COMPUTATION TOOLS 2012

The Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking

ISBN: 978-1-61208-222-6

July 22-27, 2012

Nice, France

COMPUTATION TOOLS 2012 Editors

Torsten Ullrich, Fraunhofer Austria Research GmbH - Graz, Austria

Pascal Lorenz, University of Haute Alsace, France

COMPUTATION TOOLS 2012

Foreword

The Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2012), held between July 22 and 27, 2012 in Nice, France, continued an event under the umbrella of ComputationWorld 2012 dealing with logics, algebras, advanced computation techniques, specialized programming languages, and tools for distributed computation. Mainly, the event targeted those aspects supporting context-oriented systems, adaptive systems, service computing, patterns and content-oriented features, temporal and ubiquitous aspects, and many facets of computational benchmarking.

We take here the opportunity to warmly thank all the members of the COMPUTATION TOOLS 2012 Technical Program Committee, as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to COMPUTATION TOOLS 2012. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the COMPUTATION TOOLS 2012 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that COMPUTATION TOOLS 2012 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the areas of computational logics, algebras, programming, tools, and benchmarking.

We are convinced that the participants found the event useful and communications very open. We hope Côte d'Azur provided a pleasant environment during the conference and everyone saved some time for exploring the Mediterranean Coast.

COMPUTATION TOOLS 2012 Chairs:

COMPUTATION TOOLS Advisory Chairs
Kenneth Scerri, University of Malta, Malta
Jaime Lloret Mauri, Polytechnic University of Valencia, Spain
Radu-Emil Precup, "Politehnica" University of Timisoara, Romania

COMPUTATIONAL TOOLS 2012 Industry/Research Chairs
Torsten Ullrich, Fraunhofer Austria Research GmbH - Graz, Austria
Zhiming Liu, UNU-IIST, Macao

COMPUTATION TOOLS 2012

Committee

COMPUTATION TOOLS Advisory Chairs

Kenneth Scerri, University of Malta, Malta
Jaime Lloret Mauri, Polytechnic University of Valencia, Spain
Radu-Emil Precup, "Politehnica" University of Timisoara, Romania

COMPUTATIONAL TOOLS 2012 Industry/Research Chairs

Torsten Ullrich, Fraunhofer Austria Research GmbH - Graz, Austria
Zhiming Liu, UNU-IIST, Macao

COMPUTATION TOOLS 2012 Technical Program Committee

François Anton, Technical University of Denmark, Denmark
Florin Avram, University of Pau, France
Henri Basson, University of Lille North of France (Littoral), France
Steffen Bernhard, TU-Dortmund, Germany
Ateet Bhalla, NRI Institute of Information Science and Technology, Bhopal, India
Narhimene Boustia, Saad Dahlab University - Blida, Algeria
Manfred Broy, Technical University of Munich, Germany
Luca Cassano, University of Pisa, Italy
Emanuele Covino, Università di Bari, Italy
Hepu Deng, RMIT University - Melbourne, Australia
Eugene Feinberg, Stony Brook University, USA
Cynthia Vera Glodeanu, Institute of Algebra / Technische Universität Dresden, Germany
Luis Gomes, Universidade Nova de Lisboa, Portugal
Rajiv Gupta, University of California - Riverside, USA
Fikret Gurgen, Bogazici University - Istanbul, Turkey
Cornel Klein, Siemens AG - Munich, Germany
Stano Krajci, Safarik University - Kosice, Slovakia
Giovanni Lagorio, DISI/University of Genova, Italy
Tsung-Chih Lin, Feng-Chia University, Taichung, Taiwan
Paolo Masci, Queen Mary, University of London, UK
Cecilia E. Nugraheni, Parahyangan Catholic University - Bandung, Indonesia
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
Corrado Priami, CoSBi & University of Trento, Italy
Evgenia Smirni, College of William and Mary - Williamsburg, USA
James Tan, SIM University, Singapore
Torsten Ullrich, Fraunhofer Austria Research GmbH, Austria
Miroslav Velev, Aries Design Automation, USA
Zhonglei Wang, Karlsruhe Institute of Technology, Germany

Marek Zaremba, Universite du Quebec en Outaouais - Gatineau, Canada
Naijun Zhan, Institute of Software/Chinese Academy of Sciences - Beijing, China

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Fast Efficient Fixed-Size Memory Pool
No Loops and No Overhead

Ben Kenwright
School of Computer Science
Newcastle University
Newcastle, United Kingdom,
b.kenwright@ncl.ac.uk

Abstract--In this paper, we examine a ready-to-use, robust, and computationally fast fixed-size memory pool manager with no-loops and no-memory overhead that is highly suited towards time-critical systems such as games. The algorithm achieves this by exploiting the unused memory slots for bookkeeping in combination with a trouble-free indexing scheme. We explain how it works in amalgamation with straightforward step-by-step examples. Furthermore, we compare just how much faster the memory pool manager is when compared with a system allocator (e.g., malloc) over a range of allocations and sizes.

Keywords-memory pools; real-time; memory allocation; memory manager; memory de-allocation; dynamic memory

I. INTRODUCTION

A high-quality memory management system is crucial for any application that performs a large number of allocations and de-allocations. In retrospect, studies have shown that in some cases an application can spend on average 30% of its processing time within the memory manager functions [1–4] and in some cases this can be as high as 43% [5].

However, speed is only one of the features we look at for a good memory manager; in addition, we are also concerned with:

- Memory management must not use any resources (both memory or computational cost)
- Minimize fragmentation
- Complexity (ideally a straightforward and logical algorithm that can be implemented without too many problems)
- Ability to verify and identify memory problems (corruption, leaks).

Nevertheless, the majority of applications use a general memory management system, which tries to provide a best-for-all solution by catering for every possible scenario. For some systems, where speed is critical, such as games, these solutions are overkill. Instead, a simplified approach of partitioning the memory into fixed sized regions known as pools can provide enormous enhancements, such as increased speed, zero fragmentation and memory organization.

Hence, we focus on a fixed-pool solution and introduce an algorithm that has little overhead and almost no computational cost to create and destroy. In addition, it can be used in conjunction with an existing system to provide a hybrid solution with minimum difficulty. On the other hand, multiple instances of numerous fixed-sized pools can be used to produce a general overall flexible general solution to work in place of the current system memory manager.

Alternatively, in some time critical systems such as games; system allocations are reduced to a bare minimum to make the process run as fast as possible. However, for a dynamically changing system, it is necessary to allocate memory for changing resources, e.g., data assets (graphical images, sound files, scripts) which are loaded dynamically at runtime. The sizes of these resources can be determined prior to running. This then makes the fixed memory pool manager ideal. Alternatively, as mentioned a range of pools can be used for a best-fit approach to accommodate varying size data.

Naive memory pool implementations initialize all the memory pool segments when created [6][7]. This can be expensive since it is usually necessary to loop over all the uninitialized segments. Our algorithm differs by only initializing the first element and so has little computational overhead when it is created (i.e., no loops). Hence, if a memory pool is only partially used and destroyed, this wastes fewer processor cycles. Furthermore, for dynamic memory systems where partitioned memory is constantly created and destroyed this initialization cost can be important (e.g., pools being repeatedly partitioned into smaller pools at run-time).

In summary, a memory pool can make an application execute faster, give greater control, add greater flexibility, enable greater customizability, greatly enhance robustness, and prevent fragmentation. To conclude, this paper presents the implementation for a straightforward, fast, flexible, and portable fixed-size memory pool algorithm that can accomplish O(1) time complexity memory allocation and de-allocation that is ideal for high speed applications.

The fixed-size pool algorithm we present boasts the following properties:

- No loops (fast access times)
- No recursive functions
- Little initialization overhead
- Little memory footprint (few dozen bytes)
- Straightforward and trouble-free algorithm
- No-fragmentation

- Control and organization of memory

The rest of the paper is organized as follows. First, Section II discusses related work. In Section III, we outline the *contribution of the paper*, followed by Section IV, which gives a detailed explanation of how the memory pool algorithm works. Section V discusses practical implementation issues. Section VI outlines some limitations of the method. Section VIII gives some benchmark experimental results. Finally, Section IX draws conclusions and further work.

II. Related Work

The subject of memory management techniques has been highly studied over the past few decades [8–12][13]. A whole variety of techniques and algorithms are available, while some of them can be overly complex and confusing to understand. On the other hand, the technique we present here is not novel, but is a modification of an existing technique [14][6][13]; whereby loops and initialization overheads are removed; this makes the resulting algorithm extremely fast and straightforward. The method also boasts of being one of the most memory efficient implementation available since it has very little memory footprint and while giving an O(1) access time. We also give an uncomplicated implementation in C++ in the appendix.

Memory pools have been a well known choice to speed-up memory allocations/de-allocations for high-speed systems [15][16][17]. Zhao et al. [18] grouped data together from successive calls into segregated memory using memory pools to reduce pre-fetch latency. An article by Applegate [19] gave a well-defined overview of the various methods and advantages of high-performance memory in portable applications and the advantages of memory pools. Further discussion in Malakhow [20] outlines the advantages of memory pools and their applicability in high-performance multi-threaded systems.

While we present a similar single-pool allocator to Hanson [7], our algorithm is more clear-cut and makes it easier to customize for an ad-hoc implementation.

Additionally, performance considerations are discussed by Meyers [21], e.g., macros and monolithic functions, that can be applied to gain further speed-ups and gain greater reliability while incorporating good coding practices. A comparison of the computational cost of a memory management system implemented in an object orientated language (e.g., C++) is less efficient than one implemented in a functional language (e.g., C) [3][22]; however, we implemented our fixed-size memory pool in C++ because we believe it makes it more re-usable, extensible and modular.

III. Contribution

The contribution of this paper is to demonstrate a practical, simple, fixed-size memory pool manager that has no-loops, virtually no-memory overhead and is computationally fast. We also compare the algorithm with the standard system memory allocator (e.g., *malloc*) to give the reader a real-world computational comparison of the speed differences. The comparison emphasizes just how much faster a simple and smart algorithm can be over a more complex and general solution.

IV. How It Works

We explain how the fixed-size memory pool works by defining what information we have and what information we need to calculate (to help make the details more understandable, see Figure 1 and Figure 2 for illustrations).

When the pool is created, we obtain a continuous section of memory that we know the start and end address of. This continuous range of memory is subdivided into equally sized memory blocks. Each memory blocks address can be identified at this point from the start address, block-size, and the number of blocks.

This leaves the dynamic bookkeeping part of the algorithm. The algorithm must keep track of which blocks are used and un-used as they are allocated and de-allocated.

We begin by identifying each memory block using a four-byte index number. This index number can be used to identify any memory location by multiplying it by the block size and adding it to the start memory address. Hence, we have 0 to n-1 blocks; where n is the number of blocks).

The bookkeeping algorithm works by keeping a list of the *unused* blocks. We only need to know which blocks are being unused to find the used blocks. This list of unused blocks is modified as blocks are allocated and de-allocated.

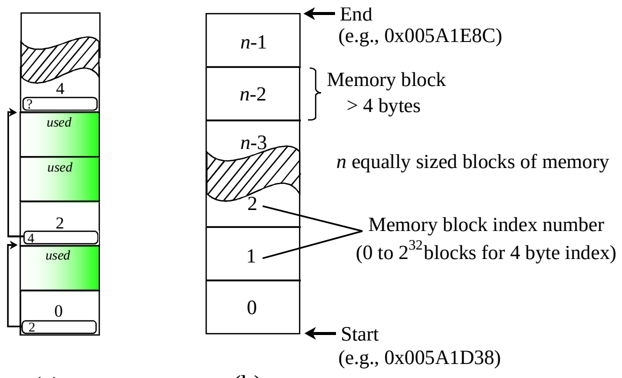


Figure 1. (a) Illustrate how the unused memory is linked together (the unused memory blocks store index information to identify the free space). (b) Example of how memory is subdivided into a number of n blocks.

However, we avoid the cost of initializing and link together all the unused blocks. We alternatively initialize a variable to inform us of how many of the n blocks have been appended to the unused list. Whereby, at each allocation unused blocks are appended to the list and the number of initialized blocks variable is updated (see Figure 1).

The list uses no additional memory. Since the memory blocks that are being kept track of are not being used, we can store information inside them. Each unused block stores the index of the next unused block. The pool keeps track of the head of the unused linked chain of blocks.

For this bookkeeping algorithm to work a minimum size constraint must be imposed on the memory blocks. The individual memory blocks must be greater than four-bytes. This is because each unused memory block will hold the index of the next unused memory block to form a linked list all the unused blocks.

Therefore, each unused block holds the index to the next unused block and so on. Our pool stores the index to the head of the first unused block. For each allocation an unused block is removed from the list and returned to the user. We keep track of the head of the unused list of blocks and is updated after each allocation. Alternatively, when a block de-allocated we can calculate its index from its memory address then append it to the list of unused blocks.

We only add new unused blocks to the list during allocation. We keep track of how many blocks have been added to the list and stop appending new blocks when we have reached the upper limit. This avoids any loops and initialization costs since we only initialize blocks as we need them. In summary, as we allocate blocks, further unused blocks are initialized and appended to the list as needed.

Figure 1 is used to help further illustrate the working mechanism of the algorithm; in addition, Listing 1 gives the pseudo-code.

A. *Step-by-Step Example*

To follow the fixed-pool method through, we use a simple step-by-step example shown in Figure 2 to see the algorithm in action.

We create a fixed pool with four-blocks. We show how unused blocks and member variables change during the process of creation, allocation and de-allocation sequentially from the start (identifying uninitialized and unknown memory with question marks – the three variables in Figure 2 represent the necessary variables used by the pool for bookkeeping).

B. *Verification*

Writing a custom memory pool allocator can be both difficult and error prone. While the fixed size memory pool algorithm is relatively straightforward and trouble-free to implement, it is advised that additional verification and sanity checks be incorporated to ensure a robust and reliable implementation.

These sanity and safety checks can come at the cost of extra memory usage and increased computational cost. For example, running experimental simulations of system allocations within the debugger would increase allocation

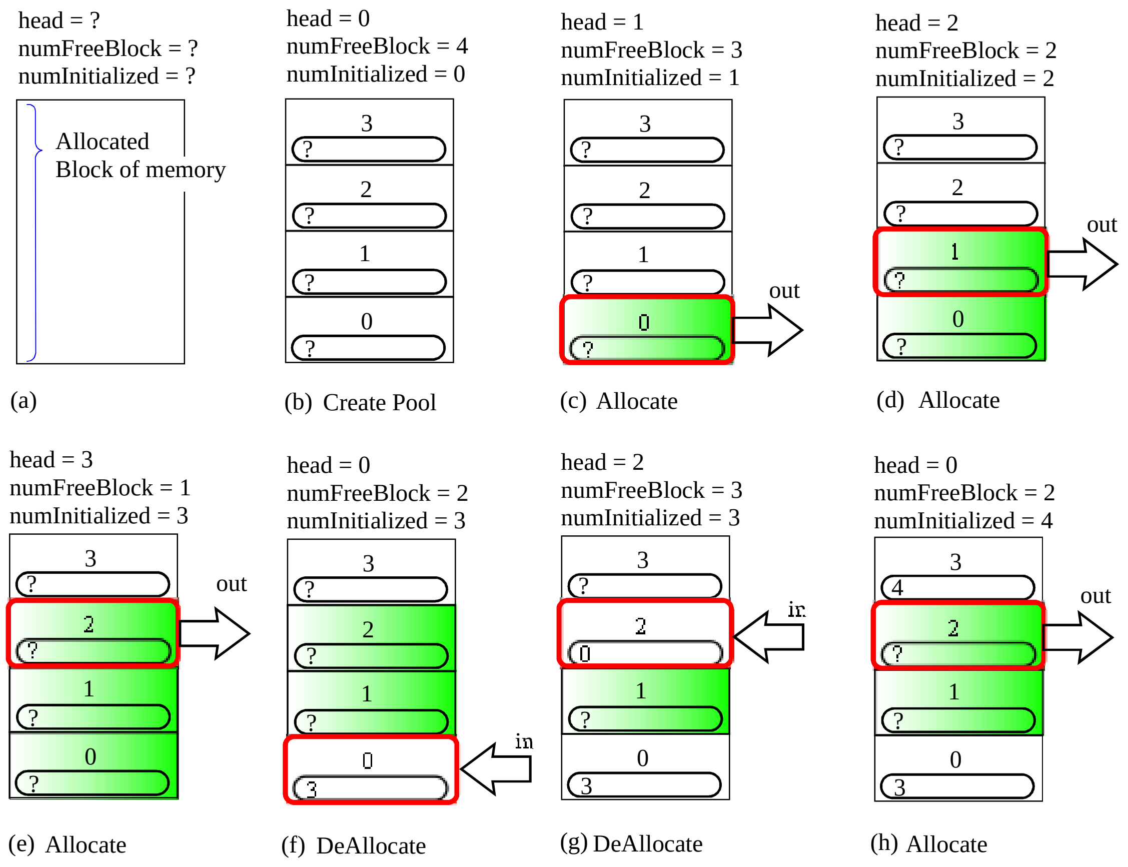


Figure 2. Step-by-step example of the memory pools internal workings for a simple 4 slot segmentation - the sequence of events from (a) to (h).

times by up to 100 times (see Figure 3 and Figure 4, which show the different allocation times of running within and outside the debugger).

