The South African Institute of Computer Science
and
Information Technology

Proceedings
of the
The 1997 National
Research and
Development
Conference
Towards 2000

Riverside Sun
Vanderbijlpark
13 & 14 November

Edited by
L.M. Venter
R.R. Lombard
©1997 Copyrights reside with the original authors who may be contacted directly

ISBN 1-86822-300-0

Printed and Binded by Xerox Printers, Potchefstroom

The views expressed in this book are those of the individual authors.
Foreword

This book contains a collection of papers presented at a Research and Development conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT). The conference was held on 13 & 14 November 1997 at the Riverside Sun, Vanderbijlpark. Most of the organization for the conference was done by the Department of Computer Science and Information Technology of the Vaal Triangle Campus, Potchefstroom University for Christian Higher Education.

The programming committee accepted a wide selection of papers for the conference. The papers range from detailed technical research work to reports of work in progress. The papers originate mainly from Academia, but also describe work done in and for Industry. It is hoped that the papers give a true reflection of the current research scene in Computer Science and Information Technology in South Africa. Since one of the aims of the conference is Research development, the papers were not subjected to a refereeing process.

A number of people spent numerous hours helping with the organization of this conference. In this regard, we wish to thank the members of the Organizing committee, and the Programming committee who had very little time to screen the abstracts and compile the program. A special thanks goes to the secretary of the department, Mrs Helei Jooste, whose very able work was interrupted by the birth of her first child.
Organizing Committee

Conference General Chairs
Prof. J.M. Hattingh (PU for CHE)

Organizing Chair
Prof. Lucas Venter (PU for CHE)

Organizing Committee
Mrs. S. Gilliland
Mr. J.P. Jooste
Mr. R.R. Lombard
Mrs. M. Huisman

Secretariat
Mrs. H. Jooste

Program Chair
Prof A de Waal (PU for CHE)

Program Committee
Prof. D. Kourie (UP)
Prof. C. Bornman (UNISA)
Prof. L.M. Venter (PU for CHE)
# Table of Contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>i</td>
</tr>
<tr>
<td>Organizing Committee</td>
<td>ii</td>
</tr>
<tr>
<td>List of Contributors</td>
<td>vii</td>
</tr>
<tr>
<td>Software Objects Change : Problems and Solution</td>
<td>1</td>
</tr>
<tr>
<td>S.A. Ajila</td>
<td></td>
</tr>
<tr>
<td>Liming-like Curve Constructions</td>
<td>26</td>
</tr>
<tr>
<td>M.L.Baart and R. McLeod</td>
<td></td>
</tr>
<tr>
<td>A Model for Evaluating Information Security</td>
<td>27</td>
</tr>
<tr>
<td>L. Barnard and R. von Solms</td>
<td></td>
</tr>
<tr>
<td>Integrating Spatial Data Management and Object Store Technology</td>
<td>31</td>
</tr>
<tr>
<td>S. Berman, S. Buffler and E. Voges</td>
<td></td>
</tr>
<tr>
<td>Metamodelling in Automated Software Engineering</td>
<td>32</td>
</tr>
<tr>
<td>S. Berman and R. Figueira</td>
<td></td>
</tr>
<tr>
<td>Using Multimedia Technology for Social Upliftment in Deprived Communities of Southern Africa</td>
<td>33</td>
</tr>
<tr>
<td>L. Bester and E. de Preez</td>
<td></td>
</tr>
<tr>
<td>Extending the Client-Server Model for Web-based Execution of Applications</td>
<td>36</td>
</tr>
<tr>
<td>L. Botha, J.M. Bishop and N.B. Serbedzija</td>
<td></td>
</tr>
<tr>
<td>Access Control Needs in an Electronic Workflow Environment</td>
<td>45</td>
</tr>
<tr>
<td>R.A. Botha</td>
<td></td>
</tr>
<tr>
<td>The Use of the Internet in an Academic Environment to Commercially Supply and Support Software Products</td>
<td>51</td>
</tr>
<tr>
<td>B. Braude and A.J. Walker</td>
<td></td>
</tr>
<tr>
<td>Explanation Facilities in Expert Systems Using Hypertext Technology</td>
<td>63</td>
</tr>
<tr>
<td>T. Breetzke and T. Thomas</td>
<td></td>
</tr>
<tr>
<td>Theoretical Computer Science: What is it all about, and is it of any relevance to us?</td>
<td>75</td>
</tr>
<tr>
<td>C. Brink</td>
<td></td>
</tr>
<tr>
<td>Representing Quadrics on a Computer</td>
<td>76</td>
</tr>
<tr>
<td>M.A. Coetzee and M.L. Baart</td>
<td></td>
</tr>
</tbody>
</table>
Global Optimization of Routes after the Process of Recovery  
M. Mphahlele and J. Roos  

Using a Lattice to Enhance Adaptation Guided Retrieval in Example Based Machine Translation  
G.D. Oosthuizen and S.L. Serutla  

Information Systems Development and Multi Criteria Decision Making / Systems Thinking  
D. Petkov, O. Petkova  

The Development of a Tutoring System to Assist Students to Develop Answering Techniques  
N Pillay  

Combining Rule-Based Artificial Intelligence with Geographic Information Systems to Plan the Physical Layer of Wireless Networks in Greenfield Areas  
K. Prag, P. Premjeeth and K. Sandrasegaran  

A Distributed Approach to the Scheduling Problem  
V. Ram and P. Warren  

More readings than I thought : Quantifier Interaction in Analysing the Temporal Structure of Repeated Eventualities  
S. Rock  

Ray Guarding Configuration of Adjacent Rectangles  
I. Sanders, D. Lubinsky and M. Sears  

Developing Soft Skills in Computer Students  
C Schröder, T. Thomas  

Information Security Awareness, a Must for Every Organization  
M. Thomson and R. von Solms  

Pla Va: A Lightweight Persistent Java Virtual Machine  
S Tjasink and S. Berman  

Beliefs on Resource-Bounded Agent  
E. Viljoen  

Object-Orientated Business Modelling and Re-engineering  
M. Watzenboeck
On Indexing in Case Based Reasoning Applied to Pre-Transportation Decision Making for Hazardous Waste Handling
K.L. Wortmann, D. Petkov and E. Senior

Author Index
List of Contributors

S.A. Ajila  
Department of Mathematics and Computer Science  
National University of Lesotho  
Roma, 180  
Lesotho

L. Baart  
Department of Mathematics  
Vaal Triangle Campus of the PU for CHE  
PO Box 1174  
Vanderbijlpark, 1900

L. Barnard  
Faculty of Computer Studies  
Port Elizabeth Technikon  
Private Bag X6011  
Port Elizabeth, 6000

S. Berman  
University of Cape Town  
Rondebosch, 7701

L. Bester  
Faculty of Computer Studies  
Port Elizabeth Technikon  
Private Bag X6011  
Port Elizabeth, 6000

J.M. Bishop  
Computer Science Department  
University of Pretoria  
Pretoria, 0002

L. Botha  
Computer Science Department  
University of Pretoria  
Pretoria, 0002

R.A. Botha  
Faculty of Computer Studies  
Port Elizabeth Technikon  
Private Bag X6011  
Port Elizabeth, 6000

