekaran and J. M. Conrad, "Human-Robot Collaboration: A Survey," *Proceedings of the Southeast Con*, Fort Lauderdale, FL, USA, pp 1-8, April 2015.
- [12] C. F. DiSalvo, F. Gemperle, J. Forlizzi, and S. Kiesler, "All Robots are not Created Equal: the Design and Perception of Humanoid Robot Heads," *Proceedings of the 4th ACM Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, London, UK, pp 321-326, June 2002.
- [13] B. R. Duffy, "Anthropomorphism and the Social Robot," In *Robotics and Autonomous Systems*, 42(3/4):177-190, March 2003.
- [14] Dynamixel Servo motor, <http://www.Trossenrobotics.com/c/robotis-dynamixel-robot-servos.aspx>, 2018, last accessed March 13, 2019.
- [15] K. Erdoğan, H. Kömür, A. Durdu and R. Ceylan, "Human Robot Interaction to Guide a Person," *Proceedings of the 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey, pp 1-4, October 2018.
- [16] N. Esteve-Gibert, J. Borràs-Comes, E. Asor, M. Swerts, and P. Prieto, "The Timing of Head Move-ments: the Role of Prosodic Heads and Edges," *Journal of Acoustics Society of America*, 141(6):4727-4739, June 2017.
- [17] H. Fadli, C. Machbub, and E. Hidayat, "Human Gesture Imitation on NAO Humanoid Robot using Kinect based on Inverse Kinematics Method," *Proceedings of the International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, Surabaya, Indonesia, pp 116-120, October 2017.
- [18] M. Ghayoumi, M. Thafar, and A. K. Bansal, "Towards Formal Multimodal Analysis of Emotions for Affective Computing," *Proceedings of the International Conference on Distributed Multimedia Systems*, Salerno, Italy, pp 48-54, November 2016.
- [19] D. F. Glas, T. Minato, C. T. Ishi, T. Kawahara, and H. Ishiguro, "ERICA: The ERATO Intelligent Conversational Android," *Proceedings of the 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, New York, NY, USA, pp 22-29, August 2016.
- [20] B. Goertzel, J. Mossbridge, E. Monroe, D. Hanson, and G. Yu, "Loving AI: Humanoid Robots as Agents of Human Consciousness Expansion," available at <https://arxiv.org/pdf/1709.07791.pdf>, last accessed March 7, 2019.
- [21] S. Goldin-Meadow, *Hearing Gesture: How our Hands Help us Think*, Harvard University Press, Boston, MA, USA, 2005.
- [22] D. Hanson, "Progress Toward EAP Actuators for Biomimetic Social Robots," *Proceedings of the SPIE 8687, Electroactive Polymer Actuators and Devices (EAPAD)*, San Diego, CA, USA, DOI: doi: 10.1117/12.2014238, April 2013.
- [23] J. Hirth and K. Berns, "Concept for behavior generation for the Humanoid Robot Head ROMAN Based on Habits of Interaction," *2007 7th IEEE-RAS International Conference on Humanoid Robots*, Pittsburgh, PA, pp. 360-365, 2007.
- [24] Y. Hu, Z. Zhao, A. Vimal and G. Hoffman, "Soft Skin Texture Modulation for Social Robotics," *Proceedings of the IEEE International Conference on Soft Robotics (RoboSoft)*, Livorno, Italy, pp 182-187, April 2018.
- [25] C. T. Ishi, C. Liu, H. Ishiguro and N. Hagita, "Head Motion during Dialogue, Speech and Nod Timing Control in Humanoid Robots," *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Osaka, Japan, pp 293-300, March 2010.
- [26] C. T. Ishi, D. Machiyashiki, R. Mikata and H. Ishiguro, "A Speech-Driven Hand Gesture Generation Method and Evaluation in Android Robots," *Proceedings of the IEEE Robotics and Automation Letters*, 3(4):3757-3764, October 2018.
- [27] M. Johansson, T. Hori, G. Skantze, A. Höthker, and J. Gustafson, "Making Turn-Taking Decisions for an Active

- Listening Robot for Memory Training,” *Proceedings of the 8th International Conference on Social Robotics*, Kansas City, MO, USA, pp 940-949, November 2016.
- [28] A. Kendon, *Gesture: Visible Action as Utterance*, Cambridge University Press, Cambridge, UK, 2004.
- [29] A. Kim, H. Kum, O. Roh, S. You and S. Lee, “Robot Gesture and User Acceptance of Information in Human-robot Interaction,” *Proceedings of the 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, MA, USA, pp 279-280, March 2012.
- [30] M. Kipp, M. Neff, K. H. Kipp, and I. Albrecht, “Towards Natural Gesture Synthesis: Evaluating Gesture Units in a Data-driven Approach to Gesture Synthesis,” *Proceedings of the 7th International Working Conference on Intelligent Virtual Agents*, Paris, France, Springer, Berlin, Heidelberg, LNCS 4722:15-28, September 2007.
- [31] D. Kragic, “Acting, Interacting, Collaborative Robots,” *Proceedings of the 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Vienna, Austria, pp 293-293, March 2017.
- [32] C. Liu, C. T. Ishi, H. Ishiguro and N. Hagita, “Generation of Nodding, Head-tilting and Eye gazing for Human-robot Dialogue Interaction,” *Proceedings of the 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Boston, MA, USA, pp 285-292, March 2012.
- [33] D. McNeill, *Hand and Mind: What Gestures Reveal about Thought*, The University of Chicago Press, Chicago, IL, USA, 1992.
- [34] J. Oh, D. Hanson, W. Kim, Y. Han, J. Kim and I. Park, “Design of Android Type Humanoid Robot Albert HUBO,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, pp 1428-1433, October 2006.
- [35] I. Poggi, F. D’Errico, L. Vincze, “Types of Nods. The Polysemy of a Social Signal,” *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, Malta, pp 2570-2576, May 2010.
- [36] A. Roberts, *The Complete Human Body: The Definitive Visual Guide*, 2nd Edition, Dorling Kindersley Publishing, New York, NY, USA, 2016.
- [37] M. Salem, S. Kopp, I. Wachsmuth, and F. Joubin, “Towards Meaningful Robot Gesture,” *Human Centered Robot Systems: Cognitive Systems Monographs*, H. Ritter, G. Sagerer, R. Dillmann, and M. Buss (eds.), Springer, Berlin, Germany, 6:173-182, 2009.
- [38] Sophia-Hanson Robotics Latest Humanoid, <http://www.hansonrobotics.com/robot/sophia/>, 2018, last accessed March 13, 2019.
- [39] L. Stecco, *Atlas of Physiology of the Muscular Fascia*, Piccin Publications, Padua, Italy, September 2016.
- [40] J. Stolzenwald and P. Bremner, “Gesture Mimicry in Social Human-Robot Interaction,” *Proceedings of the 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Lisbon, Portugal, pp 430-436, August/September 2017.
- [41] F. Vannucci, G. Di Cesare, F. Rea, G. Sandini and A. Sciutti, “A Robot with Style: Can Robotic Attitudes Influence Human Actions?,” *Proceedings of the IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, Beijing, China, pp 1-6, November 2018.
- [42] J. Wainer, D. J. Feil-seifer, D. A. Shell and M. J. Mataric, “The Role of Physical Embodiment in Human-Robot Interaction,” *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication*, Hatfield, UK, pp 117-122, September 2006.
- [43] Y. Wu, R. Wang, Y. L. Tay and C. J. Wong, “Investigation on the Roles of Human and Robot in Collaborative Storytelling,” *Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Kuala Lumpur, Malaysia, pp 63-68, December 2017.



Aditi Singh is a doctoral student and a graduate assistant in the Department of Computer Science at Kent State University. Aditi completed M.S. in Computer Science from Kent State University in 2017. Her thesis area was “gesture generation in humanoids”. Her doctoral research interest is in social robotics, human-humanoid interactions, and speech-gesture correlation. Aditi is a member of IEEE and is very active in ACM activities within the department.



Arvind Bansal is a full Professor in the Department of Computer Science at Kent State University, Kent, Ohio, USA. He did his B. Tech (Electrical Engineering) and M. Tech (Computer Science) from Indian Institute of Technology, Kanpur, India in 1979 and 1983 respectively. He finished his Ph.D. in Computer Science from Case Western Reserve University in 1988. He is a member of IEEE and ACM. He has contributed in multiple research areas such as fault-tolerant multiagent systems genomics, proteomics, parallel and associative logic programming, machine learning, massive parallel knowledge bases multimedia systems, programming languages for multimedia systems, ECG analysis, health informatics and social robotics including facial-expression analysis, gesture analysis and gesture generation. He has been active in a large number of program committees in these research areas and has written a textbook for design concepts in programming languages.

Generation of Audiovisual Materials Considering Semantic and Impressive Harmony Based on Time Change of Music

Yuto Shinjo, Teruhisa Hochin* and Hiroki Nomiya*
Kyoto Institute of Technology, Kyoto-Shi, Kyoto, 606-8585 JAPAN

Abstract

A method to detect change points of the impression of a music piece has been proposed for retrieving pictures having the impression similar to that of a music piece. This method can detect the change points of the impression. However, when the change repeatedly occurred continuously, and when the impression was felt as a group, there was sometimes a change point in a place that the change of impression could not be felt. We aim at slideshow generation which copes with this subject point and considers semantic and impressive harmony based on time changes of music impression. This paper proposes a method to cope with the cases when impression changes occur repeatedly, a method to generate an HTML file in which the slideshow is displayed by acquiring images matching each impression of the music pieces when entering the music piece, and a method of measuring semantic similarity between slideshow images. The proposed methods use factor scores of the impression of images and music pieces. It is shown that the proposed method can improve the accuracy of the change point detection and the operation of the current system.

Key Words: Change of pictures, impression of music, impression change, slideshow, meaning of image, multiple regression analysis, factor score, image recognition, word similarity.

1 Introduction

In recent years, due to the development of multimedia technology, multimedia data such as images, music pieces and videos are used in various fields. In order to effectively and efficiently prepare multimedia works, a mechanism for searching harmonized images and music pieces is required. Various researches have been conducted to the mutual retrieval of multimedia data based on impressions [4-5, 12-16], to the retrieval of image data based on impressions [2-3, 6, 8, 11], and to the retrieval of audio data based on impressions [7, 9-10]. In the mutual retrieval of multimedia data based on impression, for example, pictures are retrieved by specifying a music piece as a retrieved key. The retrieved pictures have impressions similar to that of the music piece specified.

Here, there are many music pieces whose impression changes

in time, but mutual retrieval system based on the impression for the multimedia data on the Web does not consider temporal change of the impression of a music piece. In order to be able to take that into consideration, research is being conducted to detect temporal changes in the impression of music piece [13]. However, in the detection method of the change point of the impression of the music piece, there were improvement points such as the same change repeated and the impression was detected as a change point at a place where the impression was felt as a group. In addition, when displaying an image that matches the impression of a music piece that changes over time, a method of using a slideshow that presents an image while reproducing the music piece is conceivable, and it is necessary to consider whether the images used in the slideshow are in harmony with each other.

This paper proposes a method to cope with the cases when impression repeatedly change, a method to generate an HTML file, in which slideshow is displayed by acquiring images matching each impression of the music pieces when entering the music piece, and a method of measuring semantic similarity between slideshow images. The proposed methods use factor scores of the impression of audio materials and images. This paper shows that the accuracy of the change point detection is improved subjectively by the proposed method, and describes the operation of the current system.