The memory pool gives the maximum amount of control and can implement various custom checks. They can be enabled and disable at will, and can be less computationally expensive than the system memory checks enabling builds to run at fast speeds while gaining debug information.

For example, the de-allocated memory addresses can easily be verified, since each memory address must be within an upper and lower boundary of the continuous memory region. Furthermore, the de-allocated memory address must be the same as one of the addresses from the divided memory blocks. In addition, memory guards can be added to include boundary checks by adding a pre and post byte signature to each block. These memory guards can be checked globally (i.e., for all blocks) and locally (i.e., currently deleted block) to identify problems and provide sanity checks.

Furthermore, leaks can be found by extending and embedding the memory guards to store additional information about the allocation; for example, the line number of the allocation.

V. Implementation

We implemented the code in C++. The pool was created using create/destroy functions instead of the constructor/destructor so that the pool could be dynamically resized without destroying and recreating the pool each time it needed reconfiguring.

The implementation has four essential public functions: Create, Destroy, Allocate, and De-allocate.

The fundamental source code that implements the fixed-size memory pool is given in the appendix. To keep the source code as straightforward and as easy to read as possible all the validation and sanity check code has been excluded.

```
Initialize pool
[ Block of memory is allocated or obtained ]
1.  Store the start address, number of blocks and the number of uninitialized unused block
Allocator
2.  Check if there any free blocks
3.  If necessary - initialize and append unused memory block to the list
4.  Go to the head of the unused block list
5.  Extract the block number from the head of the unused block in the list and set it as the new head
6.  Return the address for the old block head
De-allocator
7.  Check the memory address is valid
8.  Calculate the memory addresses index id
9.  Set its contents to the index id of the current head of unused blocks and set itself as the head
```

Listing 1. Pseudo-code for pool.

Combining the fixed pool allocator with an existing memory management system in C++ by overloading the *new* and *delete* operators would give better performance with the minimum amount of disruption, since 38% of execution time can be consumed by the dynamic memory management [3]. This ad-hoc approach works by checking the memory allocation size within the new operator; if space is available inside the pool, and the size is within a specified tolerance the memory is taken from the pool, but if not, the general system allocator is called to supply the memory.

Additionally, the greatest care must be exercised to ensure that classes and structures in C++ that are allocated and de-allocated by the fixed-size pool allocator have their constructors and destructors manually called.

VI. Limitations

The fixed-pool memory manager relies on it being assigned a continuous block of memory. This can be a serious limiting factor if the assigned block of memory is scattered around.

Furthermore, we have focused on the algorithm and not discussed hardware limitations. For example, a page fault can result in an access time being 10,000 times slower than normal. Additionally, we have not addressed the issue of using the memory pool in a multi-threaded environment. This also raises the question of how the memory manager can be managed across multiple cores and the subject of scalability.

As well, the presented memory pool implementation is limited to systems with direct access to the memory and so cannot be implemented in managed memory environments (e.g., Java and C#).

The amount of memory requested from the fixed-size pool allocator can raise two major problems. Firstly, if the requested memory is dramatically smaller than the slot-size then lots of memory will be wasted. Secondly, and worse, if the requested memory is greater than the slot-size then it is impossible to allocate memory from the pool. Nevertheless, to combat these problems and to reduce memory wastage and largely miss-sized allocations an ad-hoc solution can be used. Whereby, a general system allocator in conjunction with multiple fixed-size pools would help to reduce memory wastage while still benefiting from the pool speedups.

On the other hand, it should be pointed out, that a general memory management system could become slower and fragmented over time. Whereby, a suitable block of memory would require considerable searching overhead, in addition to, small chunks of unsuitable and unusable memory being scattered around.

VII. Resizing

The fixed-size memory pool holds a list of unused memory blocks. This list resides in the unused memory and is incrementally extended when a memory block is allocated. Hence, if more memory blocks are needed than are available, and further additional memory follows the end of the continuous memory pools allocation, the pool can be extended effortlessly with little cost by updating its member variables. Once the member variables have been updated to incorporate the new end memory address it will automatically extend and fill the new region of memory during block allocations.

The algorithm currently always initializes the next unused memory block during the allocation call. However, an additional check can be added to avoid initialization of further unused blocks if they are not needed. For this reason, we could identify the maximum allocated number of unused blocks. Then, optionally the large pool of memory could be resized-down without needing to destroy and re-create the pool.

VIII. EXPERIMENTAL RESULTS

The algorithm itself is simple with no loops, no recursion, and little computational cost, and produces extremely fast allocations and de-allocations. To get a ballpark idea of how much faster the memory pool manager can be over a general memory system; we allocated and de-allocated a range of memory chunks as shown in Figure 3 and Figure 4. The figures show the fixed-pool allocator to be ten times faster than the general system allocator, and a thousand times faster when running within a debug environment.

IX. CONCLUSION AND FURTHER WORK

We have shown a fundamental, unsophisticated, raw-and-ready memory pool algorithm that produces remarkably fast speeds with nearly no-overhead and boasts the added advantage of being straightforward to understand and easy to implement. The fixed-size memory pool provides the best solution for processes such as games, which assume that relatively few memory allocations happen, and when they do happen they are of a deterministic size and need to be extremely fast (for example, graphical assets, particles, network packets and so on).

The Keep It Short and Simple (K.I.S.S) approach was a target goal for the fixed-size memory pool since the presented algorithm is a fundamental building block for constructing, if desired, a more elaborate and flexible memory manager.

Further work is needed to investigate if the algorithm could be optimised to use less decisional logic (i.e., if statements). In addition to exploring hardware considerations (e.g., caching, paging, registers, memory alignment, threading) and how the algorithm can be enhance to accommodate platform specific speed-ups.

REFERENCES

[1] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," *Lecture Notes in Computer Science*, pp. 1–1, 1995.

[2] B. Zorn, "Empirical measurements of six allocation-intensive C programs," *ACM Sigplan notices*, no. July, 1992.

[3] B. Calder and D. Grunwald, "Quantifying behavioral differences between C and C++ programs," *Journal of Programming*, vol. 2, no. 4, pp. 313–351, 1994.

[4] W. Li, S. Mohanty, and K. Kavi, "A Page-based Hybrid (Software-Hardware) Dynamic Memory Allocator," *IEEE Computer Architecture Letters*, vol. 5, no. 2, pp. 13-13, Feb. 2006.

[5] E. Berger and B. Zorn, "Reconsidering custom memory allocation," *Sciences-New York*, 2002.

[6] J. Deng, "Why to use memory pool and how to implement it," *(3 July)*, 2008. [Online]. Available: http://www.codeproject.com/Articles/27487/Why-to-use-memory-pool-and-how-to-implement-it. [Accessed: 06-Jan-2011].

[7] R. D. Hanson, *C Interfaces and Implementation*. O'Reilly Safari, 1997.

[8] A. Gorine, "Memory Management and Embeded Databases," *Dr. Dobb's (1st December)*, 2005. [Online]. Available: http://drdobbs.com/database/184406355. [Accessed: 06-Jan-2011].

[9] J. Bartlett, "Inside Memory Management: The choices, tradeoffs, and implementations of dynamic allocation," *IBM (16th November)*, 2004. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-memory/. [Accessed: 06-Jan-2011].

[10] J. L. Risco-Martín, J. M. Colmenar, D. Atienza, and J. I. Hidalgo, "Simulation of high-performance memory allocators," *Microprocessors and Microsystems*, vol. 35, pp. 755-765, Aug. 2011.

[11] E. D. Berger, B. G. Zorn, and K. S. McKinley, "Composing high-performance memory allocators," in *ACM SIGPLAN Notices*, 2001, vol. 36, no. 5, pp. 114–124.

[12] D. Atienza, J. M. Mendias, S. Mamagkakis, D. Soudris, and F. Catthoor, "Systematic dynamic memory management design methodology for reduced memory footprint," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 2, pp. 465-489, Apr. 2006.

[13] D. Lea, "A memory allocator," 1996. [Online]. Available: http://gee.cs.oswego.edu/dl/html/malloc.html. [Accessed: 06-Jan-2011].

[14] Stephen Cleary, "Boost C++ Memory Pool Librarys," *(5th December)*, 2006. [Online]. Available: www.boost.org/libs/pool; http://www.boost.org/doc/libs/1_47_0/libs/pool/doc/concepts.html. [Accessed: 06-Jan-2011].

[15] D. Bulka and D. Mayhew, *Efficient C++: performance programming techniques*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[16] D. Gay and A. Aiken, *Memory management with explicit regions*, vol. 33, no. 5. ACM, 1998, pp. 313-323.

[17] K. Frazer, *C++ in action: industrial-strngth programming techniques*. SIGSOFT Softw. Eng. Notes, 2002.

[18] Q. Zhao and R. Rabbah, "Dynamic memory optimization using pool allocation and prefetching," *ACM SIGARCH Computer Architecture*, 2005.

[19] D. A. Applegate, "Rethinking Memory Management : Portable techniques for high performance," *Dr. Dobb's (June)*, 1994. [Online]. Available: www.ddj.com/184409253/. [Accessed: 06-Jan-2011].

[20] A. Malakhow, "Scalable Memory Pools: community preview feature. Retrieved from Intel Software Network," *Intel (December 19)*, 2011. [Online]. Available: http://software.intel.com/en-us/blogs/2011/12/19/scalable-memory-pools-community-preview-feature/. [Accessed: 06-Jan-2011].

[21] S. Meyers, *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison-Wesley Longman Publishing Co., 1995.

[22] D. Detlefs, A. Dosser, and B. Zorn, "Memory allocation costs in large C and C++ programs," *Software: Practice and Experience (June)*, vol. 24, no. 6, pp. 527-542, 1994.

APPENDIX

A. *Fixed-Size Pool Manager - C++ Code*

```
class Pool_c
{ // Basic type define
  typedef unsigned int    uint;
  typedef unsigned char   uchar;

  uint   m_numOfBlocks;      // Num of blocks
  uint   m_sizeOfEachBlock; // Size of each block
  uint   m_numFreeBlocks;    // Num of remaining blocks
  uint   m_numInitialized;  // Num of initialized blocks
  uchar* m_memStart;         // Beginning of memory pool
  uchar* m_next;             // Num of next free block
public:

  Pool_c()
  {
    m_numOfBlocks     = 0;
    m_sizeOfEachBlock = 0;
    m_numFreeBlocks   = 0;
    m_numInitialized  = 0;
    m_memStart        = NULL;
    m_next            = 0;
  }
  ~Pool_c() { DestroyPool(); }

  void CreatePool(size_t sizeOfEachBlock,
                  uint numOfBlocks)
  {
    m_numOfBlocks     = numOfBlocks;
    m_sizeOfEachBlock = sizeOfEachBlock;
    m_memStart        = new uchar[ m_sizeOfEachBlock *
                                     m_numOfBlocks ];
    m_numFreeBlocks   = numOfBlocks;
    m_next            = m_memStart;
  }

  void DestroyPool()
  {
    delete[] m_memStart;
    m_memStart = NULL;
  }

  uchar* AddrFromIndex(uint i) const
  {
    return m_memStart + ( i * m_sizeOfEachBlock );
  }

  uint IndexFromAddr(const uchar* p) const
  {
   return (((uint)(p - m_memStart)) / m_sizeOfEachBlock);
  }

  void* Allocate()
  {
    if (m_numInitialized < m_numOfBlocks )
    {
      uint* p = (uint*)AddrFromIndex( m_numInitialized );
      *p = m_numInitialized + 1;
       m_numInitialized++;
    }

    void* ret = NULL;
    if ( m_numFreeBlocks > 0 )
    {
      ret = (void*)m_next;
      --m_numFreeBlocks;
      if (m_numFreeBlocks!=0)
      {
         m_next = AddrFromIndex( *((uint*)m_next) );
      }
      else
      {
         m_next = NULL;
      }
    }
    return ret;
  }

  void DeAllocate(void* p)
  {
     if (m_next != NULL)
     {
        (*(uint*)p) = IndexFromAddr( m_next );
        m_next = (uchar*)p;
     }
     else
     {
        *((uint*)p) = m_numOfBlocks;
        m_next = (uchar*)p;
     }
     ++m_numFreeBlocks;
  }
}; // End pool class
```

Listing 2. C++ Source Code.

B. *System Information*

Simulation tests were performed on a machine with the following specifications: Windows7 64-bit, 16Gb Memory, Intel i7-2600 3.4Ghz CPU. Compiled and tested with Visual Studio.

C. *Speed Comparison Graphs*

Each line represents a fixed allocation size and the time taken to allocate repeatedly.

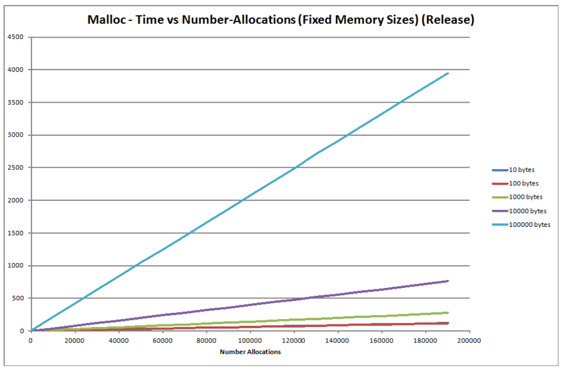
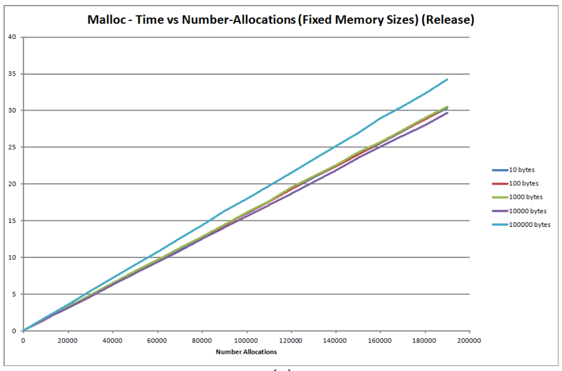
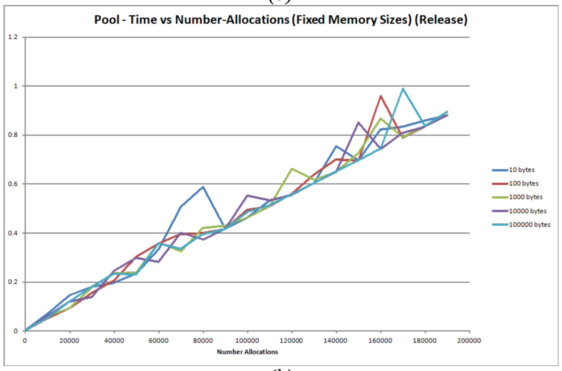


Figure 3. Release build with full optimization running within the debugger (Time in ms); system malloc only.



(a)



(b)

Figure 4. Running outside the debugger – standalone (Time in ms); (a)system malloc and, (b)custom pool.