B. Braude  
Software Engineering Applications Laboratory, Electrical Engineering  
University of the Witwatersrand  
Private Bag 3  
Wits, 2050

T. Breedtke  
Faculty of Computer Studies  
Port Elizabeth Technikon  
Private Bag X6011  
Port Elizabeth, 6000

C. Brink  
University of Cape Town  
Rondebosch, 7700

M. Bruynooghe  
Departement Computerwetenschappen  
Katholieke Universiteit Leuven  
Celestijnenlaan 200A  
B-3001 Heverlee  
Belgium

S. Buffler  
University of Cape Town  
Rondebosch, 7701

M.A. Coetzee  
Department of Mathematics  
PU for CHE  
Private Bag X6001  
Potchefstroom, 2520

R. Cools  
Katholieke Universiteit Leuven  
Celestijnenlaan 200A  
B-3001 Heverlee  
Belgium

E. de Preez  
Faculty of Computer Studies  
Port Elizabeth Technikon  
Private Bag X6011  
Port Elizabeth, 6000

D.A. De Waal  
Department of Computer Science and Information Systems  
PU for CHE  
Private Bag X6001  
Potchefstroom, 2531

B. Dekenh  
The Board of Executors

M. Denecker  
Departement Computerwetenschappen  
Katholieke Universiteit Leuven  
Celestijnenlaan 200A  
B-3001 Heverlee  
Belgium

M. Dunley-Owen  
Department of Information Systems  
University of Cape Town  
Rondebosch, 7700

R. Figueira  
University of Cape Town  
Rondebosch, 7701

A. Foster  
Department of Computer Science  
University of Cape Town  
Rondebosch, 7701

C. Gee  
Software Engineering Applications Laboratory, Electrical Engineering  
University of the Witwatersrand  
Private Bag 3  
Wits 2050
K. Sandrasegaran
Department of electrical Engineering
University of Durban-Westville
Private Bag X54001
Durban, 4000

C. Schoder
Faculty of Computer Studies
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth, 6000

M. Sears
Department of Mathematics
University of the Witwatersrand
Private Bag 3
Wits, 2050

E. Senior
International Center for Waste Technology
University of Natal, Pietermaritzburg
Private Bag X01
Scotsville, 3209

N. B. Serbedzija
GMD FIRST
Rudower Chausee 5
D-12489 Berlin
Germany

S.L. Serutla
Department of Computer Science
The University of Pretoria
Pretoria, 0002

T. Steyn
PU for CHE
Private Bag X6001
Potchefstroom, 2520

M. Thielscher
Fachgebiet Intellektik, Fachgebiet Informatik
Technische Hochschule Darmstadt
Alexanderstrasse 10
D-64283 Darmstadt
Germany

T. Thomas
Faculty of Computer Studies
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth, 6000

M. Thom-'s
Faculty of Computer Studies
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth, 6000

S. Tjasink
University of Cape Town
Rondebosch, 7700

E. Viljoen
Department of Computer Science and
Information Systems
University of South Africa
PO Box 392
Pretoria, 0001

E. Voges
University of Cape Town
Rondebosch, 7701

R. Von Solms
Faculty of Computer Studies
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth, 6000

A. J. Walker
Software Engineering Applications Laboratory,
Electrical Engineering
University of the Witwatersrand
Private Bag 3
Wits, 2050

P. Warren
Department of Computer Science
University of Natal
Private Bag X01
Scotsville, 3209

M. Watzenboeck
University of Botswana
Private Bag 0022
Gaborone
Botswana

K. L. Wortmann
Department of Computer Science
University of Natal, Pietermaritzburg
Private Bag X01
Scotsville, 3209
Developing Soft Skills in Computer Students

Cheryl Schröder, Theda Thomas
Faculty of Computer Studies, Port Elizabeth Technikon,
Private Bag X6011, Port Elizabeth, 6000, SOUTH AFRICA
cheryl@ml.petech.ac.za
theda@ml.petech.ac.za

Abstract

Today software developers are required to work closely with users when defining, developing and implementing systems. Interaction between software developers and users can influence the success or failure of the system development and implementation. The emphasis has shifted towards the need for IT developers who, not only have the technical capabilities and knowledge, but who also have strong interpersonal and business skills. IT professionals need to be able to form close working relationships and communicate effectively with both end-users and their IT peers. Tertiary institutions have, however, continued to emphasise technical and research skills and usually ignored the need for soft skills. It is not enough for tertiary institutions to tell students that they need these soft skills, opportunities must be created so that they can develop them. At the Port Elizabeth Technikon special modules have been created within the Information Technology course to afford students the opportunities to develop these skills. These modules run throughout the year for second and third year students. The emphasis is on providing an environment in which the students are encouraged to practise their soft skills. The paper will discuss the module content, methods used and our experiences in teaching these modules. The modules alone, however, are not sufficient for development of the necessary soft skills, therefore opportunities for this must be created within other subjects. The paper will also describe some of the methods that can be implemented to achieve this.

1. Introduction

Few professions have seen as rapid a change over the past decades as the field of information technology (IT). These changes are mainly due to rapidly changing technology. Not only are computers becoming smaller, cheaper, faster and more powerful, but IT professionals are continually searching for new methodologies, tools and techniques to help with the development of computerized systems. Despite the technologies and techniques that have been developed, today’s software development problems lie with IT’s inability to keep pace with user expectations.

IT professionals are required to work closely with users when defining, developing and implementing systems. Interaction between IT professionals and users can influence the success or failure of the computerised system. The emphasis has shifted towards a need for IT professionals who, not only have the technical capabilities and knowledge, but who also have strong interpersonal and business skills.

Concerns have been raised by business executives regarding the knowledge and skills required by IT professionals to function effectively in this changing technological and business environment, and how tertiary institutions can meet the changing needs of the profession [Ambler, 1995, p. 10].

This paper investigates how this dynamic IT world has affected the IT professional and what soft skills are needed by the new generation of IT professionals. An important aspect of this paper is to highlight how the Port Elizabeth Technikon has addressed the problem of affording IT students
the opportunities of developing the necessary soft skills.

2. Changing Role of the IT Professional

The days of the IT professional working in a vacuum are long gone. Today, IT professionals are required to work closely with users when defining, developing and implementing computerized systems. Interaction between IT professionals and users can influence the success or failure of system development and implementation. If conflict does occur between IT professionals and users, there is a strong possibility that the system may be delivered late, over budget or that users may avoid using the system [Green, 1989, p. 115; Joshi, 1996, p. 24; Shein, 1995, p. 17].

The IT profession has also experienced a paradigm shift in the nature of information management. “The traditional role of IS as the sole proprietor of information is being challenged by users within many organizations” [Farwell et al., 1992, p. 8]. As users become more sophisticated, they also become more demanding of their IT departments. These demands are for support and services as apposed to products. The objective of the IT department has evolved from being a provider of goods (e.g. data, reports) to being a provider of services (e.g. training, consulting).

According to Farwell et al. [1992, p. 9], the emphasis has shifted from purely technical capabilities and knowledge of traditional systems development techniques, towards strong interpersonal and business skills. It is imperative that IT professionals can form close working relationships and communicate effectively with both end-users and their IT peers. These business and interpersonal skills are often referred to as soft skills [Davis, 1993, p. 30; Farwell et al., 1992, p. 9; Watson et al., 1990, p. 24].