The remainder of the paper is organized as follows. Section 2 describes a mutual retrieval system based on impressions [15]. Section 3 describes a method of detecting the change points of the impressions of a music piece [13]. Section 4 proposes a method that does not detect consecutively repeated changes as change points and a method of retrieving images matching each section, and a method of automatically generating a slideshow and a method to measure semantic similarity between images. Section 5 shows a system execution result. Finally, Section 6 concludes the paper.

2 Mutual Retrieval of Visual and Audio Materials Based on Impression

In the previous study [15], a mutual retrieval system based on the impression for the multimedia data on the Web is realized. This system consists of the information collecting subsystem and the mutual retrieval subsystem. In the information collecting subsystem, the information of the multimedia data on the Web is analyzed and is stored into a database. The mutual

* Information and Human Sciences. Email: m7622020@edu.kit.ac.jp, {hochin, nomiya}@kit.ac.jp

retrieval subsystem uses this information for the retrieval of heterogeneous media data with similar impression.

In this research, mutual retrieval with similar impressions is performed for images and music pieces. Brightness, potency, activity, naturalness, and sharpness are used as factors of the image, while naturalness, brightness, potency, and sharpness are used as factors of the music piece. The Euclidean distance of the factor scores of each factor is used as dissimilarity. Here, since the factor of activity in the image does not exist in the music piece, the smaller Euclidean distance between the factor scores of the image's potency and activity, and the factor score of the music piece's potency is taken as dissimilarity. In the database, information such as the URL, feature values, factor scores, etc. of each piece of multimedia data is stored.

In addition, factor scores are estimated from the feature values of multimedia data by using the multiple regression analysis. The explanatory variables of the estimation formula of the factor score of the image are the feature values of the HSV system closely related to the human sense. In the feature value extraction, classification of value, saturation, and hue is performed for each pixel, and the feature value is obtained by calculating the ratios of the number of the pixels of each value, saturation, and hue to that of the whole pixels.

The explanatory variables of the estimation formula of the factor score of the music piece relate to the average and the standard deviation values of spectrum information and Mel frequency cepstrum coefficient (MFCC), the ratio of low energy in the whole audio file and the frequency spectrum, and the peak calculated on the basis of frequency spectrum. In the feature value extraction, average values of respective spectrum information are calculated for every 512 sampling points, and the mean value and the standard deviation of them are obtained.

A general flow of the mutual retrieval system is as follows. First, feature value extraction and factor score estimation are performed on a piece of multimedia data which is a retrieval key specified by the user. Then, dissimilarities of the factor scores of the data specified and the data stored in the database are obtained. Heterogeneous multimedia data which become retrieval results are stored in the ascending order of dissimilarity, and their URLs are returned.

3 Detecting Changes of Music Impressions

In the previous study [13], in order to be able to take into account the temporal change when searching images from songs whose impressions change with time, change points of impressions of the songs were detected. It was subjectively shown that a change in impression was felt at the detected change point.

In this study, factor scores of songs similar to the previous study [15] are used. The songs are divided into sections. After the MFCC mean and standard deviation for each section are obtained, factor scores are estimated in each section.

The change of the impression of a music piece is decided by using the Euclidean distance between factor scores estimated in

each section. Here, the following two points were used.

- Distance between the factor scores in the current and the next sections
- Distance between the factor score at the starting point of the current impression and the factor score in each subsequent section

The latter is to make it possible to cope with a music piece whose impression gradually changes.

As a concrete example, change point detection for classic music "Marriage of Figaro" overture was reported [13]. A 60 second music piece is divided into sections of 2.5 seconds, and change points are detected based on the distances of factor score values in each section. Here, the threshold value of the Euclidean distance used in detecting the impression change is set to 8.

When examining the change points detected when reproducing the music pieces, the impression change was felt at the point where the change point was detected. However, there were cases where no change was felt at the place where the change point of the impression was continuously detected. Changes were felt at the first change point of consecutive ones, but the subsequent impression was felt as a collective impression and no change was felt.

4 Proposed Method

4.1 Consideration of Impressive Harmony

4.1.1 Deletion Method of Continuous Detection Part: From the previous study [13], when comparing only within the section judged by the program, a difference in impression was felt. It is thought that the same change was repeated when playing through music, and it felt like a group. Therefore, we propose a method of judging that impressions are repeated, and deleting the change points if they are continuously detected and have similar impressions before and after the change [14]. In the range search performed at the mutual retrieval, it is assumed that the impression is similar if the value of each factor score is within the range of -4 to 4. For this reason, if the range is within the range of -4 to 4, the impressions appearing before and after the change are determined that they are similar to each other.

4.1.2 Retrieval of Pictures Matching Each Section: Next, for each impression section divided by the change point of impression obtained, search for images with matching impressions. The initial section of each impression is provisionally set as the score of the impression of the section. We perform a search using the distance between that score and the factor score of the picture in the database. For the score and the search method used for the search, the same method as in the previous study [15] is used. This method retains the factor scores of each impression while detecting change points and performs a retrieval after completion of detection.

4.2 Consideration of Semantic Harmony

4.2.1 Getting Meaning of Image: The factor score of the image is estimated from the color information of the image. The meaning part of the image such as “what is reflected in the image” is not considered. Therefore, by using image recognition, what is reflected in the image is acquired as a word, and it is considered as the meaning of the image. For that purpose, execute “classify_image.py” according to Tensorflows tutorial [17] to perform image recognition. This can be done by downloading a trained model from tensorflow.org at the first execution and specifying an image to be recognized as a parameter. Here, the words obtained are in English.

4.2.2 Similarity Measurement between Words: If the obtained images are semantically similar, the slideshow is considered to have semantic harmony. Therefore, it is possible to determine how semantically the slideshow is harmonized by measuring the similarity between the acquired words and performing image recognition on each of the obtained images. For that purpose, the similarity between words is calculated by specifying the words in “model.similarity” of word2vec using the published learned vector “GoogleNews-vectors-negative300.bin” [1].

4.3 Automatic Generation of Slideshow

This system creates an HTML file to display search results. It acquires the list of music piece names and search results used for the search and writes them in the file together with the fixed form sentences. It is an implementation that acquires the playing time of a music piece and changes the URL of the picture each time when the time passes the time of changing point. A file is generated continuously following change point detection/file search.

5 Example of System Operation

5.1 Detecting Change Point of Impression

We performed with the classical music piece titled “Le cygne” used in the previous study [15]. First, we describe improvement of accuracy of change point detection. Table 1 shows the transition of factor scores of this music piece and the detection points of change points. The length of the music piece is 60 seconds. Here, a section has 430 spectrum calculation points. One calculation point corresponds to 512 sample points. The value of the factor score in each section represents the impression in about 2.5 seconds. The threshold of the Euclidean distance of change point detection is eight as in the previous study [13].

The distance between the section numbers 7 and 8 shown in Table 1 is the square root of the sum of the squares of the factor score differences in the preceding and the following sections, which is 13.29. Since it exceeded the threshold value, it was detected as a change point. The same applies to the change point between section numbers 18 and 19. In addition, since the Euclidean distance between section number 8, which is the

beginning of the second impression section, and the section number 19 also exceeds eight, this change point satisfies both of the two conditions.

There are other places where the Euclidean distance of the two conditions exceeds eight, but they are not detected as change points. This is based on the proposed method. For example, the Euclidean distance between the factor scores between the section numbers 8 and 9 is 11.00. At this time, the Euclidean distance between the section numbers 7 and 8 also exceeds the threshold value, and the impression continuously changes. Here, when the impression before the last change (section number 7) and the impression after the current change (section number 9) are similar, it is judged that it is a group of impressions and the current change (the change between the section numbers 8 and 9) is not detected as a change point. In the same way, although the Euclidean distance continuously exceeds the threshold value between the section numbers 9 and 10 to between the section numbers 12 and 13, similar impressions appear repeatedly in the section numbers 8 and 10, the section numbers 9 and 11, it is judged as a collective impression, and it is not detected as a change point.

When we examine the detected change points by playing the music piece, we can feel the change of impression around the time when the change points are detected. Regarding the deleted continuous change part, it was a collective impression and we felt little change, so it was confirmed that the proposed method works well.

5.2 System Operation

Next, we describe the operation of the system when searching, using the song described in 5.1. This system is a modification of the are mutual search system created in the previous study [15]. Figure 1 shows the system screen after selecting the music file and pressing the search button.

Since we are dealing with searching for images from music pieces this time, the radio button “Image retrieval” is pressed for “Select Retrieval Type.” By clicking the “Browse” button, we can select the file and the selected file will be displayed in the text box at the top of the “Browse” button. In this example, an audio file 06.wav (classical song titled “Le cygnet”) is selected. When the “Retrieve File” button is pressed, search is executed. In this case, the change point detection of the impression is performed on the selected music piece first, and the search is executed for each impression section delimited by the detected change point. Since two change points are detected, there are three impression sections, and three images matching each of them are obtained as search results. When the result is displayed, the URL of the image matching the first impression section of the music piece is displayed in the combo box under “Retrieval Results,” and the image is displayed in “Selected Image” on the right of the screen. Also, pressing the pull-down button will also display the URLs of the images matching the second and subsequent impression sections in order. And, the image of the selected URL is displayed in “Selected Image. “Retrieval Num” at the upper center of the screen can determine the number of search result candidates. However, since this time we have

Table 1: Factor scores and change points

Section number	Naturalness	Brightness	Potency	Sharpness	Distance
1	-1.72	0.17	0.02	-0.92	
2	-2.21	1.80	-3.22	-0.29	
3	-0.71	3.86	-4.69	-1.20	
4	-0.44	-2.16	-1.40	-0.78	
5	0.61	-3.48	-2.29	-3.51	
6	0.05	2.31	-3.80	-1.99	
7	-2.04	5.28	-2.47	0.40	
Change point					13.29
8	-1.11	-5.48	0.39	-6.79	
9	-1.75	3.34	-4.27	-2.19	
10	-1.43	-9.26	3.10	-6.68	
11	-1.46	1.53	-1.86	-1.47	
12	-2.28	-6.27	0.97	-6.30	
13	-1.01	2.26	-4.04	-1.62	
14	-0.62	-0.64	-1.96	-1.42	
15	0.64	-1.04	-1.50	-2.06	
16	-1.97	-4.03	-1.11	-4.57	
17	-0.17	-6.39	1.79	-5.25	
18	-1.61	-4.92	0.20	-3.80	
Change point					8.11
19	-2.08	2.07	-1.99	-0.35	
20	-3.47	-6.46	1.29	-2.95	
21	-0.11	-2.92	-0.10	-4.09	
22	-1.55	0.58	-0.64	-2.05	
23	-1.74	0.38	-0.47	-2.24	
24	-1.43	-3.25	-0.83	-3.71	

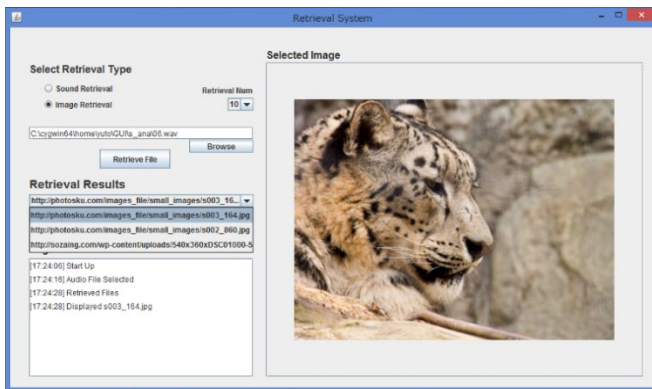


Figure 1: System screen at search execution

acquired only one that was judged to be the best match to create a slide show, this value does not affect the operation.