Utilising an Ant System for a Competitive Real-Life Planning Scenario

Christopher Blöcker[1]
Research and Development
implico GmbH
Hamburg, Germany
Email: christopher.bloecker@implico.com

Sebastian Iwanowski
Department of Computer Science
FH Wedel, University of Applied Sciences
Wedel, Germany
Email: iw@fh-wedel.de

***Abstract*—This paper describes the design of an ant system for a dynamic tour planning scenario for oil and gas delivery. The software has been integrated into an existing planning system and achieved satisfying results in the simulation of real-life scenarios considering spontaneous non-predictable changes of tasks. The notion of such dynamics is more general than in previous approaches. The response time and other complexity measures match the needs of real practice. While other papers already exist describing the functionality and advantages of ant systems and giving some case studies, this paper is the first one referring to an integration into a standard operational SAP system. Thus, this paper shows how to bridge the gap between innovative scientific research and industrial application.**

Keywords-ant system; dynamic tour planning; vehicle routing problem with time windows; greedy strategy; SAP system

I. Motivation

Software for tour planning solutions often suffers from the problem that typical real-life applications continuously violate the original input specifications due to changes of existing tasks, generation of new tasks, malfunction of operational units, and traffic congestion in the underlying route system. All these dynamic events may occur while the software is executing its current task.

Tour planning is an NP complete problem, which makes it infeasible to design an efficient solution satisfying all theoretical needs. Real-life logistics requires a solution for the even more complex vehicle routing problem VRP (cf. [1], [2]) dealing with several vehicles to serve delivery orders meeting pre-defined time windows.

Despite these theoretical obstacles, reasonable heuristics for tour planning already exist achieving remarkable run time results for problem sizes occurring in real problems. Besides classical OR techniques (e.g., cf. [3]), some promising heuristics also use innovative artificial intelligence techniques such as neural networks, genetic algorithms, and ant algorithms (cf. [4], [5], [6], [7]).

However, the benchmarks normally used in the scientific community (cf. [8], [9]) in order to evaluate the different heuristics do not consider problems of the type described above i.e., problems where the input is subject to continuous and unpredictable changes even during execution of the software.

Some of the ant papers cited above (e.g., [6]) do work with dynamic changes explicitly, which is not surprising because ant algorithms are specially suited for that situation as we will also elaborate in the following. But none of them referred to the exact planning tasks we wanted to deal with, which are described in Section II.

The task of this work is an implementation of a tour planning system coping with dynamic changes. The software has to be integrated smoothly into the IDM (Integrated Dispatch Management), which is an implico framework for tour planning of oil and gas delivery linked to an SAP system. Typical scenarios of past applications of IDM serve as evaluation benchmarks.

This paper is organised as follows: Section II presents the problem to be solved in practice. Section III describes the software architecture of the operational system, in which the new solution had to be integrated, and the functionality of the modules existing before. Section IV gives a short description of principles and advantages of ant systems in general. Section V shows how we adapted these general principles for the actual problem. Section VI gives some test results and interpretations. The conclusion in Section VII compares with other approaches.

II. The Actual Planning Task

Our actual planning problem is the scheduling and routing of oil and gas delivery to a set of customers: The customers specify the requested product and a time window, in which they want to receive the delivery. The possible transportation units are heterogeneous trucks, which are initially located at several truck depots. The products are located at several supply depots differing in availability and price. Not all trucks are eligible to transport all kind of products or fit further needs of all customers.

Among the frequent dynamic events that we want to take special care of are the failure of a vehicle, delays in the delivery procedure, incoming of new orders with high priority, and traffic congestion during the tour. Our target is to provide a substitute schedule shortly after a dynamic event occurs and to minimize the additional costs.

[1]This work was done while the author was working on his Bachelor's degree at FH Wedel.

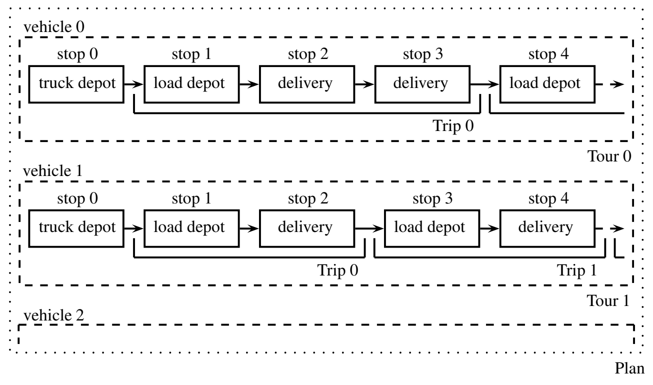


Figure 1. The structure of a delivery plan built from tours, trips and stops.

A schedule in our context consists of a set of several delivery plans each describing the deliveries to be performed within one day in the given planning period. A single delivery plan is composed of a set of several delivery tours, one tour per vehicle. Due to common practice in oil and gas delivery, every tour is defined to begin and end at the corresponding truck's home depot. A tour contains several subsequently ordered trips. Each trip is a sequence of stops. The first stop of a trip is a supply depot followed by several delivery stops. When a product for a subsequent customer is finished, the truck has to reload new supply at a depot starting a new trip. An extract of a sample plan is shown in Fig. 1.

Each customer may reserve time windows, in which he wants to be delivered. Additional attention has to be paid to the reservation of recreation breaks for the drivers prescribed by law. Furthermore, not all different products are eligible to be shipped together on the same trip due to possible chemical reactions or even explosions.

In general, we assume that there is no need to split delivery charges because for each single delivery charge there will be at least one vehicle providing sufficient transport capacity. If this is not the case, we require that the delivery charge is decomposed in appropriate sizes prior to consideration within our planning procedure.

III. Software Architecture of the Operational System

The underlying business framework IDM provides all functionalities a human dispatcher needs. For example, it visualizes current orders to be fulfilled, the current delivery plan on a map and gives easy opportunities for manual integration of new stops and rearranging the current plan. It is based on an SAP system [10], which handles the entire delivery order process. In addition to this base functionality, IDM provides an interface for the integration of various tour optimizers, which may even be operated simultaneously and should ideally make a manual interaction unnecessary. But since the dispatchers are often overcharged with the frequent dynamic changes of the situation in practice, they should at least be supported by an automatic tour optimizer being able to handle specially the dynamic case in a very short response time.

Currently, IDM involves three different optimizers in the context of delivery scheduling (including our ant system), each addressing a different part of the problem.

The first of them, TermiDe, is used to determine whether an incoming request can be served considering the time window and other constraints given by the customer. This is typically done by a phone order several days prior to the actual delivery. If a delivery can be granted, the order is put into a pre-schedule for tour planning, which is the starting point for subsequent optimizations. Since TermiDe is used for telephone sale, it must provide its decision a few seconds after the request. This leads to a feasible but not very good solution.

The second optimizer, IcedG, uses a metaheuristic approach based on tabu search [11], [12] in order to approximate solutions for the static VRP. It is run daily with the purpose to precompute the delivery plan for the following day based on the results of TermiDe. IcedG faces no competitive restrictions regarding the response time, but the quality of the result is required to be very high because it has direct influence on the operational costs of the schedule. Typically, IcedG computes almost the optimal result, but it may need several hours for computation.

The tour optimizer, which is subject of this paper, is called Dyonisys, which is short for **Dy**namic **o**ptimization using a **n**ature-**i**nspired **sys**tem. This optimizer is running during the execution of a delivery plan, i.e., during the whole day. In the morning it gets its input from the schedule, which was previously computed by IcedG. Whenever a notable event occurs, Dyonisys tries to adapt the schedule to the new situation. In fact, Dyonisys can also be configured such that it may further improve the current plan while there is no other work to do.

Most likely Dyonisys will not be able to compute as good results as IcedG would do. The advantage of Dynisys is the following: While IcedG only solves the static VRP and needs several hours to compute an approximate solution, Dyonisys is capable of solving the dynamic VRP and to reduce the response time to a few minutes.

At any time, IDM may ask Dyonisys for the next stop for a particular vehicle and assign a task to it according to the current schedule. Dyonisys then considers the stop the vehicle is currently heading to as granted and estimates the time point when the vehicle will be available for further deliveries when the next delivery has been rolled out.

If the situation happens that a certain delivery cannot be performed at all according to the current schedule due to an unexpected event, this is detected by IDM, which informs Dyonisys. Then, Dyonisys would reply with a substitute schedule, as soon as required. This can be guaranteed,

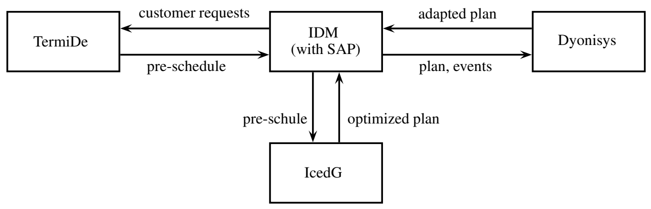


Figure 2. Communication between IDM and its optimizers TermiDe, IcedG and Dyonisys.

because ant systems are anytime algorithms which always can provide a solution. Of course, the quality of the solution depends on the allowed response time.

The communication between the different modules described above is given in Fig. 2.

IV. General Principles of Ant Systems and their Advantages

Ant systems are used to solve optimization problems on graphs. The current optimization problem is defined by a target function depending on edge costs and possibly further information about the graph. Ant systems are specially designed for the scenario that edge costs and the target function vary during operation. They resemble the behavior of natural ants when they seek for food.

Being living animals with a nondeterministic behaviour, ants differ in their strategy from classical optimization methods in the way that there are always some individuals searching a solution on an obviously non-optimal path. An ant colony consists of a large number of individuals, who communicate via pheromones, which are chemical substances dropped on the paths. The intensity of the pheromones biases the behaviour of ants, which leads to a nearly optimal solution for the majority of the individuals. However, the nondeterministic behaviour of the minority enables such a system to react rather quickly to dynamic changes of the environment. This is the main advantage of ant systems.

In the following, we describe the general idea of artificial ants and ant systems.

An (artificial) ant is a software unit, which is continuously generated over time by the ant system. Each ant uses the current data of the graph, considers the current constraints to be solved at the time of its generation, and tries to find a single solution for this problem. The quality of the result influences the modification of pheromones, which are dynamic information chunks placed onto the edges of the graph. The pheromones represent the collected memory of previous ants using the respective edge. Subsequently generated ants are biased by these pheromones for their own construction of a solution.

In general, ant systems use complete graphs for tour planning problems since this will always enable them to complete partial solutions. For our problem, this assumption is reasonable because in practice it is always possible to find a route from one location to any other.

The probability of selecting an edge for tour completion depends on the quality of pheromones put so far as well as on some heuristic value, which is usually derived from the graph's cost function. Usually, this heuristic value is static, but for ant systems even that need not be. The trade-off between the dynamic pheromones and this heuristic value may vary depending on the stage of the process or on the application in general. The continuous generation of ants by the system guarantees that the pheromone value is successively updated to the latest situation in an eager way i.e., prior to a possible request from a user to the ant system. This guarantees a quick response time for any request. However, after the occurrence of a new event, the longer the ant system is running, the better do the pheromones reflect the current situation.

The construction of solutions is carried out in different phases.

First, in the initiation phase, initial pheromones are distributed to all edges in the graph. Normally, all edges get the same pheromone value, but at initialization we could make that also dependent on the cost function [4].

Then, in a loop, construction and coordination phase are executed in turn as long as the ant system is needed. In the following, we denote a construction step followed by a coordination step as one iteration.

In the construction phase, a certain number of ants is generated simultaneously. Each ant has to find a solution. Applied to our VRP, the task of a single ant is to construct an assignment of vehicles to the stops in a certain order such that the tour is feasible for each vehicle, each station is served in the requested time window, and there is always sufficient product supply. At each stop, which is reached by an ant during the construction of a tour, the probability that this ant will use a certain edge leaving this stop is directly proportional to the amount of the pheromone value. Thus, more ants will use the edges baring good pheromone values.

When all ants that were generated in the construction phase have constructed a solution, the construction phase is finished and the coordination phase starts, in which all pheromone values are updated: First, all pheromones are decreased resembling evaporation. This makes the future results more decisive than the past ones. After evaporation all ants increase the pheromones on the edges they actually used for their specific solution. The increase of the pheromones is inversely proportional to the real costs of the associated solution. In total, this makes pheromones of edges belonging to favorable tours increasing and of disfavorable tours decreasing.

Note that the alternating phases of construction and coordination correspond to a discrete simulation of a continuous process which was first described in [4].

If a current solution is requested, the system returns the solution obtained from the current pheromones. This enables an ant system to give a quick answer with a solution corresponding to the current status of the dynamic system and makes it suit well for the problem we have addressed above.

For further information about ant systems, we refer to the standard literature such as [4], [5], [13], [14].

V. How Dyonisys Works in Detail

A request to the ant system is generated every time a vehicle in operation needs the information about the next target. Dyonisys then looks up which delivery is scheduled next, passes this information to IDM and removes it from the set of deliveries that are left to be performed. We have decided to take this approach because it is much more practical to inform a driver about his next target once he needs to know this than to inform the driver continuously about the best schedule found so far.

At the beginning of a day, Dyonisys starts with the schedule produced by IcedG. As long as no dynamic event occurs there is no need in running the ant optimizer because the IcedG schedule achieved near optimality. Once an event occurs, Dyonisys catches up with the current state of the real world situation i.e., it modifies the graph's cost function or adds / removes vehicles or deliveries.

Dyonisys uses the following representation of the scenario: For every delivery, load depot, and truck depot, the underlying graph maintains a separate node. Since IDM already provides a distance matrix between the locations involved, Dyonisys need not perform the actual navigation on street level.

According to the general principle described in the previous section, each ant to be generated will try to find a solution for the total planning task, which is currently in consideration. Each solution is evaluated according to a quality function considering the overall time and the consumption of resources. The solution is also compared with the constraints currently valid. If a constraint is violated, a penalty function is applied decreasing the evaluated quality of the current solution. There are several penalties depending on the different types of constraint violations.

In the initialization phase, Dyonisys creates the ants, which will be used for solution construction and sets the initial pheromones τ_0 to all edges (i, j) from nodes i to j (this is the way it is explained in [4]). For performance reasons, we prefer to reuse the ants rather than to dispose them and create new ones every iteration.

Right after that, the loop of construction and coordination phase is started, and each ant produces a schedule. We chose a greedy strategy for this step: An ant picks one of the vehicles and creates a complete tour before proceeding with the next one until all deliveries are assigned to a tour or there is no capacity left among the vehicles. The benefits of this technique are a good response time and quite satisfactory results.

Note that there always exists the trivial solution where no delivery is carried out. So we will always have an initial solution. However, in nearly all cases we will get a better one, because this trivial solution is associated with very high penalties.

As mentioned before, the ants successively construct a tour starting at the selected vehicle's home depot. At any point, they decide which node of the graph (i.e., which delivery stop respectively, which supply depot) to visit next depending on a value derived from the pheromone level and heuristic value of the incident edges.

If an ant is located at node i, then the probability of visiting node j next is obtained by evaluating term (1), where $\tau_{i,j}$ denotes the pheromone level on the edge (i, j), $\eta_{i,j}$ the heuristic value of the same edge and α, respectively β, are weighting parameters to control the contribution of pheromones and heuristic. $N(i)$ is the set of unvisited neighbours of node i (cf. [4], [5]).

$$p_j = \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum\limits_{j \in N(i)} \tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}} \tag{1}$$

For reasons of efficiency, we only evaluate the numerator of (1) and accumulate the values to obtain the denominator. Instead of normalizing the probabilistic values according to (1) and generating a random value $r \in [0, 1]$, we introduce the new approach to let them unchanged and take a random value $r' \in [0, \sum_j p_j]$. Choosing this implementation we were able to save a lot of runtime without altering the result. Note that if α were set to 0, the ant system would act in a completely deterministic way, and the pheromone values would be ignored.

After that, the coordination phase is started:

First, evaporation takes place. A fraction ρ specifies the amount, by which a pheromone value should evaporate, cf. (2).

$$\tau_{i,j} \leftarrow (1 - \rho)\, \tau_{i,j} \tag{2}$$

Then, each ant increases the pheromone value on the edges it used for constructing its individual solution. According to Dorigo et al. [4], if the cost of a solution of ant a is c_a, then the pheromone increase is $\frac{1}{c_a}$. Dorigo et al. also suggest to apply additional methods such as allowing only the best ants to increase pheromone values at all or to add so-called elite ants, which reinforce the best solutions found so far (cf. [15]).