Studies of software developer skills suggest that soft skills are perceived by both software developers and users, to be more important than technical skills. Soft skills empower software developers and therefore, enables them to interact and communicate with people [Vitalari, 1985, p. 225].

3. Soft Skills Required by IT Professionals

The need for soft skills was confirmed by the results of a survey done in 1996 among 300 IT professionals in South Africa [Schröder & Thomas, 1997]. The respondents were asked to indicate to what extent they thought the various soft skills were important when developing software. A total of 126 questionnaires were returned, giving a 42% response rate. The respondents were asked to rate the skills as not important, of limited importance, important or extremely important. The following tables depict a summarised version of those results. The values specified in the tables represent the percentage respondents who indicated the skills to be Important or Extremely Important. The skills have been specified in the table in descending order of their importance according to the respondents.

3.1 Interpersonal Skills

*Interpersonal Skills* can be defined as those skills required to interact effectively with other people. The various Interpersonal Skills were grouped together into the following skill categories:

- **Communication skills**
  Those skills required to share or interchange thoughts, ideas and opinions in a manner that is understood by others.
• Negotiation skills
  Those skills required to engage in bargaining or discussion with a view to coming to an agreement.

• Leadership skills
  Those skills required to get the work done, while keeping all parties involved, satisfied and motivated.

• Empowerment skills
  Those skills required to improve self-reliance, self-management and self-motivation.

<table>
<thead>
<tr>
<th>Communication Skills</th>
<th>Leadership Skills</th>
<th>Negotiation Skills</th>
<th>Empowerment Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active listening</td>
<td>Goal setting</td>
<td>Decision making</td>
<td>Creative thinking</td>
</tr>
<tr>
<td>Writing skills</td>
<td>Decision making</td>
<td>Resolving conflict</td>
<td>Time management</td>
</tr>
<tr>
<td>Conduct meetings</td>
<td>Managing change</td>
<td>Diplomacy</td>
<td>Goal setting</td>
</tr>
<tr>
<td>Presentation skills</td>
<td>Team building</td>
<td>Assertiveness</td>
<td>Self motivation</td>
</tr>
<tr>
<td>1-1 Training</td>
<td>JAD facilitation</td>
<td>Persuasion skills</td>
<td>Stress management</td>
</tr>
<tr>
<td>Interviewing skills</td>
<td>Motivating others</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public speaking</td>
<td>Managing meetings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delegation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Interpersonal Skills

3.2 Problem Solving Skills

Problem Solving Skills are those skills required to resolve, explain or clarify a problem or difficulty. Kreitner [1995, p. 247] explains that we are all problem solvers. Nevertheless, this does not mean we are all good problem solvers or even for that matter, that we know how to solve problems systematically. Most daily problem solving is done on a haphazard, intuitive basis. A difficulty arises, we look around for an answer, jump at the first workable solution that comes to mind, and then move onto other things. Although this sequence of events qualifies as a problem solving process, and it works quite well for informal daily activities, it is certainly not suitable when dealing with the various problems that can arise during the software development process.

<table>
<thead>
<tr>
<th>Problem Solving Skills</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical thinking</td>
<td>100</td>
</tr>
<tr>
<td>Investigative skills</td>
<td>97</td>
</tr>
<tr>
<td>Creative thinking</td>
<td>96</td>
</tr>
<tr>
<td>Decision making</td>
<td>95</td>
</tr>
<tr>
<td>Brainstorming</td>
<td>89</td>
</tr>
<tr>
<td>Interviewing skills</td>
<td>79</td>
</tr>
<tr>
<td>Research skills</td>
<td>77</td>
</tr>
<tr>
<td>S.W.O.T. Analysis</td>
<td>61</td>
</tr>
</tbody>
</table>

Table 2. Problem Solving Skills
3.3 Project Management Skills

*Project Management Skills* are those skills required to assist with the planning, organizing and controlling of projects so that they are completed on time and within budget.

Power programming languages and fast compilers do not produce good software. Advanced development methods and software engineering practices may help, but offer no guarantees. In the real world of software and applications development, even the most rapid prototyping takes time, and even mildly sophisticated systems need the contribution of multiple developers. Under these real circumstances, how the personnel of programming are organized and managed become crucial factors in the success or failure of projects. Only good people, well organized and well managed to enhance their productivity and the quality of their work, can produce good systems [Constantine & Lockwood, 1993].

<table>
<thead>
<tr>
<th>Project Management Skills</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time management</td>
<td>99</td>
</tr>
<tr>
<td>Leadership skills</td>
<td>98</td>
</tr>
<tr>
<td>Communication skills</td>
<td>97</td>
</tr>
<tr>
<td>Trustworthiness</td>
<td>95</td>
</tr>
<tr>
<td>Negotiation skills</td>
<td>92</td>
</tr>
</tbody>
</table>

Table 3. Project Management Skills

4. Life Skills Course Content and Structure

Keeping in mind the new, dynamic IT world, tertiary institutions need to ask themselves if they are providing the right type of education for future IT professionals. An industry as innovative and progressive as IT needs continuous knowledge and skill updating.

The challenge facing tertiary institutions is twofold:

i. To keep the IT curriculum up to date; and
ii. to ensure that IT graduates develop the much needed soft skills.

Moad [1995, p. E1] claims that there is a huge gap between what the IT departments at universities are teaching their students and what IT divisions need. A developing country, like South Africa, has many factors that influence curricula development. Apart from the fact that it is a combination of a first and a third world country, the many different cultures, eleven official languages and underdeveloped schooling system all need to be taken into consideration when any curricula changes are made. Students in their secondary education, especially those from disadvantaged backgrounds, often do not have opportunities to develop their basic communication, social and other soft skills.

This paper makes no attempt at prescribing what changes need to be made to the IT curricula. It rather tries to discuss how the Port Elizabeth Technikon has addressed this need for soft skills and to highlight some of the methods available to develop IT students’ soft skills. The Technikon has promoted a two pronged attack on this problem. The first is a series of two modules offered at
second and third year level to help students gain competence and confidence in their ability to interact with people and work in teams. The second was to incorporate into the curricula of other subject areas, the opportunities for students to develop their soft skills. These two facets will be described below.

4.1 Description of the Course

At the Port Elizabeth Technikon special modules have been created within the Information Technology course to afford students the opportunities to develop the soft skills identified by industry. These modules run throughout the year for second and third year students. The emphasis is on providing an environment in which students are encouraged to practise their soft skills.

4.1.1 Second Year Students

The module for second year students is included as part of the subject Information Systems II. This Life Skills II module is run separately from the formal Information Systems II lectures, although the students are given a mark (based on attendance, participation and completed assignments) which forms part of their year mark. The soft skills covered in the Life Skills II module are as follows:

- **Communication Skills**
  - Active listening
  - Body language
  - Feedback
  - Writing skills
  - Conducting meetings

- **Negotiation Skills**
  - Persuasion skills
  - Assertiveness
  - Decision making
  - Resolving conflict

- **Leadership Skills**
  - Delegation
  - Managing meetings
  - JAD facilitation

- **Problem Solving Skills**
  - Brainstorming
  - Lateral thinking
  - Mind mapping
  - Logical thinking
  - Interviewing skills

- **Project Management Skills**
  - Time management
  - Trustworthiness
  - Communication skills
  - Negotiation skills

- **Empowerment Skills**
  - Creative thinking
  - Goal setting
  - Self motivation

The Life Skills II module consists of a double lecture (hour and a half) held weekly. Although each skill was “taught” during specific class sessions, many skills were reinforced at various points throughout the course. Because the course emphasis was on skill application and practice rather than on theory and research, a variety of instructional techniques were used. These included role plays, games, brainstorming, large and small group discussions and audiovisual presentations.