5.3 Generated Slideshow

Next, we describe the generated slideshow. The HTML file

that displays the slideshow is generated at the same time when search is executed by pressing the "Retrieve File" button of the system. The slideshow screen automatically generated at this time is shown in Figure 2. In the slide show, the input audio file name and the impression section number (the number of display images) of the music piece are displayed. Music pieces can be played and stopped with the "play" and the "stop" buttons, and the playback time of the music piece is displayed in "audio now". The image matching the first impression section is displayed at the bottom before playing the music piece, and after playing the music piece, the image is changed (the URL of the image is changed) each time the playback time of the music piece exceeds the time of the impression change.

The image transition of the slideshow is shown in Figure 3. In this case, from Table 1, the first impression section is the section numbers 1 to 7, which is from 0 seconds to 17.5 seconds of the music piece. The second impression section is the section numbers 8 to 18, which is from 17.5 seconds to 45.0 seconds of music piece. The last impression section is the section numbers 19 to 24, which is from 45.0 seconds to 60.0 seconds of the music piece. Image data similar to impression of the section numbers 1, 8 and 19 which is the start section of each impression



Figure 2: Slideshow screen



Figure 3: Image transition of a slideshow

sections are obtained as a search result, and the image is switched at the timing of impression change point of 17.5 seconds, 45.0 seconds.

It was a somewhat calm music piece, and the impression was not far from all the pictures. The switch from the second to the third was also easy to understand on the music piece side. It was a bright impression from the dark impression, and we felt similar about the result image. However, we felt that switching from the first to the second was felt on the music piece side, but it was not reflected on the image side.

The image data stored in the database to be searched this time is a part of the image data used in the previous study [15]. The number of image data is 16,387. Improvement in accuracy can be seen by increasing the number of image data.

5.4 Similarity Measurement between Images

Tables 2 to 4 show the results of image recognition for the search results. Those for which multiple results are obtained are synonymous with each other. Focusing on the result with the highest estimation rate, the first image is estimated to be snow leopard, and there is no problem in recognition. The second image is presumed to be birdhouse. Although there is no such thing in the image, it cannot be said that it is a perfect estimation, but the relation with “tree” is seen. The third image is presumed to be nail, but it is considered to be a false recognition. And in each recognition result, Table 5 shows the similarity between the words estimated to have the highest possibility. This time,

“snow leopard” is a word consisting of multiple words, so “leopard” is used. In addition, for the second and third images, Table 6 shows the similarity when “tree” and “leaf” are used as words judged from the subjectivity separately from the recognition results.

Table 2: Recognition result of the first image

Acquisition word	Estimated rate
snow leopard, ounce, Panthera uncia	0.82735
leopard, Panthera pardus	0.08432
jaguar, panther, Panthera onca, Felis onca	0.00664
tiger cat	0.00132
cheetah, chetah, Acinonyx jubatus	0.00114

Table 3: Recognition result of the second image

Acquisition word	Estimated rate
birdhouse	0.20035
buckeye, horse chestnut, conker	0.06871
park bench	0.05759
mushroom	0.05494
bolete	0.05452

Table 4: Recognition result of the third image

Acquisition word	Estimated rate
nail	0.85775
screw	0.01485
hock, claw	0.01441
plunger, plumber’s helper	0.00697
knot	0.00680

Table 5: Similarity between words of recognition result

Word1	Word2	Similarity
leopard	birdhouse	0.19916947
leopard	nail	0.16623408
birdhouse	nail	0.12244315

Table 6: Similarity between words when determining words by subjectivity

Word1	Word2	Similarity
leopard	tree	0.21785429
leopard	leaf	0.17773704
tree	leaf	0.48228538

6 Consideration

Looking at the picture obtained as the retrieval result, the third image includes white leaves, and the proportion of white occupying the picture is high and the impression which is brighter than the second picture is felt. In fact, the brightness of the section number 8 (the second impression section) is lower than the brightness of the section number 19 (the third impression section). Therefore, it seems that switching from the second picture to the third one was consistent with the switching

from the dark impression to the bright impression in the music piece. Also, the brightness of the section number 8 (the second impression section) is lower than the brightness of the section number 1 (the first impression section). Although the second image may feel the brightness more than the first image, the impression of the image is estimated from the color information and its occupying ratio. Therefore, it is considered that the second image occupied by the trunk portion of the tree has a lower level of clarity than the first image.

In this time, concerning the switching part of the music piece, a great discomfort was not felt on the music piece side. However, since it cannot be determined as a change point only in the 2.5 second break of a music piece, it is considered that a slight discomfort is generated.

About the semantic similarity of the image, the word was acquired by image recognition, but misrecognition existed obviously. Since the accuracy of this image recognition greatly influences the determination of semantic similarity, it is considered necessary to improve the image recognition accuracy. Looking at the results of word similarity, Table 5 shows that the images are not semantically similar. Focusing only on the images, the relationship between “the plant” is felt between the second image and the third image, but such irrelevant words are selected by misrecognition. From Table 6, when the word is determined by subjectivity, the similarity between the second image (tree) and the third image (leaf) has a higher value than the others, and the relationship is shown.

7 Conclusion

Various researches for retrieving multimedia data based on impressions are being conducted. Consideration of the temporal change of the impression of a music piece was one of the problems in the mutual retrieval system of the previous study [15]. In order to be able to take that into consideration, research is being conducted to detect temporal changes in the impression of a music piece [13]. However, in the detection method of the change point of the impression of the music pieces, there were improvement points such as the same change repeated and the impression was detected as a change point at a place where the impression was felt as a group. In addition, when displaying an image that matches the impression of a music piece that changes over time, a method of using a slideshow that presents an image while reproducing the music pieces is conceivable, and it is necessary to consider whether the images used in the slideshow are in harmony with each other. This paper proposed a method to cope with the cases when an impression repeatedly changed, a method to generate an HTML file, in which the slideshow is displayed by acquiring images matching each impression of the music pieces when specifying the music piece, and a method of measuring semantic similarity between slideshow images.

We conducted a trial experiment of the proposed method and showed that the deletion of the part where the impression is perceived as a unit and the repeated changes is successful. The proposed method can improve the accuracy of change point detection of impression of music piece. The change point

detection was performed at the time of the operation of the search system, and the search in consideration of the change in the impression of the music was made possible. By using the slideshow, the search result is presented at the same time as the change part more easily. Although we did not detect anything other than the classical music piece this time, we are planning that the proposed method will perform the change point detection which can be executed regardless of the genre of music piece.

Currently, in this system, only one set of slide-show that is judged to be most suitable for each impression can be created by presenting multiple candidates for each impression and changing the combination. The image data to be searched this time is a part of the data automatically collected from the Web and stored in the database in the previous study [15]. Since the image data are managed by the URL, image data that would not be desirable as a search result are also included. In addition, the prepared image data are not sufficient, image data such as icons whose impressions do not coincide with music pieces may be obtained as retrieval results in some cases. This is because the impression of the image data is estimated only from the color information and it is determined that they are similar. In addition to deleting unfavorable image data, it is conceivable to modify it so as not to collect undesirable data when collecting data from the Web in the system of the previous study [15]. As for the estimation of the impression of the image data, combinations other than the color information may be combined to make it difficult to obtain undesirable results as the search result.

It is required to designate the number of searches and to present candidates with similarities up to that number for each impression section. As a result, by automatically performing image recognition and inter-word similarity measurement, it is possible to automatically generate a slideshow with a combination of high semantic harmony. Also, a slideshow can be created by using an image selected by the user from among a plurality of candidates. This is to prevent a sense of incompatibility from occurring when presented as a slide show, even though it most closely matches each impression section. In addition, there are individual differences among users in the impression felt from the image, so multiple presentations are necessary.

In addition, it is necessary to perform a subject experiment such as detection of the change part of the impression and whether the presented slideshow is not only partially but also entirely matched with the impression.

Acknowledgment

This work was partially supported by JSPS KAKENHI Grant Number 16K00370.

References

- [1] Google Code | Archive / Projects / word2vec, from <https://code.google.com/archive/p/word2vec/>, Retrieved January 22, 2019.

- [2] K. Hachimura, and S. Eiho, "Retrieval of Paintings based on Color Distribution and Impression Words," *IPSJ SIGCH*, 95-CH-27, (in Japanese), 95(91):37-44, 1995.
- [3] S. Harada, Y. Itoh, and H. Nakatani, "Interactive Image Retrieval by Natural Language," *Optical Engineering*, Vol. 36, No. 12, pp 3281-3287, 1997.
- [4] T. Hochin and T. Tsuji, "Mutual Multimedia Access Using Kansei Factors," *Kansei Engineering International*, 2(4):9-18, 2001.
- [5] T. Hochin, K. Yamada, and T. Tsuji, "Multimedia Data Access Based on the Sensitivity Factors," *Proc. of the 2000 International Database Engineering & Applications Symposium*, pp 319-326, 2000.
- [6] W. Hsu, T. S. Chua, and H. K. Pung, "An Integrated Color-Spatial Approach to Content-based Image Retrieval," *Proc. of 3rd ACM Int'l Multimedia Conf. and Exhibition (MULTIMEDIA'95)*, pp 305-313, 1995.
- [7] T. Kitagawa, T. Nakanishi, and Y. Kiyoki, "An Implementation Method of Automatic Metadata Extraction Method for Music Data and Its Application to Semantic Associative Search," *System and Computers in Japan*, 35(6):59-78, 2004.
- [8] Y. Kiyoki, T. Kitagawa, and T. Hayama, "A Meta-Database System for Semantic Image Search by a Mathematical Model of Meaning," *SIGMOD RECORD*, 23(4):34-41, 1994.
- [9] T. Kumamoto, *Design and Implementation of Natural Language Interface for Impression-based Music-retrieval Systems*, Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin Heidelberg New York 3214:139-147, 2004.
- [10] T. Kumamoto, and K. Ohta, "Design and Development of Natural Language Interface for an Impression-based Music Retrieval System," *Technical Report of IPS of Japan*, 2003-NL-153(13):97-104, 2003.
- [11] H. Lu, B.-C. Ooi, and K.-L. Tan, "Efficient Image Retrieval by Color Contents," *Proc. of 1st Int'l Conf. on Applications of Databases*, pp 95-108, 1994.
- [12] T. Nakanishi, T. Kitagawa, Y. Kiyoki, "An Implementation Method of Associative Search for Heterogeneous Media Data Utilizing the Mathematical Model of Meaning and Its Application to Image Data and Facial Expression," *Proc. of 2003 IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, pp 613-618, 2003.
- [13] Y. Shinjo, T. Hochin, and H. Nomiya, "Detecting Changes of Music Impressions for Changing Pictures," *Proc. of 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD2017)*, pp 537-542, 2017.
- [14] Y. Shinjo, T. Hochin, and H. Nomiya, "Improvement of Similar Image Retrieval Considering Temporal Changes of Music Impression," *Proc. of 31st International Conference on Computer Applications in Industry and Engineering (CAINE2018)*, pp 189-194, 2018.
- [15] Y. Takahashi, T. Hochin, and H. Nomiya, "Improvement of Mutual Retrieval of Visual and Audio Materials Based on Impression," *Proc. of 4th International Conference on Applied Computing & Information Technology (ACIT 2016)*, pp 117-122, 2016.
- [16] Y. Takahashi, T. Hochin, and H. Nomiya, "Mutual Retrieval of Pictures and Sounds on the Web Based on Impression," *Proc. of 2015 International Conference on Computer Application Technologies (CCATS 2015)*, pp 80-85, 2015.
- [17] TensorFlow – Develop – TUTORIALS - Image Recognition, Retrieved January 22, 2019, from https://www.tensorflow.org/tutorials/images/image_recognition.