Since our solution had to be effective also for huge networks which may occur in practice, we had to adapt the pheromone update a little more sophisticated than previous papers suggest:

Since some overall cost values c of our scenarios were rather high, the inverse $\frac{1}{c}$ (which should be the pheromone update value) came close to 0. In these cases, the evaporation rate of pheromone values would outnumber the increase rate of pheromone values, which eventually would lead to a pheromone value nearly 0. Subsequent ants would then use the (static) heuristic function only, but not the dynamic pheromone values, which makes our system less prone for dynamic changes. This is why we use the update value $\frac{c_{best}}{c_a}$ instead of $\frac{1}{c_a}$, where c_{best} denotes the value of the best solution found so far and c_a the costs of the solution computed by and a. Then the update value for ants having found a rather good solution is close to 1 (or even higher if the ant found a better solution than before). This would make the respective edge favourable for subsequent ants. Only when an ant finds a very bad solution, this would still decrease the update value as it did before our modification, but a frequent occurrence of such solutions would indicate that the link really deteriorated. This makes our system reacting on dynamic changes just as desired.

We summarise the algorithmic improvements to the state-of-the-art described in Section IV:

1) The probabilistic values of (1) are not normalized, which saves a lot of time without altering the result.
2) Our evaporation update of the pheromones is done in such a way that pheromone changes are also realised in huge networks.

Considering the operational system, special attention should be taken to handle dynamic events such that the proper execution of the ant system is not too much disturbed by the unexpected input of a new event. For this sake, we make a distinction between events changing the structure of the graph and events leaving the graph as it is.

Since we are only dealing with complete graphs, a cost function change leaves its structure always unmodified. Thus, an information about a traffic congestion would simply result in an update of our cost function and let the ants continue their work. The same argument holds for the opposite case, i.e., when edge costs are reduced.

However, adding or deleting a node would be a change of the graph structure. Such changes need a more sophisticated treatment: The ant system has to be halted, the changes have to be applied, and then the ant system may resume its work.

The following considers the tasks of such structure changing events in more detail:

New customer orders must be added to the set of deliveries and marked as unplanned. Simultaneously, a new node is added to the graph, connected to all other nodes, and pheromones are put onto the new edges. Feasible values for the new pheromones are the initial values for starting graphs or the average of pheromone values computed so far.

A vehicle break down results in the deletion of a node and its adjacent edges. But this requires a scan of all ants and the removal of this vehicle from all partial solutions. We implemented a proper data structure and method such that this is still superior than deleting all ants and starting from the scratch.

In detail, this is achieved by the following:

Note that we only need to consider ants that have already finished constructing a solution or - at least - that have already started constructing a solution. All other ants will be supplied with the new situation before they start and, thus, need not be bothered by the fact that in previous times we had a different situation.

Our data structure allows an easy access to the tour belonging to the removed vehicle, which is possible in constant time because there is a direct correspondance between a vehicle and the tour it is used for.

This is why we wait until all ants having started, before removal was announced, have finished their construction. Then we delete the tour corresponding to the removed vehicle and mark all deliveries from the deleted tour as unplanned, which automatically leads to higher penalties in the evaluation of the corresponding solution. Only then we start the ants for the new situation. This can be expected to happen very fast due to our greedy strategy for tour construction.

VI. Test Results

We tested Dyonisys using typical real world scenarios scanned by IDM in the past. We chose a standard configuration of 100 iterations, an evaporation rate of $\rho = 0.1$, initial pheromones of $\tau_0 = 25$, $\alpha = 1$ and $\beta = 5$ and ran our tests on an Intel Core-i5 M560 with 8GB RAM. In our tests we used different values for the parameters and studied their effect on the runtime of the ant system and the quality of the solutions.

Not surprisingly, we found that the runtime is directly proportional to the amount of ants used for optimization, cf. Table I and Fig. 3.

Interestingly, we did not get a corresponding result for the quality of the solutions. In general, using more ants enlarges the chance of finding a better solution, but the more ants are used, the smaller is the benefit of using an additional ant.

We observed a similar result with respect to the number of iterations: The solutions also get better as more iterations are performed, cf. Table II and Fig. 4. But, if we have already applied a high number of iterations, the benefit of an additional iteration is rather low.

As expected, if the number of iterations is kept constant, Dyonisys is able to find better solutions using more ants and vice versa. Thus, a higher amount of ants used and a higher number of iterations performed have a similar quality enforcing effect.

Summarizing, we conclude that it is generally more practical to perform a higher number of iterations keeping the number of ants not too high. How many iterations and ants should exactly be used depends on the actual scenario.

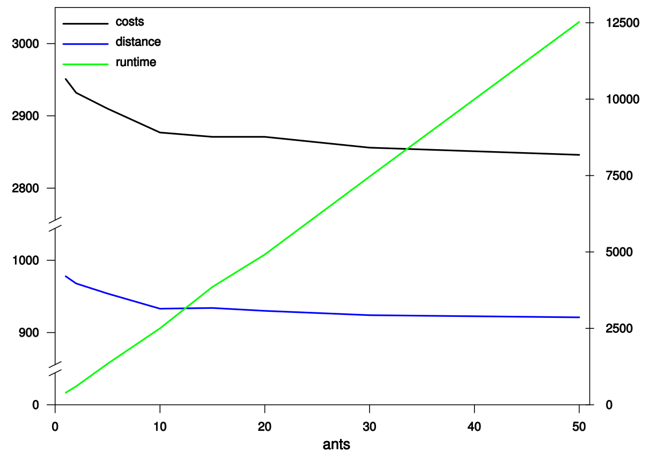


Figure 3. Progression of runtime, total costs and total distance of a schedule depending on the count of ants used for optimization.

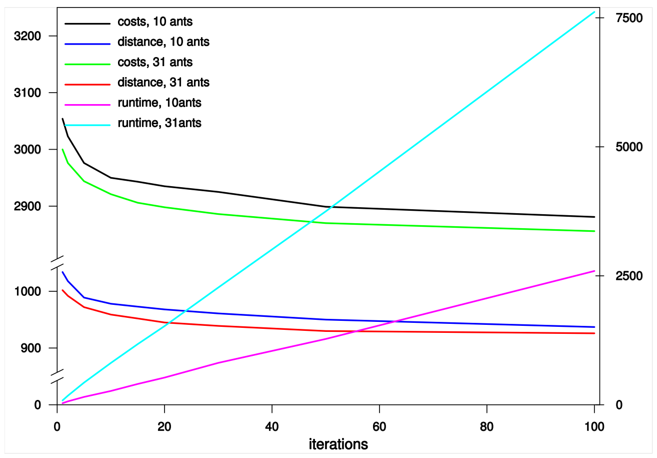


Figure 4. Progression of runtime, total costs and total distance of a schedule depending on the iterations performed by the ant system.

ants	costs	distance	time
1	2.951	978	395
2	2.932	968	604
5	2.910	954	1.343
10	2.877	933	2.501
15	2.871	934	3.856
20	2.871	930	4.918
30	2.856	924	7.468
50	2.846	921	12.529

Table I

	10 ants			31 ants		
iterations	costs	distance	time	costs	distance	time
1	3.054	1.034	31	3.000	1.002	85
2	3.023	1.018	67	2.976	992	177
5	2.976	989	152	2.944	972	429
10	2.950	978	268	2.921	959	810
15	2.943	973	404	2.906	952	1.176
20	2.935	968	527	2.898	945	1.526
30	2.925	961	812	2.886	939	2.274
50	2.899	950	1.276	2.870	930	3.749
100	2.881	937	2.592	2.856	926	7.614

Table II

In most cases, we were able to achieve satisfying results using only 10 ants. In general, there is a trade-off between runtime and quality.

The complete results dependent on the variation of several parameters are elaborated in [16] (in German).

VII. Conclusion

We developed an ant system solving the VRP with time windows for a special application scenario in practice. The ant system was integrated into an operational SAP software environment. The ant heuristics used were simple using a greedy strategy, which resulted in a system with reasonable run time and space consumption. Our intensive testing revealed, which parameters to adjust in order to obtain qualitatively best results in short time. We could thus obtain a setting that fulfils all functional needs.

The improvements to the state-of-the-art are the following:

1) We proposed some algorithmic improvements concerning runtime and applicability for huge networks (cf. Section V). The feasibility is shown in Section VI in examples and in [16] in detail.
2) We took special care for some implementation specific details important to improve the usability in practice. One example is the distinction between different types of events (cf. Section V).
3) Our notion of dynamics is more general than that of other papers. For example, compared to [6], we admit the removal of vehicles and the removal of deliveries from a certain tour.
4) A unique novelty of this paper is the combination with other (standard) techniques for the VRP problem in an integrated software environment (SAP), which makes the novel technique of ant systems ready to be sold in a software product. Previous papers dealing with practical applications showed stand-alone field studies and were not integrated into a software product.

A promising target of improvement would be a replacement of the greedy strategy by a tabu search. Besides this conceptional improvement our focus will be the further product development fulfilling all needs of our customers.

The general message of this work is:

Operational logistics systems baring dynamic behaviour profit from ant technology.

A question that may arise to the reader is: Would this result also hold for other innovative approaches of soft computing?

Our previous experience showed that theoretically successful approaches cannot always be adapted straight forward as this happened to our ant approach:

Before we applied the ant approach we tried to solve our problem with a neural network approach. In particular, we tried to apply self organizing maps (cf. [17]). In a quick implementation we were able to achieve remarkable results on the standard set of benchmarks for the traveling salesman problem (cf. [18]), which were even better than our results with ant systems.

But, we faced two main problems when we attempted to design a neural network approach to solve our vehicle routing problem as stated above.

First, we found no adequate way of readjusting the learning parameters in case of a dynamic event.

Second and even more severe, in our application the triangle inequality does not hold for all triples of nodes in the graph, because we have to obey toll costs in the traffic network. But, the validity of the triangle inequality is an important prerequesite for readjusting the neurons' positions properly.

We do not claim that a neural network approach would fail for our application in general, but at least our attempts revealed that the development of a successful solution would not be that easy as the one presented in this paper using ant systems. Thus, a further value of this work is that it proved the practical applicability of ant systems in a real world setting which is not self understood as indicated with our search for alternatives. This makes the future development of ant systems for logistics applications more attractive even if other approaches may prove superior in closed world experiments.

References

[1] A. Larsen. The Dynamic Vehicle Routing Problem. PhD thesis, University of Denmark, 2000.

[2] P. Toth and D. Vigo. The vehicle routing problem, volume 9. Society for Industrial and Applied Mathematics, 2002.

[3] O. Bräysy and M. Gendreau. *Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms*. *Transportation Science*, 39(1):104–118, 2005.

[4] M. Dorigo and T. Stützle. Ant colony optimization. The MIT Press, 2004.

[5] M. Dorigo, M. Birattari, and T. Stützle. *Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique*. *IEEE Comput. Intell. Mag*, 1:28–39, 2006.

[6] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. Ant Colony System for a Dynamic Vehicle Routing Problem. *J. Comb. Optim*, 10(4):327–343, 2005.

[7] O. Bräysy and M. Gendreau. *Vehicle Routing Problem with Time Windows, Part II: Metaheuristics*. *Transportation Science*, 39(1):119–139, 2005.

[8] M. Gendreau, F. Guertin, J. Potvin, and R. Seguin. *Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries*. *Transportation Research Part C: Emerging Technologies*, 14(3):157–174, June 2006.

[9] A. E. Rizzoli, F. Oliverio, R. Montemanni, and L. M. Gambardella. *Ant Colony Optimisation for vehicle routing problems: from theory to applications*. Technical report, 2004.

[10] SAP online reference. http://www.sap.com/uk/solutions/business-suite/erp/index.epx. [retrieved: 05, 2012].

[11] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[12] P. Toth, and D. Vigo. The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. *INFORMS J. on Computing*, 15(4):333–346, December 2003.

[13] M. Dorigo and L. M. Gambardella. *Ant Colony System: A cooperative learning approach to the traveling salesman problem*. *IEEE Transactions on Evolutionary Computation*, 1997.

[14] É. D. Taillard. *Ant Systems*. *Kluwer*, 2000:131–144, 1999.

[15] M. Dorigo, V. Maniezzo, and A. Colorni. *The Ant System: Optimization by a colony of cooperating agents*. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[16] C. Blöcker. *Entwurf und Implementierung zur dynamischen Optimierung von Liefertouren mit einem Ameisen-System* (in German). Bachelor's thesis, FH Wedel, 2011. http://www.fh-wedel.de/mitarbeiter/iw/eng/r-d/done/bachelor/ [retrieved: 05, 2012].

[17] T. Kohonen, M. R. Schroeder, and T. S. Huang, editor. Self-Organizing Maps. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[18] G. Reinelt. *TSPLIB - A Traveling Salesman Problem Library*. *INFORMS Journal on Computing*, 3(4):376–384, 1991.

Towards Discrete Event Multi Agent Platform Specification

Sébastien Mattei, Paul-Antoine Bisgambiglia, Marielle Delhom, Evelyne Vittori
University of Corsica, CNRS UMR SPE 6134, UMS Stella Mare 3514
Corte, France
{smattei,bisgambiglia,delhom,vittori}@univ-corse.fr

Abstract— **Nowadays, simulation tools have become essential. They allow to study and understand complexes actions that may be impossible to study in situ. In this paper we introduce an approach to use Discrete Event System Specification model as agents and finally create an original platform which allows using DEVS framework and a Multi Agent System platform working together to simulate population dynamics. We first apply this approach on anthill model and then the final approach to a fish model. Ants' basic behaviors are added successfully in DEVS formalism.**

Keywords-DEVS; MAS; Modeling; Simulation.

I. Introduction

For years, we have been working on modeling and complex system simulation [1–4], and on Discrete Event System Specification (DEVS [5]). Our researches are essentially based on DEVS formalism [6]. In 1970's, Professor Zeigler [5] introduced this method that has proved successful. It represents: (1) a complex system from an interconnected collection with more simple subsystems; (2) a separation between modeling and simulation, simulation algorithm are automatically generated according to defined models. This formalism is open, flexible and offers a large extension capacity.

According to recent works [4], [7–11], it has been proved that DEVS formalism might be qualified as a multi-formalism thanks to its opening capacity, to its capacity to encapsulate others modeling formalisms. In one heterogeneous system, it is possible to use modeled subsystems from different formalisms, differentials equations, neuron networks, continuous systems.

These opening and extension capacities are really interesting in our researches, because this formalism has boundaries and doesn't allow a representation of all kind of systems like living systems. In order to get over these boundaries, Multi Agent System (MAS [12], [13]) seems to be an interesting alternative.

MAS's purpose is to create cooperation between entities (agents) that have intelligent behavior, and to coordinate their purposes and their action plans to solve a problem. In our case, MAS utilization is justified because they are adapted to reality, they also allow: (1) agents cooperation; (2) to solve complexes' issues; (3) incomplete expertise integration. An agent is a physical or virtual identity determinated by movements collections (individual objectives, functions of satisfactions or survival), owning its own resources and getting just one partial representation (or none) of its environment. An agent's behavior goes to satisfy its objectives according to its resources, its skills and its functions of perceptions, representations and communications. MAS have got a lot of applications into artificial intelligence. They can reduce complexity of issue's resolution by breaking the sub-collection's knowledge by associating an intelligent independent agent to each of its sub-collections and coordinating activity of these agents [14]. MAS are related to Distributed Artificial Intelligence (DAI).

Our team works on several scientific research and technology development. These two domains include concepts (scientific way) and concepts implementations (technological way). They are used for issue's study linked to artificial or naturals complexes system's behavior like management and modeling evolutive interfaces systems (spread of pollutants) and natural system's modeling (tides, fishes), telecommunication, acquisition's system's conceptions (sensors) and analysis and data treatment (decision's help).

So, our objective is to capitalize the twenty last years gained experiences and skills to propose a brand new and ambitious project. This project's final objective is to propose a brand new platform to:

- Modelize different processes working on fauna and flora's evolution (tide, wind, phytoplankton, zooplankton, larva, algae, fishes, pollutants, etc.);
- Simulate from autonomous and intelligent's agents the interaction and evolution of these processes. We are going to develop and integrate in a multi modelization and simulation based on multi formalism's environment «Discrete Event System Specification» (DEVS a hybrid platform based on multi agent system's properties (MA S [14]);
- Finally, we want to provide tools for decision help to make easier the simulating results operations and resource management.