Apart from the Life Skills II module, students are also given opportunities within the formal subjects to develop certain soft skills (i.e. group work, project management skills, writing skills, research skills, investigative skills and presentation skills).

Although most of the students saw the need for this course, some students attended this course under duress and with a negative attitude, thereby putting a damper on things for the rest of the students. Those students interested in the course, participated with enthusiasm and reaped the benefits. Another problem encountered was the size of the classes, which ranged from 20 to 30
students. The lecturer should act as a facilitator and monitor the activities of the students. The class size made it very difficult to do this effectively, especially when the students were required to break into smaller groups.

As the lecturer involved in developing this course, I believe if some of the following changes could be implemented, the course could be improved:

i. Involve several lecturers as facilitators for each class or decrease the size of each class.
ii. Make attendance voluntary.
iii. Enlist people who are experts in their field (eg. student psychologists, motivational speakers).
iv. Require students to keep a journal containing reflections, ideas, questions, thoughts, feelings, opinions and experiences related to the topics and experiences of the course. Journals should be due at the end of each session. These journals can provide the lecturer insight into the success of each session and to encourage students to reflect on what they have learned.

4.1.2 Third Year Students

Life Skills III is included as a separate module of the Information Technology Skills subject at the third year level. Again the students are given a mark based on attendance, participation and completed assignments. Most of the lectures allocated for this module are spent on more practical issues which are relevant for graduating students. The areas covered in the Life Skills III module are as follows:

- **Entrepreneurship**
  - Business plan
  - Marketing
  - Motivation

- **Contract Assignments**
  - Charging
  - Do's and don'ts
  - Legal contracts

- **Job Preparation**
  - Preparation for job interview
  - Job interview
  - Follow-up of job interview

- **Communication Skills**
  - Writing skills (curriculum vitae)
  - Presentation skills
  - Public speaking

- **Negotiation Skills**
  - Conflict management
  - Assertiveness
  - Decision making

- **Financial Preparation**
  - Insurance policies
  - Unit trusts
  - Stock market

The Life Skills III module consists of a double lecture (hour and a half) held weekly. The above-mentioned topics were presented by various experts.

Because the topics covered are relevant to the students most of them found these sessions interesting, although a number found the Entrepreneurship workshop of 10 weeks a little too long and the Public Speaking exercise tedious. Again the number of students in the class presents difficulties when making use of games, role playing and group discussions.

Possible solutions to these problems are:

i. Involve several lecturers as facilitators when necessary; and
ii. Reduce the number of students in a session.
4.2 Interactive Sessions

4.2.1 Games/Role Plays

An important method of developing soft skills is interactive workshops where games or role plays are used to highlight and develop specific skills. Games are fun and this makes them an enjoyable way of learning. There is no reason that such an universal method of learning in childhood should not continue into adult life. Young people always enjoy leisure activities. This enthusiasm for fun and friendship can be tapped for teaching soft skills.

Irrespective of the skill being “taught”, the students are encouraged either to sit in a random setting or split into smaller groups. This seating arrangement allows a much greater degree of eye contact and allows people the freedom of voicing their own opinions without being judged. Although students are encouraged to participate as much as possible, they are not forced to reveal their innermost feelings. After the game or role play has ended, the lecturer involves the students in a discussion about the meaning and benefits of the exercise.

Learning by direct personal experience has far more impact than being advised by someone else’s experience, which is inevitably second-hand. First-hand experience makes it easier for someone to relate to whatever they have learnt from the game and to apply it to everyday life [Bond, 1986, p. 11]. Some soft skills that have lent themselves to this development method are active listening, feedback, problem solving, brainstorming, negotiating, nonverbal communication and trust.

Because games take place in an atmosphere of fun and the levels of anxiety remain relatively low, people do not feel defensive. This means they are more likely to hear what is said and to be able to evaluate it for themselves. Games encourage unity in the group and a sense of identity. This is particularly true if the games selected encourage cooperation and improved communication in the group.

4.2.2 Workshops

Experts from both the Technikon and local business community were enlisted to present specific workshops on relevant subjects of expertise. All of these workshops were tailored specifically for our IT students. Some successful workshops run to date covered topics such as entrepreneurship, writing skills, self empowerment and group dynamics.

These workshops are run over several weeks. Most of them are of an interactive nature, requiring the students to participate both individually and in small groups. Not only do the students exercise the skill being presented in the workshop, they are also given other opportunities in several other subjects to practise that same skill. For instance, in the writing skills workshop the students were taught how to prepare and write reports and proposals. After the completion of this workshop, the students were given writing assignments for other subjects (Information Systems II and Systems Software II). Apart from enabling the students to practise their writing skills, these assignments also presented opportunities to develop their research and presentation skills.
5. Methods of Developing Soft Skills

5.1 Group / Team project work

Port Elizabeth Technikon IT students are involved in team-oriented projects right from their first year of study. This year, for example, in their first year students were asked to do group activities using the Internet as a tool. In their second year the students have to develop a system using a 4GL such as Microsoft Access and write up the necessary documentation. The third year requires that they again work in groups to produce a system from the requirements phase to the implementation phase. The students are encouraged to write these systems for “real users” to stimulate a real work experience, thereby offering the students an opportunity to develop various communication and negotiation skills. These projects are also used to teach the students various methodologies and project management techniques.

Apart from using group work to reinforce normal academic studies, it is an excellent way to develop many soft skills without the students even realizing it. Because deadlines are set for various milestones of the project, the students learn to develop goal setting, time management and stress management skills. Working on projects in this manner also enables students to develop problem solving, empowerment and leadership skills. Not only does this system have to be developed and implemented, various documentation manuals have to be written and the students must do a formal presentation of the system.

Development of soft skills occurs in all the major categories, that is, Communication Skills, Negotiation Skills, Leadership Skills, Problem Solving Skills, Project Management Skills and Empowerment Skills.

5.2 Groupwork and JAD Sessions

Groupwork is also used in the classroom situation to let students practise their skills and to help them to think about their ideas. By defending their ideas to others in the group and by listening to others students, students are able to get a deeper understanding of the topics they must learn. Groupwork also enables the student to realise when he/she has some misconceptions about the work.

Students seem to find group work more interesting than working on their own and are more positive towards the task at hand. Because many students experience less anxiety in the group situation, they tend to tackle things that they would not normally have the confidence to do. Groupwork enables students to improve their communication, negotiation and leadership skills while also developing their problem-solving skills.

Although group work can be beneficial, there are also problems associated with the traditional methods of doing group work. These are mentioned below:

- Individuals with strong personalities tend to dominate the group.
- Often members of the group then do little or no work.
- Pressures within the group may force people to conform even if they think differently, without a true compromise being reached.
- The group is often more social than productive.