Yuto Shinjo works at OMRON SOFTWARE. He received his B.S. and M.S. degrees from Kyoto Institute of Technology in 2017 and 2019, respectively. His research interests include multimedia engineering.



Teruhisa Hochin is a Professor in the Faculty of Information and Human Sciences at Kyoto Institute of Technology. He received his B.S., M.S., and Ph.D. degrees from Nagoya Institute of Technology in 1982, 1984, and 1994, respectively. His research interests include affective engineering, Kansei information retrieval, multimedia databases, graph-based databases, scientific and flexible databases, and extensible database management system.



Hiroki Nomiya is an Associate Professor in the Faculty of Information and Human Sciences at Kyoto Institute of Technology. He received his B.S., M.S., and Ph.D. degrees from Kobe University in 2002, 2004, and 2008, respectively. His research interests include multimedia data engineering, facial expression recognition, and machine learning.

QoS of Cloud Computing - Application of the JPManager in a Cloud Service

Jiang Guo* and Yuehong Liao*
California State University, Los Angeles, CA USA

Abstract

As Java-based Software as a service is more and more widely used in developing enterprise-level private cloud-based applications, the Java based platform plays a very important role in large scale enterprise level system development. However, the performance management of Java-based software services is a challenge to software developers. It is very difficult for the software developers to evaluate, analyze, and improve the performance of Java-based software services without software tools, due to the fact that hundreds of software services, software components, and application programming interfaces need to be tracked in real time and their execution can typically be completed within seconds or milliseconds. This paper proposes an approach to collect the performance metrics information of Java-based cloud computing services by using Java instrumentation to find improvements such as reducing bottlenecks. A software tool – JPManager – a Java software service performance management system, which we developed based on our approach to monitor, analyze and manage the performance of Java-based software services, is also discussed.

Key Words: QoS, Java instrumentation; performance management; software services.

1 Introduction

One of the challenges of cloud applications is Quality-of-Service (QoS) management [4, 7, 19], which includes performance, availability and reliability. The improvement of QoS can reduce the risk management costs and deployment costs in cloud computing. The QoS data collected by software tools can improve the quality of integration of existing software services as well as provide a basis to compare the cloud services from different vendors. It helps customers purchase cost-effective cloud computing services [17, 20, 24, 30]. The QoS data also helps developers analyze the QoS margin and contingency during the cloud service development and deployment.

Java-based software services are widely used as cloud computing services. Google's App Engine, Pivotal Software's Cloud Foundry, Red Hat's OpenShift, and Amazon's Elastic Beanstalk all support Java based applications and cloud

computing services. As more and more Java applications, J2EE applications, and Mobile Apps are being moved to the cloud, the QoS of Java-based software services needs to be improved [3, 13].

To implement performance management of Java-based software services in real-time production private environments, software developers and administrators need automatic tools to monitor and manage the application systems [2, 16]. They must also monitor the performance of front-end and back-end technologies in conjunction with Java-based components.

This paper discusses the design, implementation, and application of our research project JPManager. The goal of our project is to evaluate and help to improve performance of the Java-based private cloud computing systems as well as uncover the bottlenecks of the software services and executing environments.

To evaluate the performance of each software service component or each method, we used the Java instrumentation approach in our research. The strategy of this approach is to instrument or insert additional Java codes into the application itself [5, 9, 15, 26]. These extra Java codes are called Java agents, which can be placed at check points set by the developers or administrators to collect fine grained measurements of software service performance [27, 29]. This approach enables developers to collect the metrics information at a very deep level of the application structure, including before and after the method calls.

The Java instrumentation approach that we used in our project can detect the bottlenecks in the application components, component methods, and even method lines. The method also takes measurements from dynamic, real time execution and developers can collect the performance metrics information of the execution time of the applications as well as their executing environments. This paper is an extension of our previous work in [12].

2 Related Work

QoS is very important for cloud computing [13]. Much research has been done to improve the QoS for cloud software services providers [1, 14, 23], such as broker-based QoS frameworks [32], QoS aware networks [6, 28], extensions of the UDDI data structure or SOAP to include QoS specific information, and QoS-enabled cloud computing services [8].

Many researchers have developed and implemented tools to

* Department of Computer Science. Email: {jguo@calstatela.edu}.

manage the performance of Java-based applications [18, 21, 31]. VERITAS Indepth focuses on the bottlenecks between the middle tier and the database. Quest Central emphasizes performance tuning and optimization of Java application codes. JDBInsight aims at simplifying the performance tuning and testing of applications, which access data through the Java Database Connectivity (JDBC) API. The Dirig Application Performance Platform (APP) looks at the business perspective of multi-tiered applications. DiagnoSys provides users with detail information about software component performance at the enterprise level [11].

Marshall provides a list of available tools [22] for QoS. In his list, most tools focus on CPU, memory, and basic JVM monitoring, such as the NetBeans Profiler, the VisualVM, the Eclipse Memory Analyzer, and the JProfiler. The NetBeans Profiler is part of NetBeans IDE. However, the JProfiler does not monitor JVM. The VisualVM is based on the NetBeans platform. It also supports garbage collection monitoring. The GC Viewer provides visualization of data generated by built-in Java functions, such as garbage collection. The Eclipse Memory Analyzer analyzes Java heaps while the Java Interactive Profiler supports basic profiling. The Profiler4j focuses CPU and remote profiling and the Java Performance Analysis Tool (Patty) analyzes method execution and code coverage, based on memory and CPU usage. Finally, the JRockit supports JVM profiling and monitoring; it also has diagnostic features.

Our solution focuses on performance management, especially overall performance data collection and analysis. It is quite different from those tools mentioned above. Also, our solution supports data normalization and analysis. Our data visualization is based on the UML sequence diagram. It is easier for developers to notice the red flag issues. Our approach is based on JVM modification and provides users more flexibility to control the JVM code instrumentation. In our solution, the users also can turn on/off the instrumentation. This approach will not only help developers to find operational problems in order to improve performance, but also helps administrators to deploy the application and configure the system and balance the computing load to reach the best performance. Some researchers have focused on building Java-based software service performance tools to evaluate the performance of large-scale systems to support optimization. But, many of these tools have some overhead; some tools rely on existing software; and some tools focus on specific parts of the applications.

3 The Software Structure of the JPManager

The JPManager is a software tool we designed and implemented specifically to support the performance management of Java-based software services. It provides all the critical support to collect and analyze the performance information for Java-based private cloud applications.

Performance management can be used at two phases: the development phase and the production phase. At the development phase, the performance management tools can

help developers to uncover the root cause of the performance problems and improve the software development to meet the customers' performance requirements. At the production phase, the performance management tools can help administrators to configure the software deployment and hardware configuration and balance the computing power and computing load among the different cloud software services in the production environments. The JPManager supports developers and administrators at both phases. It is a configurable and unloadable software system. That means that administrators can unload the system after they tune the performance as desired and then deploy the cloud-based software service applications. So, the approach will produce less overhead to the applications in the production environment.

The JPManager uses a multi-tier and distributed architecture to support the performance management needs of Java-based private cloud applications. It allows developers and administrators to collect and analyze runtime information of the applications. It not only helps developers to find out the bottlenecks automatically, but also allows developer to specify the performance information they want to collect and analyze if they have special needs. At the performance collections level, the developers can easily use a profile file to specify:

- (1) The types of methods and classes needed to be monitored during execution;
- (2) The methods and classes needed to be monitored; and
- (3) The metrics information needed to be collected for each method and class.

All components in the JPManager are integrated to complete the task of performance information collection and analysis. At the lowest layer are the Data Collection Agents. These agents are responsible for inserting Java probes into the application code and collect the performance data as the software services are executed. Then the Data Collection Agents forward the performance data to the middle layer, the JPManager Server, which maintains a database as the centralized log data system. The JPManager stores and manages all the performance data to make them ready for analysis agents to use. The highest level is the Analysis Agent, which is responsible for analyzing the performance data and displaying the dynamic analysis and measurement information to the developers and administrators according to their needs and configuration.

The JPManager enable users to control the Java instrumentation easily. The application components and component structure are automatically discovered according to the users' instrumentation profile. Thus, the Data Collection Agents can monitor the software services and application execution environment continuously and make adjustments according to the users' requirements. Also, the "byte-code" instrumentation technology used by the JPManager can perform the instrumentation of the third-party Java components, which are the classes retrieved from remote sites, to evaluate the individual component's performance

consistently and overall system performance seamlessly. The byte code instrumentation logs specific events of the application and execution environment during its execution, such as method calls and object instantiations.

An instrumentation task for a given cloud service includes two main steps. The first step is to specify the components of an enterprise level application to be evaluated, such as methods in a package. The second step is to specify the types of methods or classes to be instrumented, such as a method's entry and exit, and the information will be collected for each method or class, such as time stamps. A composite instrumentation task contains several small instrumentation activities. In this way, the users can collect run time information from the different components of the tested application.

The JPManager provides performance data to help developers and administrators to monitor and analyze the bottlenecks of the systems accurately and quickly. Performance data of the components and systems are visualized for users to identify the performance bottleneck easily. The JPManager uses an UML sequence diagram-based display technology to help users to track the interactions between the objects and the components. This type of time stamp-based performance information greatly helps developers and administrators to uncover performance problems and unearth the exact location of the bottleneck in the applications.

To evaluate the performance and unearth the bottlenecks of Java-based services, the major function of the Data Collection Agent (DCA) is to extract the performance information, such as response time, execution and latency of the components, objects, and methods, from the services implemented in Java. A DCA runs on the software services executing platforms (See Figure 1). The DCA works very well on most available Java-based platforms, application servers, JDBC APIs and database servers, such as Apache Web Server, Tomcat Application Server, JBoss EJB Containers, BEA's WebLogic, IBM's WebSphere, and IBM's Cloudscape.

4 Our Dynamic Java Instrumentation Approach

A DCA runs in the JVM memory space (Figure 2). The Java instrumentation can be implemented as either static or dynamic.

In the static approach, we can read through the program to be instrumented and generate a properly instrumented program to implement an instrumentation task, based on the instrumentation profile. In this approach, the source code of the Java-based services will be changed before they are compiled. It is a completely intrusive approach. The probe codes will be inserted into the source before and after the method called or before and after the object creations to log the response time, execution and latency. Then, the instrumented program can be compiled and executed, and the execution information will be extracted and logged. Based on the logged performance data, performance analysis agents can evaluate the application performance and find the bottlenecks.

However, this static approach has some problems that limit its applicability. The first issue is the overhead; the execution time and response time might not be accurate because of the side-effect of the inserted code. The second issue is the source code; if the source codes are not available, then this approach cannot be used. This means we cannot obtain the performance data of the third-party classes retrieved from the remote site. We also cannot obtain the performance data in the operating environment. The third issue is real-time evaluation; developers and administrators cannot change the instrumentation profile and the performance monitor tasks at run time because the application codes need to be inserted and compiled before the performance data can be logged. Because of these issues, we cannot obtain the overall and detailed performance information for the Java-based services with this approach. Because of these issues we discussed above, we did not use this approach in our JPManager design and implementation.