To meet our goals we are going to proceed step by step. In this paper, we propose to associate DEVS formalism and MAS. From advantages of these two paradigms, we want to define an extension of DEVS's formalism, faithful to standard, that make possible to modelize agents and way of communication and environment interactions as a DEVS's system. This transformation, from agents to model, emerge a lot of issues that we propose to study and solve by our self.

In the first part, we introduce DEVS formalism and MAS principles. Then, in a second part, we detail these two modeling and simulation's paradigms advantages. In the third part, we propose our approach formalization. Then, the fourth part is dedicated to a MAS study case presentation before proposing its DEVS's model conversion. At last, the final part concerns the conclusion and a presentation of our future works.

II. Overview

Today, simulation's tools have become essential. They allow to study and understand complexes actions that may are impossible to study in situ. In this part, we introduce our works based on our researches and in particular two modeling and simulation's methods.

A. *DEVS presentation*

DEVS formalism [5], [6] is based on the definition of two types of components: atomic models and coupled models.

Atomic model (Fig. 1) provides an autonomous description of the system behavior, defined by states, input/output functions and transition functions. The coupled model is a composition of atomic models and/or coupled models. It is modular and presents a hierarchical structure which enables the creation of complex models from basic models.

1) *DEVS models*

Atomic DEVS model:

$$AM: < X; Y; S; t_a; \delta_{ext}; \lambda; \delta_{int} > \qquad (1)$$

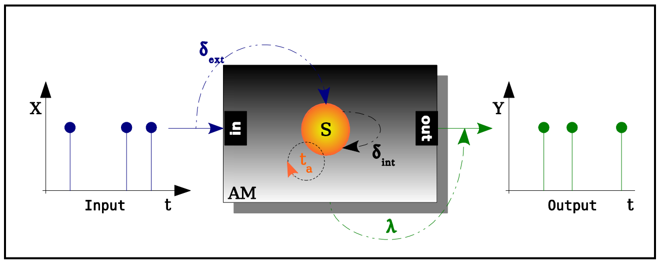


Figure 1. Atomic model.

where:
- X: is the set of input events, is characterized by a couple (port, time, value), where the port means the input on which the event occurs, the time is the date of occurrence of the event, it is blank for internal events, and the value symbolizes the data from the event;
- Y: is the set of output events;
- S: is the set of partial or sequential states, which includes the state variables;
- t_a: $S \rightarrow T_\infty$: is the time advance function which is used to determine the lifespan of a state;
- δ_{ext}: $QxX \rightarrow S$: is the external transition function which defines how an input event X changes a state of the system, where $Q = \{(s, t_e) \mid s \in S, t_e \in (T \cap [0, t_a(s)]\}$ is the set of total states, and t_e is the elapsed time since the last event, T is the total time of the simulation;
- λ: $S \rightarrow Y\iota$: is the output function where $Y\iota = Y \cup \{\iota\}$ and $\iota \notin Y$ is a silent event or an unobserved event. This function defines how a state of the system generates an output event, when the elapsed time reaches to the lifetime of the state;
- δ_{int}: $S \rightarrow S$: is the internal transition function which defines how a state of the system changes internally, when the elapsed time reaches to the lifetime of the state.

Every state S is associated with a lifetime t_a, which is defined by the time advance function. When a model receives an input event X, the external transition function δ_{ext} is triggered. This function uses the input event, the current state and the time elapsed since the last event in order to determine what the next model state is. If no events occur before the time specified by the time advance function for that state, the model activates the output function λ (providing outputs Y), and changes to a new state determined by the internal transition function δ_{int}.

Coupled model: coupled model is a composition of atomic models and/or coupled models. It is modular and presents a hierarchical structure which enables the creation of complex models from basic models. It is described in the form of:

$$CM :< XM, YM, CM, EIC, EOC, IC, L > \qquad (2)$$

With:
- XM: all the input ports;
- YM: all the output ports;
- CM: the list of models forming the CM coupled model;
- EIC: all the input links connecting the coupled model to its components;
- EOC: all the output links connecting the components to the coupled model;
- IC: all the internal links connecting the components between themselves;
- L: the list of the priorities between components.

With the DEVS formalism, each model is independent and can be considered as its own entity or as a model of a larger system. DEVS formalism is closed under coupling, that is to say that for each atomic or coupled DEVS model it is possible to build an equivalent DEVS atomic model.

The DEVS models are executed by abstract simulators [15–17] that are independent from the models themselves. Consequently, separated concerns between models and implementations of simulation can be achieved and enhance the verification of each layer independently. DEVS is a popular method to simulate a variety of systems. However, since its introduction by B.P. Zeigler, significant efforts were taken to adapt this formalism to different fields and situations. The many proposed extensions proved its ability to extend and openness.

2) *DEVSIMPY framework*

DEVSimPy framework allows a simple graphical interface to create and use DEVS models. It is a WxPython based environment for the simulation of complex systems.

Its development is supported by the CNRS (National Center for Scientific Research) and the SPE research laboratory team.

The main goal of this framework is to facilitate the modeling of DEVS systems using the GUI library and the drag and drop approach. The interface is designed to help the implementation of DEVS model in form of blocks. The modeling approach of DEVSimPy is based on UML Software, and there is a separating between the GUI part and the implementation part of DEVS formalism.

With DEVSimPy we can: (1) describe a DEVS model and save or export it into a library; (2) edit the code of DEVS model to modify behaviors also during the simulation; (3) import existing library of models which allows the specific domain modeling (Power Systems, Fuzzy, Continuous ...); (4) automatically simulate the system and perform its analysis during the simulation.

3) *DEVS Advantages and drawbacks*

Discrete event simulation has a quickly execution because of its way to treat event, avoiding continuous treatment. Moreover, coupling and separation between modeling and simulation on DEVS formalism allow reusing existing models in new models. DEVS is a powerful formalism allowing reusing models through library already developed and also interconnecting of these models to compose heterogeneous models based on a different formalism. In our team, it used to simulate continuous systems [2], and differentials equations, and fuzzy system [18], and sensor network and neural network.

As such DEVS does not allow simulating all kind of systems. For example, it is not quite complete to study systems describing behavior of living species, and their interaction with environment. As is a formalism associating other approaches, we would like to add new functionalities to coupling it with MAS.

After a description of the DEVS formalism and his working, we'll introduce Multi Agent System.

B. *MAS presentation*

MAS are more suited to living organism's modeling where communication between system's members is complex.

The multi agents' paradigm is issued from the distributed artificial intelligence in the early 80's [13], [14]. This bottom up approach is used to build individual based model dedicated to the study heterogeneous systems and solve problems with complex interactions. Multi-agent systems consist of agents and their environment.

According to Michael Wooldridge we consider that "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [13].

Agents can either be physical (human being, robot, etc.) or virtual. The global behavior of MAS emerges from the sum of individual actions of agents, from the interactions between agents and between agents and their environment. Many Multi-agent systems platforms and frameworks are created by researchers and developers. They implement common standards useful to save developers time and also aid in the standardization of MAS development. Multi-agent systems are applied in the real world to computer games, environment, E-commerce defense, transportation, logistics, GIS.

1) *MAS organisation*

In MAS, environment is created in first. Then, agents are positioned on the environment with random position or known position. There are three kinds of agents:

- Reactive agent;
- Cognitive agent;
- Hybrid agent.

Reactive agent has no representation of its environment or others agents, it only reacts to environment stimuli. It has a simple behavior.

Cognitive agents are smart agents, they detect environment and others agents. They have skills, and they are able to plan an action with his skills and his thoughts. Hybrid agents are the most complex kind of agent: they are the middle way between cognitive agent and reactive agent. They may have simple behavior in reaction of stimuli or complex behavior like a cognitive agent. If we need both behaviors, we can use hybrid agent to describe simple behavior to manage memory (use less memory and it is less complex to code) and complex behavior when we need it. Agent may communicate with others and with environment thanks to three communications methods:

- Share memory (blackboard);
- Communication by messages;
- Message by environment.

Share memory is like a database process. Each agent fills a general knowledge base and takes information in. At any time an agent can ask blackboard for information. Communication by messages is like a conversation. Agents can send a message to one agent or various agents, and they can ask information to the others agents. The final method consists to give the whole responsibility to the environment. Only it can send information to agents.

2) *MAS Advantage and drawback*

Various kinds of agents exist and we explain their perspective's advantage in living organism's modeling.

TABLE I. *MAS AGENT AND COMMUNICATION SUMMARY*

Kind of agents	**Way of communication**	**Architecture**
Reactive	Shared memory (black board)	Subsumption architecture
Cognitive	By message	BDI architecture
Hybrid	By environment	Hybrid architecture

We have chosen reactive agent because its behavior is simpler to modelize, and it doesn't have an environment perception and it reacts only to exterior stimuli that we can modelize with an input message. Thereafter, we plan to use others kind of agents to make behavior and simulation more specific and reliable.

The MAS' major drawback is timing constraint. Using continuous simulation is longer and can cause issues. For example, it is difficult to apply an action to an agent at a given time. While DEVS make it possible. Moreover, MAS does not allow interconnection between heterogeneous models. For example, it is not possible to associate an agent model with a model describing flow of a river.

C. *Existing approaches*

In this part, we introduce two existing approaches and explain our choices about our method.

The first approach we have studied is the platform GALATEA [19].

GALATEA is offered as a family of languages to model MAS to be simulated in a DEVS, multi-agent platform. GALATEA is the product of two lines of research: simulation languages based on Zeigler's theory of simulation and logic-based agents. There is in GALATEA a proposal to integrate, in the same simulation platform, conceptual and concrete tools for multi-agent, distributed, interactive, continuous and discrete event simulation. It is also GALATEA a direct descendent of GLIDER, a DEVS-based simulation language which incorporated tools for continuous modeling as well. In GALATEA, GLIDER is combined with a family of logic programming languages specifically designed to model agents.

This platform would allow modelize different formalisms, in different languages in a same interface. But this implementation seems tedious and difficult.

The second approach is the "Specification of Dynamic Structure Discrete Event Multiagent [20]" article. It matches a lot with our works. An agent is represented by an atomic model and stimuli by exterior messages. But using of VLE plateform [21], [22] to simulate and using of CELL-DEVS formalism [7], [23] to describe environment can make the simulation slower to generate. Indeed, each cell is represented by an atomic model and for a big environment number of atomic model is very high. Because of its number, simulation use lot of resources and memory, especially in our case with a big agent's concentration and a large environment (all of both are atomics models if we'll use CELL-DEVS.

III. OUR APPROACH

After analyzing DEVS and MAS' good points and weaknesses, it is important to remember the final objective of this project: to realize a M&S platform to study population dynamics.

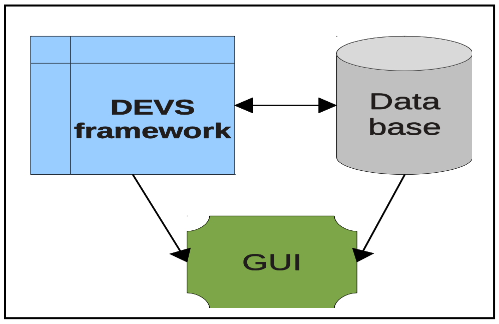


Figure 2. Simplified approach.

Fig. 2 points a global vision of our DEVS based architecture. Agents are DEVS models interacting with a database. Simulation results are displayed by an external tools (viewing tool). It's the first step of our project: describe MAS from DEVS model.

Our goal is to propose a coupling between these two modeling approaches in order to keep their advantages: flexibility and opening (in Information technology) for DEVS formalism and good living organism's representation and their interactions for MAS.

To modelize living organisms and their interactions, the tool must allow:

- To create and destroy agent during simulation;
- To modify variable during simulation;
- To have a graphical and dynamic representation of models' evolution;
- To follow evolution thanks to curves;
- To save simulation's results in database or file;

In this case, we need:

- Data on species studied (by simulation or by levy and field study);
- A modeling environment;
- Decision aid tools.

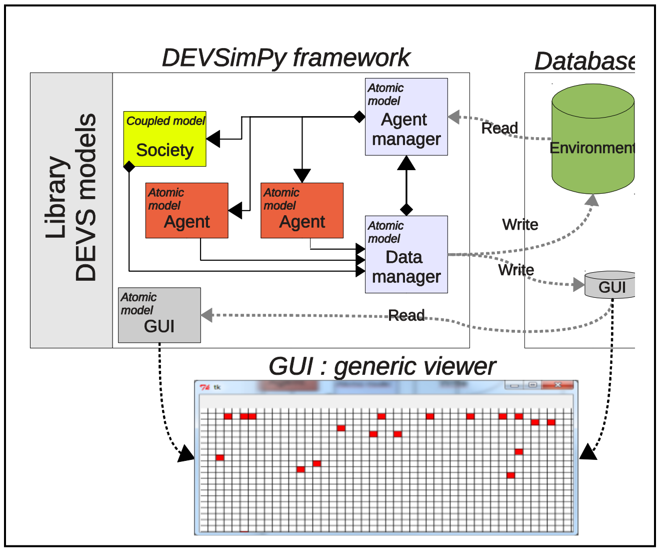


Figure 3. AgentDEVS platform.

Fig. 3 represents our platform architecture. It is composed of a DEVS model library including in the DEVSIMPY framework and two databases. The first database contains all data and the second is simpler and contains only ants' location used to make the graphical interface.

DEVSIMPY framework is composed by various atomic and coupled models. We detail these models:

- One atomic model "Agent manager" is charging to read and transmit information from database to agents (send answer) and it is able to create and destroy agents;
- One atomic model "Data manager" to manage agents output (it receives their messages) and to update the two databases (a full database and another allowing

to display information). It can also transmit directly a message to the Agent manager;

- N atomic agent models with their own behavior and N coupled society models to gather agents with same behavior;
- An atomic model to allow graphical display.

Data side contains:

- A database containing all information;
- A small database containing only information to display.

To represent an agent in its environment with DEVS formalism, we use atomic models. We have one atomic model for each agent and one atomic model for the environment. Environment is also a supervisor and there are coupled models to represent society and agents' group (Anthill for ant or shoal). Another approach was defined in [19] with a hybrid simulation platform called GALATEA. This platform seems to slow in simulation time and unsuited to utilization that we wish have.

The chosen way of communication is shared memory (Blackboard). After studying and thoughts it seems that it is the most adapted way of communication to couple with DEVS. DEVS being hierarchical, there are a lot of messages outstanding. So, using an associated data file (here blackboard), or a database, allows to limit the number of messages (towards high level model: environment), and avoid concurrent access issues to data (files are generated by high level model and information are centralized). Using communication by messages, it would have been very difficult to manage important flow of messages.

Other issues to consider are:

- File management;
- Messages' format;
- Dynamic agent destruction and creation;
- Graphical interface.

The issue of file management is in first on the file format used. There are lots of data to represent and we suggest using a XML file or Netcdf. This kind of file can represent data easily and clearly. We have also thought on a unique file reading issue to graphical display and we propose solution: to have two files. One has only agent's location and identification and is use only to graphical display: it's a CSV or excel file to represent a grid and the environment. And the other is a XML or Netcdf file to represent all data. The other advantage of file management with one model is data centralization and so we don't have concurrent access issues.

In MAS, message's format is defined by a standard: FIPA-ACL. The minimum message is: type of send messages (syntaxes), message sender, message receiver, message content. But often the minimum message isn't enough to communicate. We may need to indicate others information like used language in message content, used protocol, message's ontology, reference to an earlier message and reference to a conversation. Then message must be transmitted by a Message Transport Service (MTS) by communication channel (Agent Communication Channel: ACC).

In the DEVS formalism, message is composed as following: port, time, value. When compared these two messages, we could say that "port" in DEVS could be "sender" and "receiver" in MAS. "Value" could be "content message". We did not define remaining parameters yet. To avoid agents' creation and destruction issues we suppose that we can use a DEVS' extension. This extension allows create and destroy agents dynamically with a manager [8–10], [16]. The manager is our environment model with a supervisor role.

The final goal being to represent MAS in DEVS model, it is essential to use a graphical interface to display data. We plan to implement a coupling with Google API (Google map to display data on a map world) and use a viewer to display data in a board.