The Technikon has been experimenting with letting students practise their JAD facilitation skills...
while learning design topics in Information Technology. The students are given instruction in the facilitation process as well as the design techniques. They are then given the opportunity to practise these techniques in class.

To explain how facilitated group work would work, the example of teaching students how to design databases will be used. This is done in the Information Systems II course at the Port Elizabeth Technikon. The students use entity-relationship (ER) diagrams to design the conceptual database and then from the ER diagrams they construct the database tables.

The students are taught the basics of ER diagrams in the traditional manner in class. The students are then given problems before the JAD class so that they can try to understand what is being asked - not to try to do it on their own. Practical groups of 20 are further subdivided into groups of 6 or 7, which are each assigned to one section of the classroom where there is a white board for them to use.

The students take turns being the facilitator, the scribe or the people who have to define the system. The facilitator stands at the board and will ask the people who have to define the system (i.e. the other students) what data they feel should go into the system, what entities and attributes they use. They then discuss the solution and try to design the database. The facilitator must make sure that the terminology used is understood by all members of the group. He/she must also ensure that all group members participate and that one person does not dominate the group. The facilitator is also supposed to help resolve any conflicts and make sure that consensus is reached. (Note that this should solve many of the problems discussed under the more conventional methods of group work.)

The lecturer makes sure that the facilitators do their job and checks the results of the design by discussing any problems with the group as a whole. Only in exceptional circumstances should the lecturer get involved in solving the problem with the student. This usually happens when the students are going round and round in circles without being able to any result.

All the students benefit as all the students are involved. The facilitator must ensure that a student is not allowed to sit back and allow the others to do the work for him/her. The lecturer, in turn, must ensure that the facilitator is doing his/her work. JAD sessions, not only help develop JAD facilitation skills, but also several other soft skills. These include active listening, conducting meeting, assertiveness, persuasion skills and team building.

5.3 Classroom Opportunities

A lesson learnt early in our experimentation with having life skills modules in the Information Technology course was that the numbers of students and the amount of time available meant that the students needed additional opportunities to practise the skills. The staff that taught the students were asked to incorporate opportunities for the students to develop their soft skills during the formal classes. As mentioned before, projects and groupwork, are two of the methods used but other examples of the type of situations that were used are given below:

Although a short course on giving presentations is given in the life skills course, the students only have the opportunity to speak for a very short time in the course. Lecturers of some of the formal courses let the student do presentations on selected topics relevant to their subject. The students are expected to do their own research of the topic, using the library and the Internet. They, thus, not only improve their presentation skills but are also able to develop their research and
investigative skills.

Writing skills is another topic covered in the life skills component. The students are given writing assignments in class and it is emphasized that in business, the first impact a person makes on others is often through words on paper - in the form of a letter, report or memo. The students are expected, not only to write "articles" but also to be able to write recommendations to management, proposals and documentation that is easy to read and maintain. It is, once again, not only the writing skills that are developed in this way but also the investigative skills of the students.

6. Summary

The world of the IT professional has changed. It is no longer sufficient for IT graduates to be only technically proficient. They should also be proficient in the skills needed to interact with other people and empower themselves. Skills like communication, empowerment, problem solving and team building have become very important.

It is the task of tertiary institutions to provide IT students with the necessary opportunities to developing these much needed soft skills. Group work, JAD sessions, interactive workshops, classroom opportunities and community involvement are but few of the ways that can be used by tertiary institutions to develop these soft skills.

REFERENCES


Moad, J. 1995. Change of course: IT execs, stymied by IS schools, are growing their staff's 'soft skills'. PC Week, vol. 12, no. 16, pp. E1-E3.


Information Technology (IT) has become part and parcel of the business world today. In fact, with the increasing sophistication of networks and the continued growth in electronic commerce, it will continue becoming an ever larger factor in the future (Fitzgerald, 1995). Information has become the most important asset in most organizations and must therefore be protected properly. Traditionally, physical and technical control measures were adequate, to a large extent, to protect information resources. With the distribution and interlinking of information systems and the fact that many users of information systems are not fully computer literate, it became necessary to augment the physical and technical controls with operational ones (von Solms, 1997). The objective of operational controls is to dictate and discipline the behaviour of users to ensure secure utilization of information and the related information systems.

Every organization should run a complete security awareness program. This program should educate and train all users of information, from top management to end-users, how to ensure that the confidentiality, integrity, and availability of information is never compromised (Violino, 1996). The different levels of employees in the organisation obviously have different functions and needs with regard to IT. For this reason, it is envisaged that this Information Security Awareness and Training Program will have to be tailored to suit these different groupings. The groupings that were identified were top management, IT staff, and the end-users. These groupings were identified because their needs for an Information Security Awareness and Training Program were significantly different to warrant different segments within the program being developed for each of them.

The program for top management will focus mainly on the importance of information security with respect to it’s importance in the daily operations of the organisation as well as the threats from internal and external users and their potential for causing harm to the organisation (ISO & IEC, 1996, GMITS part 1). Also included would be the potential future threat of accreditation to electronic
trading. This section is basically aimed at convincing top management of the importance of information security and the threats to the organisation should it be compromised, as a means of ensuring their wholehearted support for the program.

The program for IT personnel will include the section that top management received as well as concentrating more on the technical aspects relating to the specific types of threats that could be encountered, and the building of safeguards into the various systems. In order to accomplish this, the following will need to be addressed (ISO & IEC, 1995, GMITS part 2):

- Integration of security in system life cycle,
- Assignment of security roles and responsibilities,
- Selection of risk analysis strategy option,
- Making security recommendations,
- Implementation of safeguards,
- Follow-up to ensure that the safeguards are working properly,

The end-user program’s contents are similar to the contents of many existing Information Security Awareness and Training Programs. It will contain the information security policy and any information from the top management program which would help to bring home the importance attached to this program as well as the procedural aspects of maintaining security, e.g. changing passwords regularly, backing-up crucial data etc (ISO & IEC, 1996, GMITS part 3).

The Information Security Awareness and Training Program should not only educate the users about the information security policy and the related operational controls and procedures, but should also change the general behaviour of information users. The latter is an aspect that has been neglected in the past (Kabay, 1994). There is a vast amount of research that has been conducted in the area of social psychology which pertains specifically to behaviour modification. The way in which attitudes, beliefs, and behaviour are interlinked in a person’s attitude system plays an important role in the effective structuring of the Information Security Awareness and Training Program. Research has proven that there are methods that can be applied to make the process of changing behaviour more effective (Zimbardo & Leippe, 1991). These methods include:

- Changing behaviour as a means to change attitudes.
- Using persuasion to change attitudes.
- Using social learning, shaping, operant learning, and group pressure in order to gain
conformity.

- The effects of obedience, reciprocity, and commitment in changing behaviour.

This information should be included in the Information Security Awareness and Training Program and the suggested program will include some of the methods which have been proven as very effective in changing people’s behaviour.

Security awareness and training have been relegated to a relatively minor position with regards to information security within any company. This situation will have to change as the human factor in information systems is taken more seriously. Properly trained and motivated employees can be the most effective counters to possible security threats within a company.

REFERENCES


PLaVa: A Lightweight Persistent Java Virtual Machine

S. Tjasink and Dr S. Berman
University of Cape Town
October 1, 1997

1 Introduction

This paper describes the implementation and tuning of an orthogonally persistent Java Virtual Machine (JVM) for small computers. It is based on a JVM that was targetted at a digital satellite television decoder with 1 megabyte each of ROM and RAM.