Instead, we use a dynamic approach in our system design for

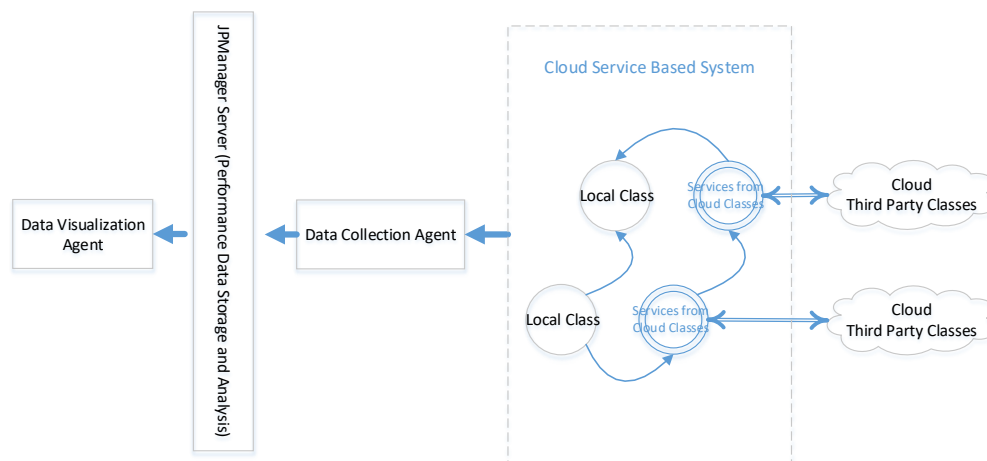


Figure 1: The structure of the JPManager

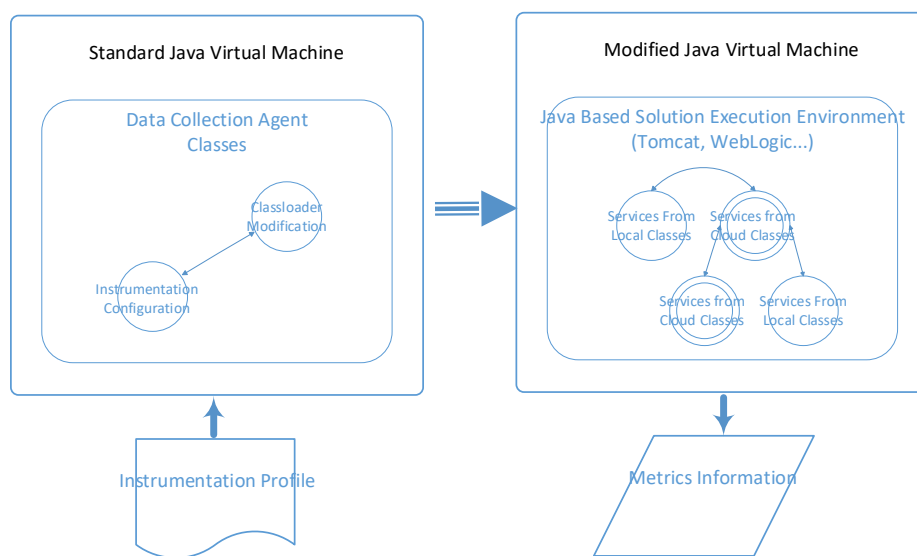


Figure 2: Two JVM structures of the DCA

the JPManager; the DCA instruments the probe code into the byte codes (class files) of the Java-based services dynamically when the byte codes are loaded into the JVM and are executed (interpreted). This means that when the Java byte codes are loaded during execution, a specialized ClassLoader will be used. This dynamic approach is very flexible and convenient if users change the instrumentation frequently. This approach completely solves the problems found in the static approach. Thus, we used this approach in our JPManager.

In our dynamic instrumentation approach, when the Java Virtual Machine (JVM) loads and executes a Java class of Java-based services, the DCA uses a profile file to construct an Instrumenter to capture the data of the software service performance metrics, such as the timestamp before and after the execution of a method. When the JVM loads the classes into the JVM memory space, the probe codes are inserted into the class byte codes. This task is performed based on the Metric Provider Class. This class has the information of the metrics and knows what data the DCA needs to collect. This class also has a method to make adjustments in the metrics, based on the instrumentation profile, which is an XML file that can be changed by the users.

The instrumentation profile is an XML file. It is used for configuring the instrumentation location, such as packages and classes. It allows users to configure the instrumentation at runtime, since the information is only used when the class loader is changed. Our approach can probe the method calls specified in the instrumentation profile, based on the users' requirement, such as entry and exit of a method, before and after a method call.

To build a new cloud-based enterprise wide system, developers must decompose the large system into subsystems. This decomposition process includes functional decomposition and non-functional decomposition. The timing constraint is a part of the non-functional decomposition. As the system architecture is defined, system performance has to be verified.

At the detail design phase, the lower level decomposition is finished and system performance is allocated to its components. The timing contingency at the component level is established. The developers can specify the instrumentation points and verify that the performance of its components meet the constraints.

For existing systems, developers can extract dependency models from the system. Based on dependency analysis, a critical path can be specified. The instrumentation points on the critical path can be checked. Based on the data collected at the instrumentation points, developers can improve algorithms of the methods and therefore system performance.

The selection of instrumentation points is a challenge. This depends on experience and intuition. However, dependency models, statistics, and analysis are always helpful.

When a class is loaded into the JVM space, the DCA parses its byte codes to determine if the class needs to be monitored. If it is, the DCA will create an Instrumenter to collect the metrics of the class.

In our dynamic approach, the third-party classes, such as the classes retrieved from a remote site, can also be monitored easily since the probe codes are inserted into byte code instead of source code. In this approach, the monitoring of classes and methods can be configured dynamically at run time since the application classes do not need to be re-compiled. The byte code instrumentation approach also greatly increases the performance management power, not only the classes of applications, but also the classes of the execution environment can be monitored.

To manage the performance of Java services, we need to collect different types of dynamic information of the application program executions, such as execution time of methods, response time of the APIs, and latency of retrieving components from remote sites.

In order to discuss our dynamic Java instrumentation, we need to discuss how the class byte codes are loaded into the

JVM space. The Java Virtual Machine (JVM) has three major components: Class loaders, Class file verifier, and Execution engine.

The class loader is one of the three important components in the JVM. It is also the most important part in our Java instrumentation approach. From the class loader delegation architecture, we can see that all of the classes are loaded by the primary class loader.

This means that we can implement our Java instrumentation by dynamically modifying the running class loader – the Primary Class Loader. Thus, we designed an approach to change the primary class loader. We consider this approach as class loader modifying. That means that we implement additional features in the primary class loader after it is loaded in the memory by the JVM. Therefore, we can implement our Java instrumentation tasks by changing the Primary Class Loader; we describe this as Primary Class Loader modifying.

When considering implementing the Java instrumentation, we need to focus on the Java instrumentation to be performed to a specific code construct. These code constructs might be the method calls that are specified by the users in the instrumentation profile file.

The DCA collects run time information of each class or method to be monitored. For example, the metrics information that a method call can be collected includes the information of the target object and all of the parameters passed to the invoked method. Also, the time stamps of the method entry and exit can be logged.

5 Data Collection and Implementation

When the classes are loaded into the JVM space, the DCA instruments the classes at the byte code level. All the

instrumentation locations are based on an instrumentation profile file. Figure 3 is a dataflow diagram of a DCA execution. The DCA uses a dynamically changed class loader to load the classes to be instrumented and implements the instrumentation based on a configuration profile file. The classes that are not to be monitored will be loaded normally. As the JVM executes the probe codes that are instrumented in the Java classes, dynamic information, such as time stamps, is logged and saved into the database. Then, the JPManager server processes the collected dynamic data.

The dynamic instrumentation of the Java byte code is implemented in four steps by modifying the Primary Class Loader (see Figure 4).

The dynamic instrumentation of the Java byte code is implemented in four steps by modifying the Primary Class Loader (see Figure 4).

(1) Execute a DCA in a standard JVM. During this step, a standard JVM is launched. A DCA is loaded and executed in the JVM memory space.

(2) The DCA uses an XML-based instrumentation profile to modify the standard ClassLoader. This step is patching the ClassLoader and instrument code according to the user's configuration. At this step, the system takes two actions.

First, the DCA needs to locate the correct class loader – the primary class loader – to modify. This class loader is an object that is responsible for loading all the classes in the delegation architecture of class loaders, as discussed above. In the delegation architecture, this class loader class searches for classes and resources in the local paths specified in the system environment variables or in the network class path specified in the URLs. It is the virtual machine's built-in class loader and

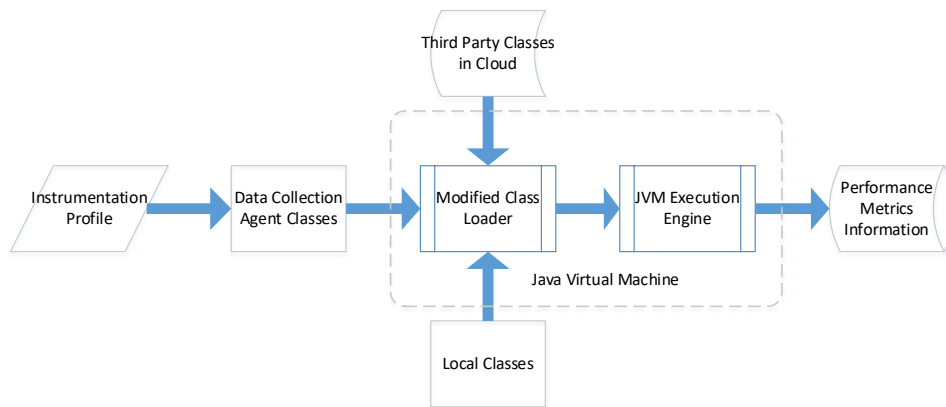


Figure 3: Java instrumentation data flow



Figure 4: Implementation of instrumentation by the modifying primary class loader

is the system class loader for all the classes to be instrumented.

The system uses the `ClassLoader.getSystemClassLoader()` to find the right `ClassLoader` to patch. A delegation model is used to find the classes and resources. In this delegation model, each `ClassLoader` has a parent class loader. The `ClassLoader` searches through the delegation model to find the class or resource requested. The virtual machine has a standard class loader as the parent of a user-defined `ClassLoader`. The Java language provides a method `getSystemClassLoader()` to acquire the standard class loader.

Second, the DCA modifies the primary class loader according to the user-defined instrumentation profile. After the modification of the JVM's built-in class loader, it will work in the desired way. Here, the most important information is the instruction list of the class loader. Then, we can employ the `defineClass` method. When the JVM loads the classes, all the classes will go through this `defineClass` method. In this method, the DCAs insert a call to invoke our class tracer. Our tracer will insert the probe byte code into the class byte codes when they are loaded into the JVM space.

The DCA uses an instrumentation profile to customize the `ClassLoader`. The method `ClassLoaderPatcher()` is used to change the standard `ClassLoader` into a customized loader.

(3) Construct another JVM with the modified class loader and execute Java-based services in the modified JVM. During this phase, we use the modified primary class loader and redefine the class approach to create a new JVM. Then, the application classes will be loaded and executed in the newly generated JVM.

The system creates another JVM with the modified `ClassLoader` and executes the software service in the changed JVM. The method `vm.redefineClasses()` can be used to launch another JVM. In fact, the JVM memory space is still the same. It only uses the modified `ClassLoader` to load the software service classes.