IV. CASE OF STUDY

In order to validate our approach, we chose to work on ant modeling and simulation.

One of the most used models in Multi Agent System (MAS) is the example of anthill. We will try to transcribe this ant model with DEVS formalism. In Multi agent system, an ant is an agent and works alone or with others agents to modify the environment (Fig. 4). Environment represents a spot where agents can interact, is a kind of grid. In a MAS, they are various kind of agents, we used reactive agent. It reacts to an exterior stimulus and does an action when it gets this stimulus. With DEVS, we can represent this with a received message on the entry port of a model. Every reactive agent is an atomic model.

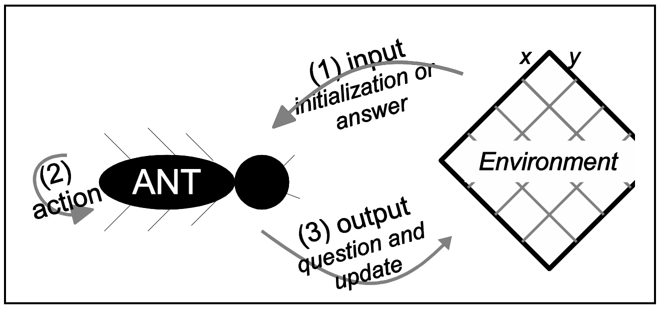


Figure 4. Agent communication with environment.

Fig. 4 points an external stimulus or initialization from environment (1). Then ant makes an action or not (2) and sends a message to the environment to ask questions and update information (3).

A. Ants' behavior description

Global behavior of an ant is made with various independents' behaviors or with behaviors Linked. Behaviors we have retained are:

- To look for food;
- To collect food;
- To drop pheromones;
- To back to the nest (brining food);

- To drop food;
- Possibility to follow pheromones.

An ant check out of the nest and look for food, it makes a random motion around the nest. When it founds food, it eats and goes back to the nest to drop it. During return, it drops pheromones on the way which are chemicals agents able to show the way to follow to find food. After have dropped the food, ant choose to follow pheromone or looking for others sources food.

B. MAS' models

In our model, ant has only one parameter, its energy. Environment have location of food source, nest and others variables like the number of food in the nest and the number of pheromone on a cell. At model initialization, the environment creates many food sources and the nest. Then agents are created with a default position: nest location. First, agents move randomly around the nest at each time step. Ant sends message to the environment to tell him their locations. Environment tests if ants' locations are equal to others object's location like food and sends him the result.

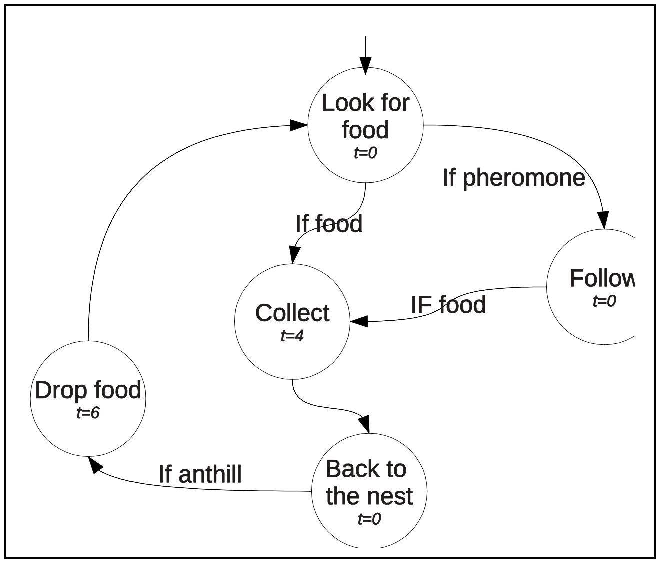


Figure 5. Ants DEVs behavior.

Ant receives message and depending on the result (Fig. 4 and Fig. 5); it keeps its behavior (looking for food) or change (eating food). When it finds food, it sends a message to the environment to inform about the food eaten. Environment knows how many foods remains on the cell. Once the food eaten ant changes its behavior and passes in «back to the nest» step. It drops pheromones (it sends a message to the environment for each cell through and environment adds pheromones on that cell). Then ant drop food in the nest and may choose to follow pheromones if the scent is strong enough (if many ants drop pheromone on a same cell, that cell smells more and catches more ants) or move randomly to find others food sources.

C. DEVS models

The Research team intends to make a link between agent behavior and a DEVS model.

Fig. 5 represents ants DEVS behavior. Message is like following: location x, location y, energy, food, and pheromone. A message involves automatically a data update and an answer.

1) Ant atomic model

The sets of states about our system are S: looking for food collects food, back to the nest (bringing food), drop food and follow pheromones. To start the system's state is 'Look for food'. According to this state and input messages it is changing its state variables (position x, y) and its state and sends message to update the database or query the environment.

The internal function δ_{int} is the way to allow the ant to change its behavior and so change state.

The externals functions δ_{ext} are messages from environment (here, if they are foods or pheromones).

The output function λ is message send by ant to environment (here, its location and number of food eat).

The time advance t_a function is evolution of time like «tick» in Multi Agent System.

Input: X = location of neighbors object and object type (MSG.x,MSG,y,MSG.type)
Output: Y = x, y, id, energy, pheromone, food
Time: Sigma = 0
State: Looking for food, collect food, back to the nest (bringing food), drop food and follow pheromones.
State variable: x, y, food, energy, id

Initialization function:

```
S = LookForFood()
X=Y=Food = 0
Energy = 100
Id = UNIQUE
Sigma = 0
```

Output function: λ

```
MSG.id=ID
MSG.x =x
MSG.y=y
MSG.energy=energy
MSG.pheromone=False
MSG.food=0
Send(MSG)
```

Time advance function: t_a

```
return Sigma
```

Behavior function: δ_{ext}(MSG) :

```
if MSG.type== food et S!= bringingFood:
        S = EatFood // we change state to EatFood
        energy+=10 // it collect food so it gain energy
        X=MSG.x // ant go on the cell with food
        Y=MSG.y // ant go on the cell with food
        food+=20 // ant increases number of food
        Sigma=4 // time to do action
if MSG.type==pheromone et s!=bringingFood:
        S =follow // ant change state to follow
        X=MSG.x // ant go on the cell with pheromone
        Y=MSG.y // ant go on the cell with pheromone
        Sigma =0 // time to 0 to continue motion
if MSG.type==anthill et s==bringingFood:
        S=drop // we change state to Drop
        Increase number of food in the anthill
(MSG.food=food)
        food=0 // food is already drop
        X=MSG.x // ant go on the anthill
        Y=MSG.y // ant go on the anthill
        Sigma=6  // time to drop food
```

Behavior function: $\delta_{int}()$:

```
if s==drop
        S=LookForFood
if s==eatFood
        S= BringingFood
if s==LoofForFood
        X=random
        Y=random
if s== BringingFood
        X=random // towards anthill
        Y=random // towards anthill
drop pheromone (MSG.pheromone=True)
if s==follow
        X=follow pheromone
        Y=follow pheromone
Sigma=0
```

2) *Environment atomic models*

Environment atomic models manage pheromones, food source.

Manager's agent model receives ant's location, if ants drop pheromone (with a Boolean) and if food was eaten. In output it sends to the ants if they are on food source and pheromones location.

Manager's data model updates data.

D. Simulation and results

Our models allow simulate an ants' basic behavior. Its aim is the validation of our theoretical approach. This example must be improved to be realistic.

V. CONCLUSION ANS PERSPECTIVES

In this paper, we proposed an original approach to associate DEVS formalism and MAS. Our approach seems similar to VLE works but these are limited to an environment representation made with cellular grid. We do not want to be limited by this mode of representation.

We propose a new method based on DEVS. We chose an easy study case, ants' evolution, to focus on the interests of our solution. Ant society modeling is a famous example in MAS. We have chosen this example to test our approach and to propose a tutorial application even if it's far to our final application.

Various issues happened: an important increase in messages that were sent between agents and environment (slow down the simulation); the need to dynamically create or destroy agents (dynamic DEVS [8]); the database management from atomic model and the display tool development.

Born of recent work [24], [25] our interest on MAS has become a real need. Moreover it perfectly integrates to the development of our platform (DEVSIMPY). Indeed, in Mediterranean, water availability and quality, as well as and biodiversity evolution represents a real issue. This is why development activities and land use planning must consider the water management and water systems. To answer to these issues, the STELLA MARE project (« Sustainable TEchnologies for LittoraL Aquaculture and Marine Research » was developed at the University of Corsica. Its objectives are large and it must be a scientific research center in innovation and appreciation of Mediterranean resources.

We wish to propose a generic software environment based on discrete event simulation principals (DEVS) and MAS described in Fig. 6. This environment must allow simulating fish resources evolution.

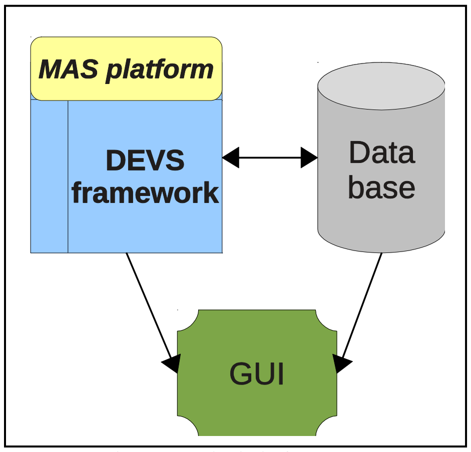


Figure 6. Final platform.

Fig. 6 points our final architecture. At time, we suggested proposed a method to describe MAS from DEVS model. The next step is completing DEVS framework with a MAS platform. The objectives of this platform are to make easier agent creation without describing atomic models behaviors.

Our research perspectives are various with the DEVS formalism and we plan to work on simulation algorithm haste and on cognitive agents' models and on agents and environment's interaction and communication. We need also visualization tools development (viewer).

Our final objective is to define agent models autonomous and intelligent and able to interact together. So, they will be influenced by their environment based on current models.

REFERENCES

[1] J.-B. Filippi, F. Bernardi, and M. Delhom, « The JDEVS environmental modeling and simulation environment », *IEMSS, Integrated Assessment and Decision Support, Lugano Suisse*, pp. 283–288, 2002.

[2] L. Capocchi, F. Bernardi, D. Federici, and P. Bisgambiglia, *Transformation of VHDL Descriptions into DEVS Models for Fault Modeling and Simulation*. 2003.

[3] P.-A. Bisgambiglia, E. de Gentili, P. A. Bisgambiglia, and J. F. Santucci, « Fuzzy Simulation for Discrete Events Systems », in *Proceedings of the 2008 IEEE World Congress on Computational Intelligence (WCCI 2008) - IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 688–694, 2008.

[4] S. Garredu, E. Vittori, J.-F. Santucci, and D. Urbani, « A methodology to specify DEVS domain specific profiles and create profile-based models », pp. 353–359, 2011.

[5] B. P. Zeigler, *Theory of Modeling and Simulation*. Academic Press, 1976.

[6] B. P. Zeigler, *Multifaceted modelling and discrete event simulation*. Academic Press, 1984.

[7] J. Ameghino, A. Troccoli, and G. Wainer, « Models of complex physical systems using Cell-DEVS », pp. 266–273.

[8] F. Barros, « Dynamic structure discrete event system specification: a new formalism for dynamic structure modelling and simulation », in *Proceedings of Winter Simulation Conference 1995*, 1995.

[9] A. M. Uhrmacher and B. Schattenberg, « Agents in Discrete Event Simulation », in *Proceedings of ESS98*, 1998.

[10] A. Uhrmarcher, « Dynamic Structures in Modeling and Simulation: A Reflective Approach », *ACM Transactions on Modeling and Computer Simulation vol. 11 2001*, pp. 206–232, 2001.

[11] P. Fishwich and B. P. Zeigler, « A multi-model methodology for qualitative model engineering », *ACM transaction on Modeling and Simulation, vol. 2*, n^{o}. 1, pp. 52–81, 1992.

[12] G. Weiss, *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[13] M. Wooldridge, *An Introduction to MultiAgent Systems*, Wiley and Sons. Chichester, West Sussex, Angleterre: Wiley and Sons, 2002.

[14] J. Ferber, *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman. Addison Wesley Longman, 1999.

[15] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation, Second Edition*. 2000.

[16] F. Barros, « Abstract simulators for the dsde formalism », in *Proceedings of WSC 1998*, pp. 407–412, 1998.

[17] A. C. Chow and B. P. Zeigler, « Abstract Simulator for the Parallel DEVS Formalism », in *Proceedings of AIS94*, 1994.

[18] P.-A. Bisgambiglia, P. A. Bisgambiglia, and J.-S. Gualtieri, « Cognitive simulation-based on knowledge evolution in fuzzy discrete event systems », pp. 895–901, 2011.

[19] J. Davila and M. Uzcategui, « GALATEA: A multi-agent, simulation platform », in *In International Conference on Modeling, Simulation and Neural Networks MSNN'2000*, Mérida, Venezuela, 2000.

[20] R. Duboz, D. Versmisse, G. Quesnel, A. Muzzy, and E. Ramat, « Specification of Dynamic Structure Discret event Multiagent Systems », in *Agent-Directed Simulation (ADS 2006)*, Huntsville, AL, USA,, 2005.

[21] E. Ramat and P. Preux, « "Virtual laboratory environment" (VLE): a software environment oriented agent and object for modeling and simulation of complex systems », *Simulation Modelling Practice and Theory*, vol. 11, n^{o}. 1, pp. 45–55, March 2003.

[22] G. Quesnel, R. Duboz, and É. Ramat, « The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems », *Simulation Modelling Practice and Theory*, vol. 17, n^{o}. 4, pp. 641–653, April. 2009.

[23] L. Ntaimo and B. P. Zeigler, « Expressing a Forest Cell Model in Parallel DEVS and Timed Cell-DEVS Formalisms », 2002.

[24] D. Urbani and M. Delhom, « Water Management Using a New Hybrid Multi-Agents System - Geographic Information System Decision Support System Framework », pp. 314–319, 2006.

[25] D. Urbani and M. Delhom, « Analyzing knowledge exchanges in hybrid MAS GIS decision support systems, toward a new DSS architecture », in *Proceedings of the 2nd KES International conference on Agent and multi-agent systems: technologies and applications*, Berlin, Heidelberg, pp. 323–332, 2008.

Multiplicative Complexity and Solving Generalized Brent Equations With SAT Solvers

Nicolas T. Courtois
University College London,
Gower Street, London, UK
Email: n.courtois@ucl.ac.uk

Daniel Hulme
University College London,
Gower Street, London, UK
Email: d.hulme@cs.ucl.ac.uk

Theodosis Mourouzis
University College London,
Gower Street, London, UK
Email: theodosis.mourouzis.09@ucl.ac.uk

***Abstract*—In this paper we look at the general problem of Multiplicative Complexity (MC) as an essential tool for optimizing potentially arbitrary algebraic computations over fields and rings in the general non-commutative setting. Our goal is to find optimizations in a fully automated way via algebraic formal coding and conversion to a SAT problem [1].**

We focus on the basic problems of minimizing the number of multiplications in Matrix Multiplication, complex number multiplication and also quaternion multiplication. Minimizing the number of multiplications in the Matrix Multiplication problem alone (and this for problems of fixed size some of which we were able to optimize [4]) is known to be able to lead to immediate improvements in countless other algorithms on formal languages, graphs, arbitrary finite groups, various real/complex/algebraic rings and fields of practical importance. Thus we may hope to translate our efforts to improve many high-profile applications in computer graphics, signal processing, cryptography, computational physics and chemistry, weather prediction, financial computing, Google page ranking, etc.

The classical tool to solve the Matrix Multiplication problem are the Brent Equations [3]. We have developed a methodology for solving these equations over small fields such as $\mathrm{GF}(2)$ with a conversion to a SAT problem and progressive lifting to larger fields and rings. We generalize the Brent Equations [3] and extend our method to similar algebraic optimizations and to tri-linear problems.