Orthogonally persistent languages allow all program values to persist on disk, regardless of their type; and enable program code to have the same form regardless of the longevity of the data on which it operates. They improve productivity because there is no need for programmers to learn a separate database language or to manage data transfer and translation when using persistent objects.

Java has received a lot of attention since its release by Sun Microsystems in 1995. Many features of Java make it well suited to orthogonal persistence: source code is compiled to a platform-independent bytecode; there is implicit heap management, strong typing and useful programming constructs like threads, exceptions and classes. Java is also appropriate for embedded applications because of its safety and security features.

Several researchers have now begun working on persistent Java implementations. This paper demonstrates the feasibility of accessing a persistent Java store from very small computers which obtain applets over the net.

The paper starts with an overview of persistent systems, followed by an explanation of how the Java language itself has been affected by the addition of persistence. The next section discusses how the support for persistence has been added to the virtual machine. That is followed by a section discussing related work and we conclude with long-term goals.

2 Overview of persistent systems

Orthogonally persistent languages allow values to persist by automatically saving them to a persistent store and enable the program code to have the same form regardless of the longevity of the data on which it operates. [AM95] gives an outline of orthogonally persistent systems and provides a tutorial for those commencing research or study in the field. It summarises the benefits of persistence over conventional database applications and covers the basics of persistence, existing achievements and current research issues.
The principles of persistence are given here. They must all be adhered to when designing an orthogonally persistent system. This section is just a quick introduction to the concepts involved before our implementation is discussed in detail.

**The Principle of Persistence Independence** The form of a program is the same whether it is manipulating persistent or transient data.

**The Principle of Data Type Orthogonality** Data of all types should be allowed to persist. This complies with the language design principle of type completeness.

**The Principle of Persistence Identification** There should be a well defined system for identifying which data is persistent and it should be orthogonal to the type system. This principle is often implemented using identification by reachability. In this case, at least one root of reachability is needed, from which other reachable objects can be found.

When an attempt is made to use an object that is referenced from another object, it must be loaded into memory from the persistent store before it can be used. There are usually two forms of addresses that are used. These are:

- **Persistent Identifier (PID)** that is used to refer to persistent objects in the store. It is usually the index of the object in the persistent store. These identifiers cannot be used directly by an executing program and must be converted to local addresses (see below).

- **Local Address (LA)** which is a pointer into physical memory. These are used to refer to objects that have been loaded into memory for use by the program.

If an LA is encountered then execution should not be interrupted. If a PID is encountered then steps have to be taken to load the required object (called an object fault, similar to a page fault in operating system parlance). First, we check to see if the referenced object has already been loaded (referred to as a residency check). If not, we need to load it. If it has already been loaded because it was used by another part of the program then of course we do not have to load it (this is called a false object fault).

When an object is loaded, the mapping between its PID and its LA is added to a table to facilitate future residency checks. The PID that was referenced is overwritten with the LA of the loaded version of the object (called swizzling). This is done so that next time the reference is encountered, it will be in LA form and execution can proceed without being interrupted.

Storage of data into the persistent store also has to be automated. When an application requests a checkpoint, all loaded persistent objects that have been changed need to be written back to the store. Also, any new objects that have been created that are now reachable from a persistent root need to be promoted to persistence. The same applies (recursively) to all new objects that are reachable via these.

## 3 Language design issues

### 3.1 The Java language and persistence

Java is a modern, object-oriented programming language [AG96, GJS96]. It is loosely based on C++ [Str91] but does not include the commonly confusing features and constructs that have made
• **PStore openStore(String storeName):** This method is used to open an existing persistent store of the given name. If the store is not found then a StoreNotFoundException is thrown. An instance of the PStore class representing the store is returned.

When a store has been created or opened, objects can be added to the store to be used as persistent roots. Roots are identified by a name, which can be used to retrieve them later. These root objects can also be removed as roots, which means that the object and any objects reachable from it will no longer persist unless they are reachable via another root. Methods that perform these functions are provided; they can be called on any instance of the PStore class that has been obtained via CreateStore or OpenStore.

• **void addRoot(String name, Object ob):** Adds the specified object as a new persistent root. It is given the specified name so that it can be retrieved at a later stage.

• **Object getRoot(String name):** Returns the specified root if a root of that name exists. It will need to be typecast to its actual type.

• **void removeRoot(String Name):** Removes the named root from the store’s list of roots.

In order to convert a standard Java program to a persistent version of the program, only a few changes need to be made:

1. Open a persistent store (or create a new store if one does not already exist).
2. Retrieve the root object for the program that you wish to run.
3. Insert calls to stabilise the store (discussed in the following section) at relevant points in the program in order to update the store if needed. This is not essential as the store is stabilised implicitly when the program execution ends.

A short example showing how the store is opened and the objects accessed follows. In practice, the calls for opening the store and accessing roots throw exceptions if errors occur during execution and these exceptions should be caught to deal with the problems.

```java
/* Open the store called "cddb.store" */
PStore p = PStore.openStore("cddb.store");

/* Ask for the root called "CDB1"... must be cast to its correct type */
CDBthing mydb = (CDBthing) p.getRoot("CDB1");

/* Run the code in the object you’ve accessed. All other persistent objects that are referenced in the code will automatically be faulted in during execution. */
mydb.run();

/* Stabilise to write all objects back to the store */
p.stabiliseAll();
```
The program that is started in the third line of code above is a standard Java program that maintains a small database of information about compact discs including title, artist and the names of songs. All that the program needs to do is to construct a collection (e.g. a tree, array or linked list) containing the database and have a reference to this collection from the object mydb that is loaded above. Because the tree is accessible from mydb and mydb is a registered root of the store, all data in the collection will automatically be saved during a stabilise.

3.3 Transactions
At present only a global stabilise method, stabiliseAll, is provided in the PStore class. This can be used to write all loaded persistent objects back to the store (and to promote all the newly reachable objects).

In the PJama system [AJDS96] developed by the Glasgow group, an internal as well as an external transaction API is specified. The external API is provided by the TransactionShell class and its specialisations. These should satisfy most application programmers, but programmers who wish to define new transactional models by defining their own specialisations of TransactionShell can use the internal API to do so.

4 Persistence mechanism design
This section presents in some detail the design and implementation of the PLaVa persistent virtual machine. First, a very brief overview of relevant aspects of the JVM architecture is given. This is followed by a discussion of how swizzling has been added to the machine and then the actual object faulting. After this, update tracking and writing back to the store are dealt with. The way in which Java heap garbage collection interacts with the persistence mechanisms is then explained and finally we discuss the matter of native methods in persistent code.

4.1 The Java Virtual Machine
As mentioned in the section on language issues, changes were made to a standard JVM in order to support persistence. The JVM is a stack-based machine that executes the Java bytecode. This means that it has an operand stack and most of the bytecode instructions take their operands from this stack and push their results onto it. Each method also has a number of local variables that it uses.

4.2 When to swizzle
Swizzling is done whenever a reference to a persistent object is loaded onto the operand stack from an object. If the field that is being loaded onto the stack is a reference and is in PID form then its corresponding LA is loaded onto the stack. This will involve faulting the object in if it is not already resident. In both cases the PID reference that is being used is overwritten at its source with the LA so that all successive uses will be able to use the LA directly.