The interface `VirtualMachine` can be used to control the execution of a VM virtual machine. It can be used to retrieve the global VM attributes and manage the VM execution. Normally, a system built-in method, `Connector`, is used to create a VM object. The system built-in method `redefineClasses` provides the ability to replace the standard `ClassLoader` with the patched `ClassLoader` during the execution of the software services and applications.

Here, we use a Java Platform Debugger Architecture mechanism to control the execution of a virtual machine. We use this mechanism to access the attributes of the JVM and control the execution of the VM. The redefine class approach allows us to replace a class file dynamically at running time. This means a class can be updated while it is running. In our Java instrumentation approach, our DCA uses this approach to replace the primary class loader definitions with our modified class loader that we used in Step 2. In this step, the standard

class loader is replaced by our patched class loader during the execution of the applications. One of the concerns of substituting a running class is the execution continuity issue. The good thing is that our dynamic approach does not cause any initialization. In other words, while redefining a class, the JVM will continuously execute the class instead of executing it from the beginning. So, the execution of the class loader will not be interrupted. Then, our modified class loader loads the software service and application classes. In this way, the behaviors of all the classes can be changed and allows the DCA to instrument the logging code into the byte code of the Java-based software services and applications.

After the primary class loader is substituted by a user patched class loader, the new Virtual Machine needs to be launched. The Virtual Machine Manager can be used for the VM just created. Then, the launch method is used to load the classes of software service and application and execute them.

(4) While the software service classes are loaded into the JVM space, our modified class loader will insert the probe codes into them. In this way, when the execution engine executes the classes, the run time information will be logged. Then, the run time metrics information is stored in the database. Afterwards, the JPManger Analysis Agent accesses this information, performs a normalization procedure, and then locates the bottlenecks of the systems.

6 Dynamic Execution Consistency Analysis

In this section, we summarize our analysis of the Java instrumentation approach from the dynamic execution consistency point of view. Our approach will keep the dynamic execution consistency after the Java instrumentation.

In order to guarantee that our Java Instrumentation keeps the dynamic execution consistency, we need to exploit the execution of the Java program, $\varepsilon(p)$. $\varepsilon(p)$ is an execution process of a Java program. From the execution point of view, there should be no side effects. That means that the instrumentation should have $\varepsilon(\pi(p)) = \varepsilon(p)$. Thus, we need an execution model to explain how to execute programs and generate outputs. In this execution model, a Java object can be represented as a relation between Java programs p and its execution $\varepsilon(p)$.

We can verify the program dynamic consistency by examining the object state space in the Java program, since the object state change is a result of the method calls.

The program dynamic consistency can be classified as:

- State consistency, which requires that every variable value should be same before and after the Java instrumentation.
- Method call consistency, which requires that the temporal relationships between method calls should be the same before and after the Java instrumentation.

We can use pre-condition and post-condition to verify our Java instrumentation will not produce a side effect on an object

state. The pre-condition states must be true before the method is called and post-condition states must be true after the method is executed.

The Appendix provides the details for our dynamic execution analysis. It suffices to say that the software tool that we have developed neither changes the state of any object nor the order of method calls. That is, from the analysis of the relative order between the timestamps of method calls, we know that our Java instrumentation keeps the execution consistency. That means our dynamic approach does not change the behavior of the application systems and there is no side effect to the Java-based software services.

The timestamp based dynamic execution consistency analysis confirms that the software services function consistently before and after the Java instrumentation, no matter what Java class code is loaded from a local machine for a third-party server. That means that our performance management system will not change the software services execution.

7 Case Study and Experiment

In this section, we discuss the experiment of using the JPManager to collect and visualize the performance data of a sample system. In this case study, the JPManager collected the metrics information of the methods calls in a software service-based application. This sample system is a cloud-based store management application. In this application, we use Java service components. For the execution environment, Apache Tomcat is used as the web server and BEA's WebLogic is used as the application server. For data persistence, IBM's Cloudscape is used. It is a pure Java-based database.

In order to test the JPManager, we need to run the DCA and Analysis Agent. The execution of the Data Visualization Agent of the JPManager is shown in Figures 5 and 6. In the view windows, the overview execution of the application is displayed. The execution information is displayed in two panes of the Visualization Agent window. The methods that have already been completed are displayed in the left pane. Some methods may be called multiple times. When a plus (+) icon before a method is clicked, the calling number appears after an icon in the line below. Its value is an integer starting from zero. A zero indicates that that method is called once. When that number is clicked, in the right pane, the method associated with the execution information appears. We use a UML style sequence diagram to display the information of the executed methods.

In the right pane, the sequence diagram shows the logic control, the method call, and the execution sequence of the application systems. It displays the information of interaction and message exchange between software application objects. In this way, it will help the developers and administrators to figure out which methods take too much time. The sequential of execution of the methods is shown along the vertical line, which is called the time line. The method calls are displayed as the horizontal arrows, which displays the sending and receiving messages between the methods. Along the vertical time line; the red and green thick lines are the methods involved during the application execution. On the method execution vertical line, ">" indicates the start point of a method and "<" indicates the end point. The bold labels show the method names. If a method execution line is green, it means the method execution time is within the expectation

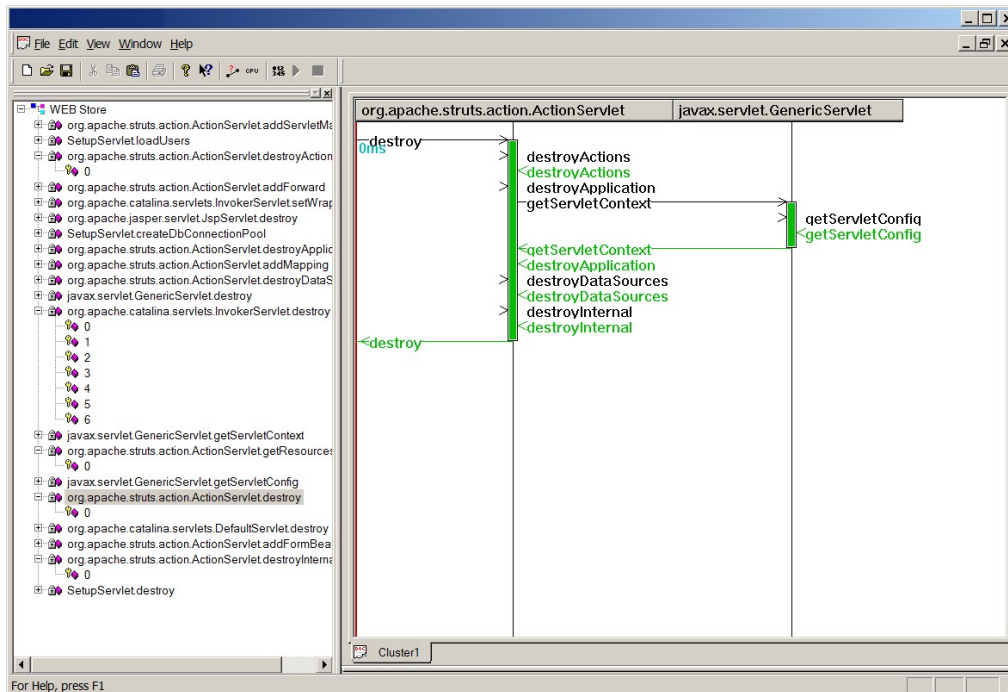


Figure 5: Execution of the Java-based software services without bottleneck

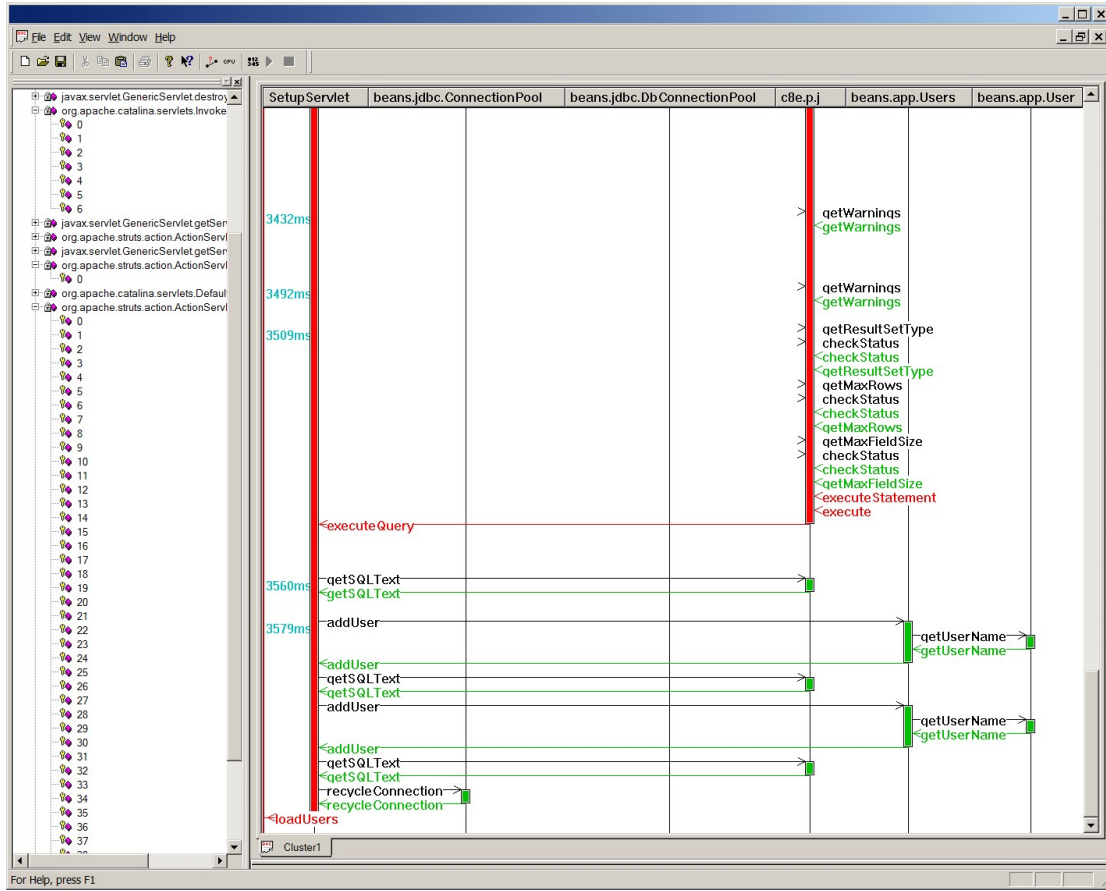


Figure 6: Execution of the Java-based software services with bottlenecks

requirements. The red line means the method call uses more time than expected. It might be a potential bottleneck of the application. The method should then be examined to improve its performance.

To implement automatic analysis of bottlenecks precisely, we not only need to analyze the data collected about the past, but we also need to analyze the data collected from other projects, especially the projects that are in the same or similar application areas. Therefore, our automatic analysis is based on a historical investigation of past projects [10].

In our system, we use $\Delta T = \tau_{after}(m) - \tau_{before}(m)$ to represent timestamp difference. ΔT metrics is the time that is needed for execution of a specific method. Thus, the ΔT metrics collected for projects may vary from project to project. Since the ΔT metrics might be different for different projects, we need to use a statistic approach for the general and valid trend of ΔT .