We have been able to obtain new results to decrease the MC of several well known operations in algebra, which to the best of our knowledge are new. For example we have obtained a new general 3×3 matrix multiplication method with 23 multiplications [4]. We also present new formulas for complex number multiplications and quaternion multiplications. Additionally, using our methodology we are able to produce highly optimized implementations of small circuits. We obtained exact lower bounds with respect to MC of two very well known block ciphers, such as PRESENT and GOST, known for their exceptionally low implementation cost. Our method is efficient for any sufficiently small circuit [5].

***Index Terms*—Linear Algebra, Fast Matrix Multiplication, Complex Numbers, quaternions, Strassen's algorithm, Multiplicative Complexity**

I. Introduction

The optimization of certain arbitrary algebraic computations over fields and rings in the general non-commutative setting is considered as one of the most important topics in theoretical computer science and mathematics. In this paper we study how the Multiplicative Complexity (MC) of certain arbitrary algebraic computations such as the Matrix Multiplication (MM), multiplication of complex numbers, multiplication of quaternions and of a general Boolean circuit can be reduced over small fields such as $\mathrm{GF}(2)$, the field of two elements, and then be progressively lifted to larger rings.

MC is the minimum number of AND gates that are needed if we allow an unlimited number of NOT and XOR gates. Informally, we are interested in reducing the number of multiplications involved in an arbitrary algebraic computation allowing unlimited number of additions.

Our method consists of three basic steps. In the first step we formally encode the problem by writing a system of equations which describe the problem and then we consider the problem over the finite field of two elements $\mathrm{GF}(2)$. In case of the MM problem and the complex or the quaternion multiplication problem we use the Brent Equations [3] in the encoding step while for circuit minimization we encode the problem formally as a straight-line representation problem, described by a quantified set of multivariate relations [5]. Then we proceed by converting the reduced modulo 2 problem to a SAT problem using the Courtois-Bard-Jefferson method [2] and then we progressively lift the solution to larger fields and rings using different heuristic techniques and other constraint satisfaction algorithms.

A. *Motivation for Low MC*

Matrix Multiplication:

One of the most famous problems in computer algebra is the problem of MM of square and non-square matrices, where the aim is to reduce the number of 2-input multiplications needed in order to compute the product of two matrices. A speed-up in MM will automatically result in a speed improvement of many other algorithms and techniques such as:

- Gauss Elimination algorithm for solving a system of linear polynomial equations
- Algorithms for solving of non-linear polynomial equations
- Recognizing if a word of length n belongs to a context-free language
- Transitive closure of a graph or a relation on a finite set
- Cryptanalysis

Circuit Complexity:

We refer to some reasons why circuits of low MC are very important especially for industrial reasons and for cryptography. For more analytic explanations, see [5].

- Lower the hardware implementation cost of a cipher in silicon
- Develop certain so called Bitslice parallel-SIMD software implementations of block ciphers such as in [16]
- In symbolic computing and numerical algebra, this kind of optimization can be applied recursively to produce asymptotically fast algorithms to solve very famous and important practical problems such as Gaussian reduction and MM
- Prevent Side Channel Attacks (SCA) on smart cards such as Differential Power Analysis (DPA) [15].

II. Methodology

We have fully automated the process as follows:

1) Form the Brent Equations (or write a quantified set of multivariate relations that describes the problem)
2) Consider only solutions in 0,1=integers modulo 2
3) Convert to SAT with Courtois-Bard-Jefferson method [2]
4) Lift the solution from $\mathrm{GF}(2)$ to the general bigger fields by another constraint satisfaction algorithm

A. Brent Equations

We use Brent Equations as a sort of "formal algebraic" method for encoding problems that optimize certain arbitrary algebraic computations. Our main idea is to encode such problems into a "language" which can be converted to a SAT problem and then we attempt to solve this hard problem using our portfolio of 500 SAT solvers.

Suppose we want to multiply a $M \times N$ matrix A by a $N \times P$ matrix B using T 2-input multiplications.

We solve the above problem by solving the following system of $(MNP)^2$ equations in $T(MN + NP + MP)$ unknowns, see [3]:

$$\{\forall i \forall j \forall k \forall L \forall m \forall n, \textstyle\sum_{p=1}^{T} \alpha_{ijp}\beta_{kLp}\gamma_{mnp} = \delta_{ni}\delta_{jk}\delta_{Lm}\}(1)$$

A solution to this set of equations implies that the coefficient entries c_{ij} of the product matrix $C = AB$ can be written as $c_{nm} = \Sigma_{p=1}^{T}\gamma_{mnp}q_p$ (2) where the products $q_1, q_2, ..., q_T$ are given by $q_p = (\Sigma\alpha_{ijp}a_{ij})(\Sigma\beta_{KLp}b_{KL})$ (3).

Thus, our aim is to form Brent-like equations for other problems such as complex multiplication and quaternion multiplication and then convert it to a SAT problem where we can apply our portfolio of SAT solvers to get the solution.

B. SAT Solvers

Satisfiability (SAT) is the problem of determining if the variables of a given Boolean formula can be assigned in a way as to make the formula evaluate to TRUE [13]. SAT was the first known example of an NP-complete problem. A wide range of other decision and optimization problems can be transformed into instances of SAT and a class of algorithms called SAT solvers can efficiently solve a large enough subset of SAT instances such as MiniSAT solver [23]. Our aim is to transform problems like MM into SAT problems.

C. Solving Brent Equations Modulo 2 and Lifting

In the first step we form the Brent Equations for our problem and we consider them over the field $\mathrm{GF}(2)$. We are interested only in simple solutions that work over small finite rings and fields. Then using the Courtois-Bard-Jefferson converter we convert this system of equations over $\mathrm{GF}(2)$ to a SAT problem and attempt to solve it. After obtaining the solution modulo 2 we begin again and try to lift the solution to a modulo 4 solution using very similar formal encoding.

D. Solving and Conversion

The system of equations is encoded algebraically and then converted to a SAT problem. We have implemented a method to convert this very hard problem to a SAT problem, and we have attempted to solve it, with our portfolio of some 500 SAT solvers and their variants.

III. Matrix Multiplication

Many attempts to solve the general MM problem in the literature work by solving fixed-size problems and applying the solution recursively. This leads to pure combinatorial optimization problems with fixed size. For square matrices the naive algorithm is cubic and the best known theoretical exponent is 2.376, due to Coppersmith and Winograd [14]. This exponent is quite low and it is conjectured that one should be able to do MM in so called "soft quadratic time", with possibly some poly-logarithmic overheads, which could even be sub-exponential in the logarithm. This in fact would be nearly linear in the size of the input.

In 2005 a team of scientists from Microsoft Research and two US universities established a new method for finding such algorithms based on group theory, and their best method so far gives an exponents of 2.41 [17], close to Coppersmith-Winograd result and subject to further improvement.

All attempts to solve the MM problem in the literature rely on solving certain fixed size problems, which can be the recursively applied to produce asymptotically fast algorithms that can be used for more general cases. In 1969 Victor Strassen established a first asymptotic improvement to the complexity of MM algorithm, by proving that two matrices 2×2 can be multiplied by using seven instead of eight multiplications [22]. Later in 1975 Laderman published a solution for multiplying 3×3 matrices with 23 multiplications [9]. Since then this topic generated very considerable interest and yet to this day it is not clear if Laderman's solution in case of 3×3 multiplication can be further improved.

As in many previous attempts to solve the problem we proceed by solving the so called Brent equations [3]. This approach has been tried many times before, see [[3],[8],[10],[12],[13],[11]].

We write the coefficients of each product as three 3×3-matrices for each multiplication $A^{(i)}, B^{(i)}$ and $C^{(i)}$, $1 \leq i \leq r$, with $r = 23$ where A will be the left hand side of each product, B the right hand size, and C says to which coefficient of the result this product contributes.

The Brent equations are as follows:

$\forall i \forall j \forall k \forall l \forall m \forall n \ \sum_{i=1}^{r} A_{ij}^{(i)} B_{kl}^{(i)} C_{mn}^{(i)} = \delta_{ni}\delta_{jk}\delta_{lm}$ (4)

For 3×3 matrices we get exactly 729 cubic equations. Then using our methodology we obtained the following solution for the case of 3×3 matrices. Our solution in non-isomorphic to any of the existing solutions:

$P01 := (a_{23}) * (-b_{12} + b_{13} - b_{32} + b_{33});$
$P02 := (-a_{11} + a_{13} + a_{31} + a_{32}) * (b_{21} + b_{22});$
$P03 := (a_{13} + a_{23} - a_{33}) * (b_{31} + b_{32} - b_{33});$
$P04 := (-a_{11} + a_{13}) * (-b_{21} - b_{22} + b_{31});$
$P05 := (a_{11} - a_{13} + a_{33}) * (b_{31});$
$P06 := (-a_{21} + a_{23} + a_{31}) * (b_{12} - b_{13});$
$P07 := (-a_{31} - a_{32}) * (b_{22});$
$P08 := (a_{31}) * (b_{11} - b_{21});$
$P09 := (-a_{21} - a_{22} + a_{23}) * (b_{33});$
$P10 := (a_{11} + a_{21} - a_{31}) * (b_{11} + b_{12} + b_{33});$
$P11 := (-a_{12} - a_{22} + a_{32}) * (-b_{22} + b_{23});$
$P12 := (a_{33}) * (b_{32});$
$P13 := (a_{22}) * (b_{13} - b_{23});$
$P14 := (a_{21} + a_{22}) * (b_{13} + b_{33});$
$P15 := (a_{11}) * (-b_{11} + b_{21} - b_{31});$
$P16 := (a_{31}) * (b_{12} - b_{22});$
$P17 := (a_{12}) * (-b_{22} + b_{23} - b_{33});$
$P18 := (-a_{11} + a_{12} + a_{13} + a_{22} + a_{31}) * (b_{21} + b_{22} + b_{33});$
$P19 := (-a_{11} + a_{22} + a_{31}) * (b_{13} + b_{21} + b_{33});$
$P20 := (-a_{12} + a_{21} + a_{22} - a_{23} - a_{33}) * (-b_{33});$
$P21 := (-a_{22} - a_{31}) * (b_{13} - b_{22});$
$P22 := (-a_{11} - a_{12} + a_{31} + a_{32}) * (b_{21});$
$P23 := (a_{11} + a_{23}) * (b_{12} - b_{13} - b_{31});$

$c_{11} = P02 + P04 + P07 - P15 - P22;$
$c_{12} = P01 - P02 + P03 + P05 - P07 + P09 + P12$
$+P18 - P19 - P20 - P21 + P22 + P23;$
$c_{13} = -P02 - P07 + P17 + P18 - P19 - P21 + P22;$
$c_{21} = P06 + P08 + P10 - P14 + P15 + P19 - P23;$
$c_{22} = -P01 - P06 + P09 + P14 + P16 + P21;$
$c_{23} = P09 - P13 + P14;$
$c_{31} = P02 + P04 + P05 + P07 + P08;$
$c_{32} = -P07 + P12 + P16;$
$c_{33} = -P07 - P09 + P11 - P13 + P17 + P20 - P21;$

Lemma 1: : Our new solution is neither equivalent to the Ladermans solution [9] nor equivalent to any of the solutions given in [1].

Proof:

Following [1], the Ladermans solution has exactly 6 matrices of rank 3 (which occur in products $P01, P03, P06, P10, P11, P14$). At the same time in all new solutions presented in [1], at most 1 matrix will have rank 3. In our solution we have exactly 2 matrices of rank 3 (which occur in products $P18$ and $P20$, there are 2 and not more such matrices, both being on the left hand size namely $A^{(18)}$, in $A^{(20)}$). This proves that all these solutions are distinct.

Remark: This result demonstrates that the space of solutions to Ladermans problem is larger than expected, and therefore it becomes now more plausible that a solution with 22 multiplications exists. If it exists, we might be able to find it soon just by running our algorithms longer, or due to further improvements in the SAT algorithms.

IV. Complex number multiplication

In order to compute the product $(a + b\mathrm{i}) * (c + d\mathrm{i}) = (ac - bd) + (ad + bc)\mathrm{i}$ (5) we need 4 multiplications using the naive algorithm. Gauss was the first to prove that the multiplication of two complex numbers $(a+b\mathrm{i})*(c+d\mathrm{i})$ can be done using 3 multiplications instead of 4. We obtained the same result using our methodology. We can translate this complex multiplication problem to a MM problem using the isomorphism between the set of complex numbers $\{a + b\mathrm{i} : a, b \in \mathbb{R}\}$ and the 2 dimensional sub-algebra of $\{\begin{pmatrix} a & b \\ c & d \end{pmatrix} : a, b, c, d \in \mathbb{R}\}$, given by:

$\{\begin{pmatrix} a & -b \\ b & a \end{pmatrix} : a, b \in \mathbb{R}\}.$

In the first step we form the 3-dimensional Brent Equations for multiplying two 2x2 matrices A and B and then using SAT solvers and lifting techniques we obtain the seven following Strassen-like products, which can be used to compute the entries $\{c_{11}, c_{12}, c_{21}, c_{22}\}$ of the matrix $C = AB$.

$P1 = (a_{12} + a_{22}) * (b_{12} + b_{22});$
$P2 = (a_{11}) * (b_{11});$
$P3 = (a_{21}) * (b_{11} + b_{12} + b_{21} + b_{22});$
$P4 = (a_{12}) * (-b_{21});$
$P5 = (-a_{11} + a_{12} - a_{21} + a_{22}) * (b_{12});$
$P6 = (-a_{21} + a_{22}) * (b_{21} + b_{22});$
$P7 = (-a_{12} + a_{21} - a_{22}) * (b_{12} + b_{21} + b_{22});$
$c_{11} = P2 - P4;$
$c_{12} = P4 - P5 - P6 - P7;$
$c_{21} = P3 + P4 - P1 - P7;$
$c_{22} = P1 + P6 + P7 - P4;$

Now if we consider these products over the 2-dimensional sub-algebra of matrices defined before we get that $\mathrm{Span}\{P_1, .., P_7\} = \mathrm{Span}\{P_1, .., P_4\}$ since we have $P_5 = 2P_4, P_7 = P_3 - P_2$ (6) and $P_6 = 2P_2 - P_3 - P_1$ (7). This suggests that four products are enough to compute the product of two complex numbers as the naive multiplication. However, if we also consider the set of entries $\{c_{11}, c_{21}\}$ over the new set of products we have that $c_{11} = P_2 - P_4$ (8) and $c_{21} = P_2 + P_4 - P_1$ (9). As we see, our method gives three multiplications in total as proposed by Gauss.

A. Multiplication of three complex numbers

We provide an exceptionally good solution which exists over GF(2) in the non-homogenous case for the problem of multiplying three complex numbers. Multiplication of three complex numbers is a trilinear problem as we aim to minimize the number of multiplications needed to represent the map $f : (V, V, V) \rightarrow V$.

Using our method we show that multiplication of three complex numbers $(a + b\mathrm{i}) * (c + d\mathrm{i}) * (e + f\mathrm{i})$ can be achieved using five multiplications at most.

Lemma 2: : $MC((a+b\mathrm{i})*(c+d\mathrm{i})*(e+f\mathrm{i})) \leq 5$ over $\mathrm{GF}(2)$.

Proof:
In $\mathrm{GF}(2)$ we can do 5 multiplications total!
$P1 := (a+b+e+f)*(c+d+e+f)$;
$P2 := (a+e)*(d+e)$;
$P3 := (c+f)*(b+f)$;
$Im := P4 := (P1+P2+P3+a+d+e)*(P1+e+f)$;
$Re := P5 := (P1+e+f)*(P1+P4+a+b+c+d+1)$;

V. Quaternion Algebra

Quaternions are a number system that extends the complex multiplication that were introduced by the Irish Mathematician Sir William Rowan Hamilton, who defined a quaternion as the quotient of two directed lines in a three-dimensional space or equivalently as the quotient of two vectors [7]. It can also be seen as the sum of a scalar and a vector. They are widely used in both theoretical and applied mathematics, especially for calculations involving three-dimensional rotations such as three-dimensional computer graphics and computer vision and in real-time symmetric cryptography [6].