This scheme is essentially a lazy, direct swizzling scheme using object faulting. It turns out to be similar to swizzling on discovery as described in [WD92]. In general compiled languages discussed in [WD92], actually discovering the original location of a reference so that it can be swizzled can be
problematic. In Java this is simplified by the fact that discoveries can be isolated to the execution of two bytecode instructions.

- `getfield`: This instruction is used for loading the contents of fields of all types from class instances onto the operand stack. In the case where the field is a reference, the field is swizzled. Reference fields are identified by the field's signature. Its signature starts with 'L' in the case of a reference to a class instance or with a '[' in the case of a reference to an array.

- `aaload`: This instruction is used for loading a reference onto the operand stack from an array of references. The corresponding array element is swizzled in this case.

Java bytecode instructions that use references always take these references from the operand stack. Thus the system of making sure that only LAs appear on the operand stack will ensure that Java bytecodes only ever encounter LAs. Because all parameters that are passed to methods are taken from the operand stack, all references that are passed (including the target object) will be LAs. This automatically satisfies target object residency and formal parameter residency as discussed in [MH94, Hos96]. Procedure result residency will also automatically be enforced as the `return` instruction that returns references from methods takes the return value from the current method's operand stack.

### 4.3 Mechanics of object faulting

When an object is faulted in, its P/P to LA mapping is added to the resident persistent object table (RPOT) so that future residency checks will find it. The table is a hash table addressable by P/P. It does not need to be addressable by LA as well, because reverse mappings are not needed in PLaVa. An object's P/P is stored in a field in the object header and when it is needed it can be read from there without having to query the RPOT.

A macro is used to determine whether or not a reference is a P/P, currently all P/Ps are negative integers. This is used to determine whether or not the reference needs to be swizzled. When an object is faulted in, the object's P/P is stored in a field in that object's header as described above. The space used by the P/P is less than the space that would be required by an index to make the RPOT addressable by LA as well as by P/P.

Swizzling can also be turned off. In this case, P/Ps are never overwritten with LAs and a residency check needs to be done every time one is used. This will slow normal program execution because of the extra residency checks that need to be done, but references will not have to be deswizzled when objects are written back to the store. In the PLaVa system deswizzling is a quick operation so this is not much of a performance gain.

### 4.4 Recording updates

All objects that are changed need to be marked as such so that the store can be updated when required by a stabilise.

The execution of bytecodes that write to fields of objects or elements of arrays was changed so that operands could reflect their altered status. If an object is changed then its `PERSIST.Updated` flag is set. If a reference field in an object or an element in an array of references is changed then its `PERSIST.RefsUpdated` flag is set. These flags are stored in the object header with a number of other flags.
Figure 2: Object faulting stage 1. A reference field in PID form is dereferenced. The PIDLAM (called RPOT in PLaVa) is queried to see if the object corresponding to that PID has already been loaded. In this case it has not.

Figure 3: Object faulting stage 2. The object corresponding to the PID is loaded from the store into the heap. Its PID and its LA (its address in the heap) are inserted into the PIDLAM.

Figure 4: Object faulting stage 3. The reference field that was used to reference the object is overwritten with the loaded object’s LA (swizzling).
The two separate flags are used to make the process of updating changed objects more efficient. If an object is marked as having had one of its reference fields updated then when it is written back to the store, the objects that it references must be checked to see if they have also been updated. In the case where none of an object's reference fields have been updated its fields do not have to be checked and it can simply be written to the store. Instructions affected are:

- putfield: This instruction needs to flag the changed class instance as updated. It is marked with the PERSISTRefsUpdated flag if it is a reference field that is updated (determined by looking at the field's signature). In the implementation of putfield, the signature had to be checked to see if fields were 32 or 64 bits side, so this is not an extra expense.

- za.store: (includes iastore, lastore, fastore, dastore, bastore, castore and sastore) These instructions need to flag the changed array as updated (with the PERSISTRefsUpdated flag in the case of aastore).

4.5 Object updating

During a stabilise, all objects that are marked as updated need to be written back to the store. Only those objects that have been marked as having a reference field updated need have their fields scanned for references. When a reference field is found, the object that it references is updated as well (this will include promotion to persistence if it is currently transient).

If the reference field is an LA then it must be overwritten with its PID so that it can be written back to the store correctly. This unswizzling just requires reading the PID from the object header since every persistent object stores its own PID in its object header. When all of an object's reference fields (or elements in the case of an array) have been stabilised then it is written to the store.

Each object that has an entry in the RPOT is considered and the following criteria are used for determining the objects that need to be written to the store:

- An object is not written back if it is already persistent and is not flagged as updated.

- An object is updated in the store if it is persistent and flagged as updated. If it is flagged as having had its reference fields updated then each of the reference fields (or elements if it's an array) is checked as well, with these update criteria being applied to the referenced objects. The reference fields have to be checked because new objects might now be reachable and these new objects will have to be promoted to persistence.

- An object is promoted to persistence if it is not already persistent, regardless of whether or not it is marked as updated. This ensures that all new objects that become reachable from persistent objects also become persistent.

Currently, when an object needs to have its reference fields checked, all fields need to be examined in order to see which of them are reference fields. This requires examining each field's signature. Checking reference fields could be made more efficient by clustering them to either the beginning or the end of the object's field storage memory. A bit map could also be maintained to track updates to fields or just to provide quick differentiation between reference fields and other fields. The bit map update-tracking method will require extra processing during actual execution although it will require fewer checks during a stabilise.
Null PID references are represented exactly the same as local null values (i.e. 0). No distinction needs to be made between the two as null values are invalid for dereferencing in both cases.

When objects are updated in the store, their swizzled fields are unswizzled back to PID form in which they are required to be when put into the store because the object is copied directly from the heap into the store. This means that these fields will have to be reswizzled when they are used again in the course of program execution. These are likely to cause false object faults on reswizzling because the objects will still be resident (unless they become victims of garbage collection before they are referenced again). The previous values of all reference fields could be stored in order to restore them to their LA values after the write-back to the store, but this would involve extra memory use during the stabilise operation. This has been added as an option that can be enabled or disabled for testing purposes.

### 4.6 Heap garbage collection

As has been mentioned previously, memory allocation and de-allocation in Java is implicit and the JVM uses a garbage collector to reclaim the memory that is occupied by objects that are no longer useful to the program that is running.

The standard (non-persistent) JVM on which the persistent JVM is based uses a simple mark-and-sweep garbage collection system for disposing of unusable objects. All objects that are reachable are marked. This involves checking all objects that are accessible from local variables and the operand stacks and recursively checking all those objects’ fields. When all reachable objects have been marked, those that are unmarked are disposed of.

The victim-selection process that the garbage collector uses is currently the same as in the non-persistent virtual machine. The garbage collector uses reachability and PIDs will not be recognised as valid memory addresses, so objects that are referenced by PID will not be considered reachable. PID references to memory-resident objects will only appear if a stabilise has occurred since the

---

Figure 5: Writing back to the store. The grey objects have not been updated and therefore do not have to be written back. Cr’s reference to Dr has been overwritten with a reference to the new object Fr. Fr is put into the store as F and D is no longer reachable from anything else in the store, so it will be disposed of by the garbage collector at some stage.
reference was last used. Naturally, reswizzling on stabilise (as discussed above) would remedy this. It may be better not to do this however, because the stabilise will have ensured that the latest version of that object was written to disk and hence the valuable memory that it was occupying can be used for another purpose. We will investigate the effects of this more fully.