We use a graphical technique to estimate if a specific point is a bottleneck, based on ΔT metrics data, which is called the control chart [25] [33]. This technique enables us to determine if ΔT has significant variability, meaning that there might be a bottleneck. We also can use this technique to determine if ΔT is in the moving average, compared with other similar projects. Here, we use the moving range control chart to assess the metrics data for ΔT .

To illustrate the control chart approach, let us assume the ΔT metrics data are collected from 12 similar projects, as shown in Figure 7. In the figure, ΔT varies from a low of 300 for Project 12 to a high of 800 for Project 5.

Since ΔT is the difference of the execution of a certain method, it is measured in ms, $\Delta T * 1000$, making it easier to display and understand.

Then, we can use Richard Zultner's approach [25] [33] to develop a moving range (mR) control chart to determine the time expected, as follows:

- (1) Calculate the differences of ΔT between each pair of successive data points; then we obtain the moving ranges for the chart.

$$mR_i = |\Delta T^i - \Delta T^{i+1}| \quad (i = 1, \dots, n)$$

- (2) Calculate the average value of the moving ranges, and then we find the mR bar, which is the center line on the chart.

$$Avg(mR) = \frac{1}{n} \sum mR_i$$

Using the data represented in Figure 7, we have a mR control

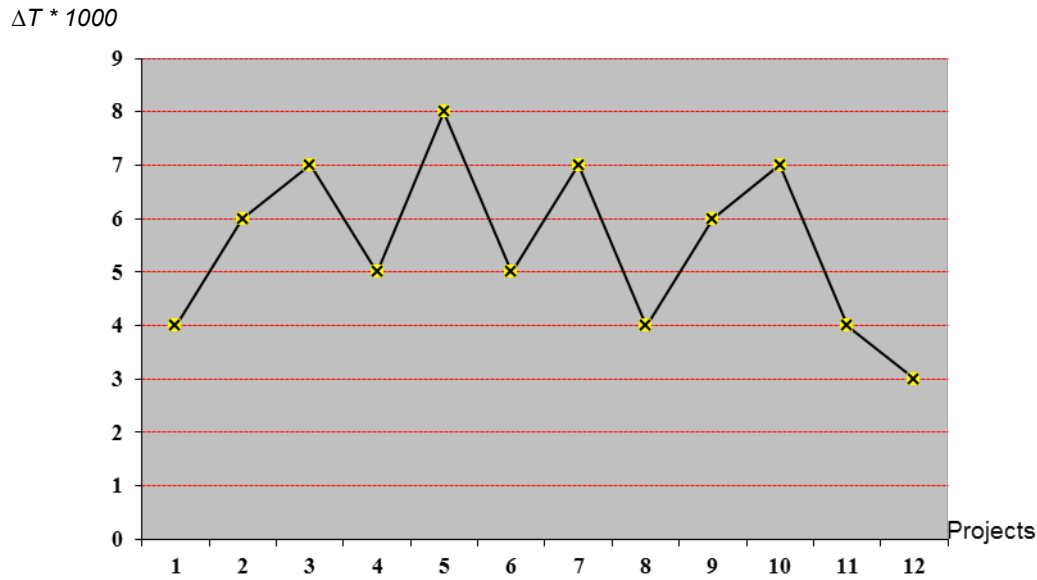
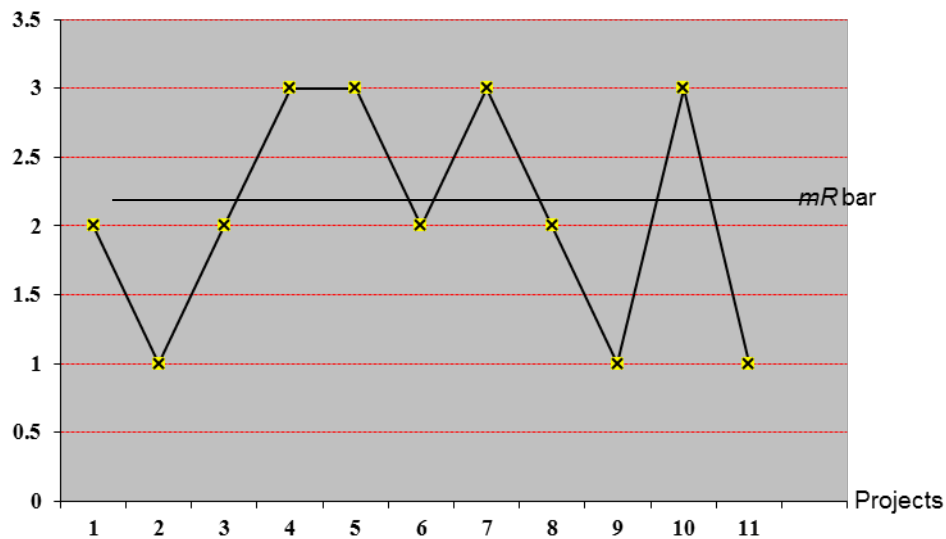
Figure 7: ΔT for 12 similar projects

Figure 8: mR control chart

chart shown in Figure 8. The mR bar value for the moving range data is 2.1. The upper control limit is 6.86.

If the value of ΔT for a project is inside of UCL , then there is no bottleneck and the performance is acceptable.

$$\Delta T < UCL$$

The goal of using Richard Zultner's approach to normalize the data is to analyze the performance data independent of the project order. The mR bar has no relation with the order of data points in Figure 7. Therefore, the order of data points does not affect the analysis results.

For Java programs, we can collect deep level information,

such as method and class level execution information, by implementing code instrumentation.

Finally, to reiterate, the JPManager is a configurable and unloadable software system. This implies that administrators can turn off the Java instrumentation after they tune the performance of the system and deploy the cloud-based software service applications. So, this will not affect the performance of system significantly.

8 Conclusions

The Java-based software service is widely accepted in industry supported by many large companies such as IBM,

Oracle, BEA. This service becomes more and more popular because of its interoperability, portability, scalability, availability, and openness. Further, private cloud computing technologies are used more and more in enterprise level software development.

However, the potential of software as a service has not been fully utilized. As Java-based software service applications are used in more and more different domains and areas, the performance management of these types of applications will become more and more important and the performance metrics that Java-based software service applications must manage will affect the performance of the entire IT infrastructure performance. It will also be a key component that needs to be optimized.

Currently, most commercial Java-based tools focus on service transactions areas, such as database servers, directory services, and application servers. Most of them do not support continuous and deep level monitoring and analysis for the Java-based software service components and APIs. So, they do not have the ability to manage the performance of applications and cannot help developers and administrators to fix the performance problems or optimize the balance of the system loads. Our Java instrumentation-based approach provides a solution to implement the performance management of Java-based software service applications. Our Java instrumentation approach is to insert the probe codes at the byte code level. It works very well not only for users developed Java source code, but also for third-party byte code, even the class byte codes retrieved from remote sites.

In our tool, the performance metrics data is visualized in UML sequence diagrams to help users to detect the performance bottlenecks in the software services and application systems. Other features, such as state monitoring and thread monitoring, are under investigation as future research.

Acknowledgements

The work was supported in part by the NASA under grant NNX15AQ06A

References

[1] A. Abdelmabouda, D. Jawawia, I. Ghania, A. Elsafia, and B. Kitchenhamb, "Quality of Service Approaches in Cloud Computing: A Systematic Mapping Study," *Journal of Systems and Software*, 101:159-179, March 2015.

[2] T. Ahmed, C. Bezemer, T. Chen, A. Hassan, and W. Shang, "Studying the Effectiveness of Application Performance Management (APM) Tools for Detecting Performance Regressions for Web Applications: An Experience Report," *Proceedings of the 13th International Conference on Mining Software Repositories*, Austin, Texas, pp 1-12, May 2016.

[3] Y. Amannejad, D. Krishnamurthy, and B. Far, "Detecting Performance Interference in Cloud-Based Web

Services," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, May, 2015.

[4] D. Ardagna, G. Casale, M. Ciavotta, J. Pérez, and W. Wang, "Quality-of-Service in Cloud Computing: Modeling Techniques and Their Applications," *Journal of Internet Services and Applications*, 5:11, 2014.

[5] W. Binder, P. Moret, É. Tanter, and D. Ansaloni, "Polymorphic bytecode instrumentation", *Software: Practice and Experience*, 46(10):1351-1380, October 2016

[6] D. D'Agostino, A. Galizia, An. Clematis, M. Mangini, I. Porro, and A. Quarati, "A QoS-Aware Broker for Hybrid Clouds," *Computing*, 95(1):89-109, May 2013.

[7] E. Demin, V. Dubinin, and S. Patil, "Automation Services Orchestration with Function Blocks: Web-Service Implementation and Performance Evaluation," *Service Orientation in Holonic and Multi-Agent Manufacturing*, pp 213-221, March 2016.

[8] A. Eleyan and L. Zhao, "Extending WSDL and UUDI with Quality Service Selection Criteria," *Proceedings of the 3rd International Symposium on Web Services*, pp 1-10, April 2010.

[9] J. Frenzel; K. Feldhoff, R. Jaekel, and R. Mueller-Pfefferkorn, "Tracing of Multi-Threaded Java Applications in Score-P Using Bytecode Instrumentation," 31th International Conference on Architecture of Computing Systems, ARCS Workshop, Braunschweig, Germany, April, 2018

[10] J. Guo, "Towards Automatic Analysis of Software Requirement Stability," *Advances in Information Sciences and Service Sciences*, 2(1):33-42, March 2010.

[11] J. Guo, Y. Liao, and B. Parviz, "A Survey of J2EE Application Performance Management Systems," *Proceedings of the 2nd IEEE International Conference on Web Services*, San Diego, California, USA, pp 724-731, July 2004.

[12] J. Guo; Y. Liao; and B. Parviz, "A Performance Validation Tool for J2EE applications," *Proceedings of 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pp 387-396, March 2006.

[13] I. Gupta, J. Kumar, and P. Rai, "Optimization to Quality-of-Service-Driven Web Service Composition using Modified Genetic Algorithm," 2015 International Conference on Computer, Communication and Control (IC4), Indore, India, September, 2015.

[14] P. Hershey, I. Raytheon, S. Rao, C. Silio, and A. Narayan, "System of Systems for Quality-of-Service Observation and Response in Cloud Computing Environments," *IEEE Systems Journal*, 9(1):212-222, January, 2014.

[15] F. Horváth, T. Gergely, Á. Beszédes, D. Tengeri, G. Balogh, T. and Gyimóthy, "Code Coverage Differences of Java Bytecode and Source Code Instrumentation Tools," *Software Quality Journal*, 27(1):79-123, March, 2019.

- [16] H. Jayathilaka, C. Krintz, and R. Wolski, "Response Time Service Level Agreements for Cloud-Hosted Web Applications," *Proceedings of the Sixth ACM Symposium on Cloud Computing*, Hawaii, pp 315-328, August, 2015.
- [17] G. Kang, M. Tang, J. Liu, X. Liu, and B. Cao, "Diversifying Web Service Recommendation Results via Exploring Service Usage History," *IEEE Transactions on Services Computing*, 9(4):566-579, July, 2016.
- [18] S. Kumari and S. Rath, "Performance Comparison of SOAP and REST Based Web Services for Enterprise Application Integration," 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Kochi, India, August, 2015.
- [19] L. Lee, C. Peng, and C. Fan, "An Empirical Study of Service Quality: Web Service Business Transformation," *International Journal of Computational Science and Engineering*, 12(1):58-64, February, 2016.
- [20] S. Li, J. Wen, F. Luo, M. Gao, J. Zeng, and Z. Dong, "A New QoS-Aware Web Service Recommendation System Based on Contextual Feature Recognition at Server-Side," *IEEE Transactions on Network and Service Management*, 14(2):322-342, June, 2017.
- [21] Y. Ma, X. Liu, Y. Liu, Y. Liu, and G. Huang, "A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web Apps on Android," *IEEE Transactions on Mobile Computing*, 17(5):990-1003, September, 2017.
- [22] A. Marshall, <https://blog.idrsolutions.com/2014/06/java-performance-tuning-tools/>, 2014
- [23] V. Nallur and R. Bahsoon, "A Decentralized Self-Adaptation Mechanism for Service-Based Applications in the Cloud," *IEEE Transactions on Software Engineering*, 39(5):591-612, 2013.
- [24] NASA'S Progress in Adopting Cloud-Computing Technologies, Report No. IG-13-021 (Assignment No. A-12-022-00), July 29, 2013.
- [25] R. Pressman, *Software Engineering: A Practitioner's Approach*, Fifth Edition, McGraw-Hill, 2001.
- [26] V. Saini and A. Singh, "Java Byte Code Instrumentation," *International Journal of Advance Research, Ideas and Innovations in Technology*, 4(2):2170-2190, 2018.
- [27] M. Sarrab, "Bytecode Instrumentation Mechanism for Monitoring Mobile Application Information Flow," *International Journal of Security and Networks*, 10(3):191-206, 2015.
- [28] M. Tajiki, B. Akbari, and N. Mokari, "Optimal, QoS-Aware Network Reconfiguration in Software Defined Cloud Data Centers," *The International Journal of Computer and Telecommunications Networking*, 120(C):71-86, June, 2017.
- [29] S. Umatani, T. Ugawa, and M. Yasugi, "Design and Implementation of a Java Bytecode Manipulation Library for Clojure," *Journal of Information Processing*, 23(5):716-729, 2015.
- [30] X. Wang, J. Zhu, Z. Zheng, W. Song, Y. Shen, and M. Lyu, "A Spatial-Temporal QoS Prediction Approach for Time-aware Web Service Recommendation," *ACM Transactions on the Web (TWEB)*, 10(1):1-25, February, 2016.
- [31] F. Yin, D. Dong, S. Li, J. Guo, and K. Chow, "Java Performance Troubleshooting and Optimization at Alibaba," 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP) Date of Conference, Gothenburg, Sweden, May, 2018.
- [32] T. Yu and K. Lin, "A Broker-Based Framework for QoS-Aware Web Service Composition," *Proceeding of IEEE International Conference on e-Technology, e-Commerce and e-Service*, Hong Kong, China, pp 22-29, March 2005.
- [33] R. Zultner, "What Do Our Metrics Mean?" *Cutter IT Journal*, 12(4):11-19, 1999.



Jiang Guo is a Professor of Computer Science currently at California State University Los Angeles. Prior to joining California State University in 2001, he worked as a Research Associate of the United States National Research Council from 1998 to 2000. Dr. Guo's research areas include software engineering, context-aware systems, and data science. He received the best paper award at the 11th International Conference on Software Engineering and Knowledge Engineering. Dr. Guo was selected as a NASA Space Grant Fellow in 2008 and 2009 and worked at NASA AMES Research Center, Jet Propulsion Laboratory and Kennedy Space Center.



Yuehong Liao is a manager of software engineers at Virtu Financial. She has more than 20 years of experience in software development as well as a Master of Science degree in Computer Science. Before Virtu Financial, she held positions as vice president, software architect, and principal software engineer at Investment Technology Group and Arinc.

Appendix

In general, a Java program execution can be described as execution of a set of objects. So, a program can be described

as: $\varepsilon(p) = \bigcup_{i=1}^n o_i$. Here $\bigcup_{i=1}^n o_i$ are all the objects generated

during the execution of Java program p .

Each object o_i can be described by a six-tuple $\Omega = (S, s_0, M_{call}, T, \delta)$, where

(1) Ω is o_i life space

(2) S is a finite set of states of object o_i , $S = \bigcup_{i=1}^m s_i$

(3) s_0 is a single start state of the object s_0 , this state is usually created by *new*.

(4) M_{call} is a set of method calls; the method calls enable the object transition from one state to another state. In the meanwhile, the method is executed when the transition occurs,

$$M_{call} = \bigcup_{i=1}^m m_{call}^i .$$

(5) T is a set of valid timestamps, $T = \bigcup_{i=1}^m \tau_i$.

(6) δ is a transition: $[S \times M_{call}, T] \rightarrow [S, T]$. For example, at time t_0 , if the state of an object is s_0 and the object received a message and there is a method call m_{call}^0 , then at time t_1 , the state of the object will be s_1 after the method execution finishes. We have $[s_0 \times m_{call}^0, t_0] \rightarrow [s_1, t_1]$

Object states are identified by their attribute values at a specific time. Therefore, an object can have one or more states at a different timestamp τ_i . So, we can use the state transition to track the attribute value change of objects in a program. In this way, we can capture the side effects of our Java instrumentation, if there are any. If the state changes, the method calls may produce different results even though the method static property is not changed by the Java instrumentation.

Since the interactions between the Java objects are actually method calls, a Java program can be modeled as a sequence of method calls. The program execution is a set of ordered method calls $\{m_{call}^1, m_{call}^2, m_{call}^3, \dots, m_{call}^n\}$ in correspondence with a set of object states $\{s_1, s_2, \dots, s_n\}$. The program execution proceeds as follows: an object executes each method call in the program and produces a corresponding state transition. This continues until all calls in the program have been executed and calls are finished, in which case the execution ceases at that stage.

The state s_i at position i in the object state sequence represents the effect that the i th method call will have in the program sequence before the Java instrumentation and after the Java instrumentation. We use state-timestamp tuple (s_i, τ_i) to represent an object state is s_i at time stamp τ_i . Then, all the state-timestamp tuples consist of a partially ordered set:

$$S-T = \{(s_1, \tau_1), (s_2, \tau_2), (s_3, \tau_3), \dots, (s_n, \tau_n)\}$$

The object state transition is:

$$(s_i, \tau_i) \xrightarrow{M_{call}^i} (s_j, \tau_j)$$

We define the relation \preceq between the two elements of $(s_x, \tau_x) \preceq (s_y, \tau_y)$ as $\tau_x \preceq \tau_y$ and the timestamp is linear. So, we have following conditions satisfied in S-T:

- Reflexive: $\forall x [x \in S-T \rightarrow (s_x, \tau_x) \preceq (s_x, \tau_x)]$
- Transitive: $\forall x, \forall y, \forall z [(s_x, \tau_x) \preceq (s_y, \tau_y) \wedge (s_y, \tau_y) \preceq (s_z, \tau_z) \rightarrow (s_x, \tau_x) \preceq (s_z, \tau_z)]$
- Antisymmetric: $\forall x, \forall y [(s_x, \tau_x) \preceq (s_y, \tau_y) \wedge (s_y, \tau_y) \preceq (s_x, \tau_x) \rightarrow (s_x, \tau_x) = (s_y, \tau_y)]$

In this way, we can verify the program dynamic consistency by observing the object state space in the Java program; we note that the object state change is a result of the method calls.

If we use α to represent the pre-condition and β to represent the post-condition, then we have: $\{\alpha\} m_{call}^i \{\beta\}$, where α must be true before the method call m_{call}^i is called. It can be used to detect the violation of the pre-condition, and β must be true afterwards. It guarantees that the behavior of the method call m_{call}^i satisfies an expectation.

If we define the timestamp $\pi(m)$ “before” the method call m_{call} as $\tau_{before}(m)$ and the timestamp “after” the method call m_{call} as $\tau_{after}(m)$, then we have Java instrumentation in method m : $\pi(m)$ is $\{\alpha\} m_{call}^i \{\beta\}$ and $(\tau_{before}(m) \wedge \alpha) m_{call}^i \{\tau_{after}(m) \wedge \beta\}$ and we also have: $\{\alpha\} (\pi(m))_{call}^i \{\beta\}$ and $(\tau_{before}(\pi(m)) \wedge \alpha) (\pi(m))_{call}^i \{\tau_{after}(\pi(m)) \wedge \beta\}$

To verify the temporal relationships between the method calls, we need to collect the method call sequence. The method calls during an execution of a program is a sequence. We define it as $\{m_{call}^1, m_{call}^2, m_{call}^3, \dots, m_{call}^n\}$. If we use $\tau(m_{call}^i)$ to represent the timestamp of method call m_{call}^i , then we have a partially order sequence: $\tau(m_{call}^1) \preceq \tau(m_{call}^2) \preceq \tau(m_{call}^3) \dots \preceq \tau(m_{call}^n)$.

Our Java instrumentation will not change the order of the method call sequence that must satisfy the condition:

$$\forall i, \forall j [(\tau(m_{call}^i) \preceq \tau(m_{call}^j)) \rightarrow (\tau(\pi(m_{call}^i)) \preceq \tau(\pi(m_{call}^j)))]$$

Instructions for Authors

The International Journal of Computers and Their Applications is published multiple times a year with the purpose of providing a forum for state-of-the-art developments and research in the theory and design of computers, as well as current innovative activities in the applications of computers. In contrast to other journals, this journal focuses on emerging computer technologies with emphasis on the applicability to real world problems. Current areas of particular interest include, but are not limited to: architecture, networks, intelligent systems, parallel and distributed computing, software and information engineering, and computer applications (e.g., engineering, medicine, business, education, etc.). All papers are subject to peer review before selection.

A. Procedure for Submission of a Technical Paper for Consideration:

1. Email your manuscript to the Editor-in-Chief, Dr. Fred Harris, Jr. Fred.Harris@sce.unr.edu.
2. Illustrations should be high quality (originals unnecessary).
3. Enclose a separate page for (or include in the email message) the preferred author and address for correspondence. Also, please include email, telephone, and fax information should further contact be needed.

B. Manuscript Style:

1. The text should be, **double-spaced** (12 point or larger), **single column** and **single-sided** on 8.5 X 11 inch pages.
2. An informative abstract of 100-250 words should be provided.
3. At least 5 keywords following the abstract describing the paper topics.
4. References (alphabetized by first author) should appear at the end of the paper, as follows: author(s), first initials followed by last name, title in quotation marks, periodical, volume, inclusive page numbers, month and year.
5. Figures should be captioned and referenced.

C. Submission of Accepted Manuscripts:

1. The final complete paper (with abstract, figures, tables, and keywords) satisfying Section B above in **MS Word format** should be submitted to the Editor-in-chief.
2. The submission may be on a CD/DVD, or as an email attachment(s). **The following electronic files should be included:**
 - Paper text (required)
 - Bios (required for each author). Integrate at the end of the paper.
 - Author Photos (jpeg files are required by the printer)
 - Figures, Tables, Illustrations. These may be integrated into the paper text file or provided separately (jpeg, MS Word, PowerPoint, eps). title of the paper.
3. Specify on the CD/DVD label or in the email the word processor and version used, along with the title of the paper.
4. Authors are asked to sign an ISCA copyright form (<http://www.isca-hq.org/j-copyright.htm>), indicating that they are transferring the copyright to ISCA or declaring the work to be government-sponsored work in the public domain. Also, letters of permission for inclusion of non-original materials are required.

Publication Charges:

After a manuscript has been accepted for publication, the author will be invoiced for publication charges of \$50 USD per page (in the final IJCA two-column format) to cover part of the cost of publication. For ISCA members, \$100 of publication charges will be waived if requested.