When a persistent object is discarded from memory it is stabilised and removed from the RPOT. If a non-persistent object is discarded then it is not written to the store since if it is not reachable from any objects in memory then it is not possible for it ever to become reachable from any persistent object.

At the moment only unreachable objects are thrown away. This can be changed so that reachable persistent objects can be written back to the store and removed from memory in order to free more heap if needed. However, this would require dealing with all LAs that reference discarded objects.

4.7 Persistence and native methods

4.7.1 The problem with native methods

Because the Java bytecode is platform-independent and interpreted by the JVM, some other mechanism has to be provided for access to platform-specific services. These services include file I/O, any textual or graphical output and network access. To provide these services, Java allows native methods to be run. These are methods that have been written in C and compiled on for the architecture on which the JVM is running. The Java programmer calls these in the same way as calls to the JVM to check a method call to see if it is a Java method or a native method and handle the invocation accordingly.

Because native methods are compiled to platform-dependent machine code and are not executed by the JVM, the normal mechanisms for discovering and swizzling PIDs and tracking updates that have been discussed cannot be applied. Programmers who have to use native methods must handle these aspects themselves. The Java Native Interface (JNI) provides a number of standard method calls that can be made from native methods to perform certain tasks. We have added a few calls to the JNI to allow PLaVa native methods in programs to handle persistence.

If these calls are not used then unreliable program behaviour will result. Because of this, the use of native methods in PLaVa programs is strongly discouraged and they should only be used where absolutely necessary.

4.7.2 Swizzling

Although all references in the method's parameters will be in LA form since they were taken from the operand stack, references to other objects from these objects may still be in PID form. Obviously, references should be in LA form before they can be used. To swizzle all references in objects that are passed to native methods would not be sufficient since any arbitrary chain of references could be followed by the native code.

The following calls must be used in native code before any reference is used (excluding of course those in the method's actual parameters). The calls will check a reference and swizzle it if it is in PID form, returning the LA so that it can be used.

• void SwizzleObjectField(JNIEnv*, jobject, jfield) This swizzles the requested field in the requested object and returns its LA. If the field is not a reference field then it returns
NULL without having taken any action.

- **void** SwizzleArrayElement(JNIEnv*, jobjectArray, jint) This swizzles the requested element of the requested array and returns its LA. If the array is not an array of references then it returns NULL without having taken any action.

The return value of a native method will be put onto the operand stack so this must also be converted to an LA so as not to introduce PIDs onto the operand stack.

### 4.7.3 Recording updates

Since changes can be made to objects from within native methods, we have added a couple of calls to the JNI to deal with this.

- **void** MarkUpdated(JNIEnv*, jobject) This marks the specified object as having been updated.
- **void** MarkRefsUpdated(JNIEnv*, jobject) This marks the specified object as having had one or more of its references updated.

These calls must be called appropriately whenever an object is updated in a native method. If they are not called then there will be no record that the object has been updated and its changes will not be written into the database, resulting in inconsistency.

### 5 Related work

The object faulting architecture discussed in this paper above is used in a number of existing systems. Early systems were PS-ALGOL [ACC82, BC85] and LOOM [KK83] which was a persistent implementation of Smalltalk [GR83].

Another approach is to use a page faulting system, such as in [Wil91, WK92, VD92]. In this approach, objects are loaded into memory a page at a time. When a reference to an object in a non-resident page is dereferenced, that is made resident (thus bringing all the objects in it into memory). This system can utilise the built-in page fault mechanism of operating systems.

Swizzling can be done either **eagerly** or **lazily**. Eager swizzling swizzles all required objects (or a subset thereof) in advance whereas lazy swizzling swizzles objects only when they are needed. Swizzling can also be either **direct** or **indirect**. Direct swizzling brings the referenced object into memory when the reference is swizzled. Indirect swizzling only brings objects into memory when they are needed. In this latter case, the swizzling process is more complex, with the first step being to point the reference to a *proxy object*. When the object is used, the second step is done and this involves replacing the proxy object with the actual object that is loaded from the store. A study comparing the performance of various swizzling techniques in the context of the Mneme store [Mos90] is given in [Mos91, HM93] and in the context of the E programming language [Ric90] in [WD92].

Other persistent implementations of Java include PJama [AJDS96, Jor96] which is being developed by the Glasgow group in association with Sun Microsystems. The PJama system's architecture is similar to that of PLaVa in its basic design. The JavaSPIN approach [KMRW96] combines Java with the existing SPIN (Support for Persistence, Interoperability and Naming) framework. This
enables JavaSPIN to interact seamlessly with other languages that have SPIN extensions. There are also attempts to interface Java with relational databases such as [dST96, Gru96]. These systems should provide a Java interface to legacy data but it is unlikely that they will be able to attain a high degree of orthogonality.

References to most of the work that has been done in the field of persistent languages can be found in [AM95].

6 Conclusion

6.1 Summary

We have extended an existing JVM to support basic persistence. With the addition of a few lines of code to a standard Java program, it can be converted into a persistent application that will run on the PLaVa machine. The rest of the program need not be changed because the object faulting and update tracking are automatically done by the PLaVa virtual machine.

The virtual machine implements persistence by detecting which references that are being dereferenced are in PID form and converting them to LA form so that they can be used. This conversion involves automatically loading the object from the store. When an object is changed, a flag in its header is set to record that it has been updated. Updated objects are all written back to the store during a stabilise operation. All new objects that have become reachable from existing persistent objects are added to the store as well.

Decisions that were taken during the design were aimed towards keeping memory usage low to facilitate use on a small machine. Further investigation into these matters will be carried out as explained in the following section.

6.2 Future work

Now that we have demonstrated the feasibility of using the PLaVa system on a small machine, our emphasis turns to performance measurements. We'll test the effectiveness of various configurations of the virtual machine with regards to speed and space efficiency and tradeoffs between them. This involves selecting and deselecting alternative strategies for residency checking, object faulting, caching and pointer swizzling.

We wish to use an efficient, universal store with our virtual machine; stores that should be available in the near future include PJSL [PAD+97] which is being developed by the Glasgow group for their PJama system and Stephen Blackburn's store which is based on his PSI interface model [BS97, Bla97]. PJSL is likely to be the most efficient as its design is based on experiences with the previous version of the PJama store and it is being supported by Sun Microsystems.

Other outstanding work includes the development of a remote store that will allow the PLaVa system to run on a small machine without a local disk. This machine would obtain all its data from a remote persistent store via a network. The internal mechanism of the persistent virtual machine that accesses the store will not need to be changed. The function calls that interact with the store will have to be modified to contact the remote store instead of accessing a local store.

We hope to implement the remote store interface using Java's remote method invocation (RMI) package, with the remote store being a Java object on a remote machine. We will investigate the efficiency of this approach and consider alternatives if the performance is not acceptable.
References


[GR83] A. Goldberg and D. Robson. Smalltalk-80: The Language and its Implementation. Addison-Wesley, 1983.