As a set, the quaternions are equal to $\mathbb{R}^4$ and every element can be represented as:

$a1 + bi + cj + dk$, where i, j, k satisfy the following relations;

$$i^2 = j^2 = k^2 = ijk = -1,\ ij = k, ji = -k,\ jk = i, kj = -i \text{ and } ki = j, ik = -j \ (10).$$

The Hamilton product of two quaternions:
$a_1 + b_1 i + c_1 j + d_1 k$, $a_2 + b_2 i + c_2 j + d_2 k$ is given by
$(a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)i$
$+(a_1c_2 + b_1d_2 + c_1a_2 + d_1b_2)j + (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k$
(11).

Our aim is to compute the minimum number of 2-input multiplications needed to compute the product of two quaternions. Using the naive multiplication method we need 16 multiplications but this number of multiplication can be reduced using the Gauss method to 12. Using our software we obtain the 12 products that are needed to compute the product of two quaternions over the general non-commutative setting. Additionally, we further investigate the number of 2-input multiplication needed over $\mathrm{GF}(2)$ and we surprisingly get eight. Below we provide the encoding of quaternion multiplication problem into Brent Equations and the next *Lemmas* provide the result obtained by our software.

Encoding $q_1 * q_2$ into Brent Equations:
Suppose $\{a_1, a_2, a_3, a_4\}$, $\{b_1, b_2, b_3, b_4\}$ are non-commutative variables and σ_{ijk} is a given three-dimensional array of numbers from the set $\{-1, 0, 1\}$, and we want to compute the 4 sums of 2-input products: $a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4, a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3, a_1b_3 + a_2b_4 + a_3b_1 + a_4b_2, a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1$.

Then our aim is to find the least possible T and scalars $\alpha_{it}, \beta_{jt}, \gamma_{kt}$ such that from the T products of the form
$p_t = (\sum_i \alpha_{it} a_i).(\sum_j \beta_{jt} b_j)$ (12) for $1 \leq t \leq T$, we can form the q_k as linear combinations of the p_t as
$q_k = \sum_{t=1}^{T} \gamma_{kt} p_t$ (13) for $1 \leq k \leq K$.

Combining these two results we formulate the problem of finding the minimum number of 2-input multiplications for multiplying two quaternions $a_1 + a_2 i + a_3 j + a_4 k$, $b_1 + b_2 i + b_3 j + b_4 k$ as follows:

Quaternion multiplication problem: Find constants $\alpha_{it}, \beta_{jt}, \gamma_{kt}$ and least T (where $T \leq 12$) such that the following system of 64 equations in $12 * T$ unknowns hold:

$$\sum_{t=1}^{T} \alpha_{it}\beta_{jt}\gamma_{kt} = \sigma_{ijk} \ (14),$$
$$\text{for } 1 \leq i \leq 4,\ 1 \leq j \leq 4,\ 1 \leq k \leq 4$$

,where σ_{ijk}: $\sigma_{111} = 1$, $\sigma_{122} = 1$, $\sigma_{133} = 1$, $\sigma_{144} = 1$, $\sigma_{212} = 1$, $\sigma_{221} = -1$, $\sigma_{234} = 1$, $\sigma_{243} = 1$, $\sigma_{313} = 1$, $\sigma_{324} = -1$, $\sigma_{331} = -1$, $\sigma_{342} = 1$, $\sigma_{414} = 1$, $\sigma_{423} = 1$, $\sigma_{432} = -1$, $\sigma_{441} = -1$ and zero elsewhere.

Lemma 3: $MC(q_1 * q_2 : q_i \in \mathbb{H}) \leq 12$

Proof: Using the complex representation of q_1 and q_2 we need to compute four entries of the form:
1) $(q_1 * q_2)_{11} = (a+bi)*(e+fi) + (c+di)*(-g+hi)$
2) $(q_1 * q_2)_{12} = (a+bi)*(g+hi) + (c+di)*(e-fi)$
3) $(q_1 * q_2)_{21} = (-c+di)*(e+fi) + (a-bi)*(-g+hi)$
4) $(q_1 * q_2)_{22} = (-c+di)*(g+hi) + (a-bi)*(e-fi)$

Using Gauss formulaes we can obtain the first two entries $\{(q_1 * q_2)_{11}, (q_1 * q_2)_{12}\}$ using 12 multiplications. Using this methodology we have obtained the following terms $ae - bf, be+af, ce+fd, ed-fc, ag-bh, bg+ah, -cg-hd, ch-dg$. However the other entries $\{(q_1 * q_2)_{21}, (q_1 * q_2)_{22}\}$ can be computed using these terms multiplied by -1. Using our software we obtained the following formulas for the quaternion multiplication using 12 multiplications which can be also directly verified using MAPLE computer algebra software:
$P01 := (a_4)*(b_2)$;
$P02 := (a_1)*(b_1 + b_2 + b_4)$;
$P03 := (a_1)*(b_3)$;
$P04 := (-a_1 + a_2)*(b_1)$;
$P05 := (-a_2)*(b_1 - b_2)$;
$P06 := (a_2)*(b_3)$;
$P07 := (a_2)*(b_4)$;
$P08 := (a_3)*(b_1)$;
$P09 := (a_1 + a_3 - a_4)*(b_1 + b_2)$;
$P10 := (a_3 + a_4)*(-b_3)$;
$P11 := (a_1 - a_3 + a_4)*(b_4)$;
$P12 := (-a_4)*(-b_3 + b_4)$;
$\mathrm{expand}(-P04 - P05 + P10 + P12 - a_1*b_1 + a_2*b_2 + a_3*b_3 + a_4*b_4)$;
$\mathrm{expand}(P02 + P04 - P11 - P12 - a_1*b_2 - a_2*b_1 - a_3*b_4 + a_4*b_3)$;
$\mathrm{expand}(P01 + P03 + P07 + P08 - a_1*b_3 - a_2*b_4 - a_3*b_1 - a_4*b_2)$;
$\mathrm{expand}(-P01 + P02 + P06 + P08 - P09 - a_1*b_4 - a_2*b_3 + a_3*b_2 - a_4*b_1)$;

Additionally, we obtain a result over the field $\mathrm{GF}(2)$ and our results are summarized in the next lemma. Obtaining results over the field of two elements is very useful as binary encoding is employed in many areas such as cipher design in cryptography and circuit design for either software or hardware implementations.

Lemma 4: $MC(q_1 * q_2 : q_i \in \mathbb{H}) \leq 8$ over $\mathrm{GF}(2)$.

Proof: Using our automated software we obtained the following solution which can be directly verified with MAPLE computer algebra software:

$P01 := (a_2 + a_3) * (b_1 + b_2 + b_4);$
$P02 := (a_1 + a_2 + a_3) * (b_1 + b_2 + b_3 + b_4);$
$P03 := (a_1 + a_2) * (b_2 + b_3 + b_4);$
$P04 := (a_1 + a_3) * (b_1 + b_2 + b_3);$
$P05 := (a_3 + a_4) * (b_1);$
$P06 := (a_1 + a_2 + a_3 + a_4) * (b_2);$
$P07 := (a_2 + a_4) * (b_4);$
$P08 := (a_1 + a_4) * (b_3);$

$\mathrm{expand}(P01 + P02 + P03 + P07 - a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + a_4 * b_4) mod 2;$

$\mathrm{expand}(P02 + P03 + P04 + P08 - a_1 * b_2 - a_2 * b_1 - a_3 * b_4 + a_4 * b_3) mod 2;$

$\mathrm{expand}(P01 + P02 + P03 + P04 + P06 - a_1 * b_3 - a_2 * b_4 - a_3 * b_1 - a_4 * b_2) mod 2;$

$\mathrm{expand}(P01 + P02 + P04 + P05 - a_1 * b_4 - a_2 * b_3 + a_3 * b_2 - a_4 * b_1) mod 2;$

VI. Exact Circuit Complexity Optimization

In case of circuit complexity we employed the heuristic proposed by Boyar and Peralta [18] based on the notion of MC and consists of the following steps:

1. (Step 1) First compute the MC.
2. (Step 2) Then optimize the number of XORs separately, see [[19],[21]].
3. Optional Step 3: At the end do additional optimizations to decrease the circuit depth, and possibly additional software optimizations, see [[18],[20]].

We encode the problem formally as a straight-line representation problem, described by a quantified set of multivariate relations and we convert it to SAT with the Courtois-Bard-Jefferson tool [2]. Our method on how we compute the MC of the circuit is found in [5].

As a proof of concept we consider the following S-box with 3 inputs and 3 outputs, which have been generated at random for the CTC2 cipher [5] and is defined as $7, 6, 0, 4, 2, 5, 1$. We have tried to optimize this S-box with the well known software Logic Friday (based on Espresso min-term optimization developed at Berkeley) and obtained 13 gates. With our software and in a few seconds we obtained several interesting results, each coming with a proof that it is an optimal result (cannot be improved anymore). We get:

Lemma 5: The Multiplicative Complexity (MC) is exactly 3 (we allow 3 AND gates and an unlimited number of XOR gates).

The Bitslice Gate Complexity (BGC) is exactly 8 (allowed are XOR,OR,AND,NOT).

The Gate Complexity (GC) is exactly 6 (allowing NAND,NOR,NXOR).

The NAND Complexity (NC) is exactly 12 (only NAND gates and constants).

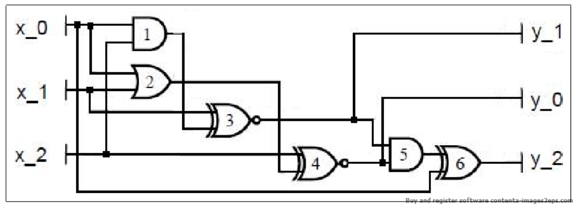


Fig. 1. Our provably optimal implementation of CTC2 S-box with 6 gates.

Proof: Unlike the great majority of circuit optimizations, needed each time a given cipher is implemented in hardware, our results are exact. They are obtained by solving the problem at a given gate count k, the SAT solver outputs SAT and a solution, and if for k-1 gates the *SAT* solver is good enough and fast enough, it will output *UNSAT* and we obtain a proven lower bound, a rare thing in complexity, see [5].

VII. Conclusion

In this paper we study the notion of Multiplicative Complexity (MC) which minimizes the number of elementary non-linear operations (AND gates) at the cost of linear operations. We used MC as an essential tool for optimizing potentially arbitrary algebraic computations over fields and rings in the general non-commutative setting.

We employed an automated method for obtaining new formulas for Matrix Multiplication (MM), complex number and quaternion multiplication based on SAT solvers. We extensively used the notion of Brent Equations [3] as a formal encoding of these problems and then we consider solutions of the corresponding system of equations over the field of two elements. After we algebraically encode the problem we convert it into a SAT problem using the Courtois-Bard-Jefferson [2] and then using our portfolio of 500 SAT solvers we try to solve the problem over $\mathrm{GF}(2)$. Starting from scratch we try to lift the solutions modulo 2 to solutions modulo 4 and also to bigger fields. We lift the solutions using another constraint satisfaction algorithm and some heuristics discovered during our simulations that reduces the complexity of our lifting technique even more.

We have been able to obtain new results in decreasing the MC of several well known operations in algebra, which to the best of our knowledge are new. For example we have obtained a new general 3×3 MM method with 23 multiplications [4]. We also derived new formulaes regarding the multiplication of three complex numbers using 5 multiplications over $\mathrm{GF}(2)$ and for multiplying two quaternions using 8 multiplications over $\mathrm{GF}(2)$. We also derived efficient implementations regarding the MC of some ciphers such as PRESENT, GOST and CTC2 [5].

So far our method works efficiently for obtaining compact representations of algebraic computations or circuits over the field of two elements. In some cases we are able to lift our solutions from GF(2) to the general non-commutative setting. However, our lifting technique sometimes is not efficient and is not able to lift the solutions. As future work we will improve our lifting techniques so that we will be able to obtain similar compact representations which hold over arbitrary non-commutative rings.

ACKNOWLEDGMENT

We would like to thank the anonymous referees of this paper who helped us a lot to improve it.

REFERENCES

[1] R.W. Johnson and A.M. McLoughlin, *Noncommutative Bilinear Algorithms for 3 x 3 Matrix Multiplication* In SIAM J. Comput., vol. 15 (2), pp.595-603, 1986.

[2] G.V. Bard, N.T. Courtois and C. Jefferson, *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers* Presented at ECRYPT workshop Tools for Cryptanalysis, 2007.

[3] R. Brent, *Algorithms for matrix multiplication* Tech. Report Report TR-CS-70-157,Department of Computer Science, Stanford, 52 pages, 1970.

[4] N.T. Courtois, G.V. Bard2, and D. Hulme, *A New General-Purpose Method to Multiply 3x3 Matrices Using Only 23 Multiplications* At http://arxiv.org/abs/1108.2830, 2011.

[5] N.T. Courtois, D. Hulme, and T. Mourouzis, *Solving Circuit Optimisation Problems in Cryptography and Cryptanalysis* Appears in electronic proceedings of 2nd IMA Conference Mathematics in Defence, UK, Swindon, 2011.

[6] R. Anand, G. Bajpai and V. Bhaskar, *Real-Time Symmetric Cryptography using Quaternion Julia Set* IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.3, 2009

[7] W.R. Hamilton, *On quaternions, or on a new system of imaginaries in algebra* Philosophical Magazine. Vol. 25, n 3. p. 489495, 1844

[8] G. Bard, *New Practical Approximate Matrix Multiplication Algorithms found via Solving a System of Cubic Equations* A draft paper submitted to a journal, can be found at: http://www-users.math.umd.edu/ bardg/

[9] J.D. Laderman, *A Non-Commutative Algorithm for Multiplying 3x3 Matrices Using 23 Multiplications* ull. Amer. Math. Soc. Volume 82, Number 1, 1976

[10] W. Smith, *Fast Matrix Algorithms And Multiplication Formulae* Available at:https://math.cst.temple.edu/ wds/matgrant.ps.

[11] N. Burr, *An investigation into fast matrix multiplication* done under supervision of Nicolas T. Courtois, and submitted as a part of BSc Degree in Computer Science at Univesity College London, 2010

[12] G. Bard, *New Practical Approximate Matrix Multiplication Algorithms found via Solving a System of Cubic Equations* A draft paper submitted to a journal, can be found at: http://www-users.math.umd.edu/ bardg/

[13] G. Bard, *Algorithms for Solving Linear and Polynomial Systems of Equations over Finite Fields with Applications to Cryptanalysis* Submitted in Partial Fulfillment for the degree of Doctor of Philosophy of Applied Mathematics and Scientific Computation, 2007

[14] D. Coppersmith and S.Winograd *On the asymptotic complexity of matrix multiplication* SIAM Journal Comp., 11, pp 472-492 , 1980

[15] E. Prouff, C. Giraud, and S. Aumonier *Provably Secure S-Box Implementation Based on Fourier Transform* In CHES 2006, Springer LNCS 4249, pp: 216-230, 2006

[16] M. Albrecht, N.T. Courtois, D. Hulme. and G. Song *Bit-Slice Implementation of PRESENT in pure standard C* , 2011

[17] H. Cohn, R. Kleinberg, B. Szegedyz and C. Umans *Grouptheoretic Algorithms for Matrix Multiplication* In FOCS05, 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 379, 2005

[18] J. Boyar and R. Peralta *A New Combinational Logic Minimization Technique with Applications to Cryptology* In SEA 2010: 178-189, 2009

[19] J. Boyar, P. Matthews and R. Penalta, *On the Shortest Linear Straight-Line Program for Computing Linear Forms* In MFCS, 2008

[20] J. Boyar and R.Peralta *A depth-16 circuit for the AES S-box* http://eprint.iacr.org/2011/332

[21] C. Fuhs and P. Schneider-Kamp *Synthesizing Shortest Linear Straight-Line Programs over GF(2) Using SAT* In SAT 2010, Theory and Applications of Satisfiability Testing, Springer LNCS 6175, pp. 71-84, 2010
Volker Strassen, ,

[22] V. Strassen *Gaussian elimination is not optimal* Numerische Mathematik 13 pp. 354-356, 1969

[23] N. Sorensson and N. Een *Minisat v1. 13-a sat solver with conflict-clause minimization* SAT journal pp. 53, 2005

UPC-CompilerCheck: A Tool for Evaluating Error Detection Capabilities of UPC Compilers

Marina Kraeva[†], James Coyle[‡], Glenn R. Luecke[*], Indranil Roy[§], Elizabeth Kleiman[‖], and James Hoekstra[¶]
High Performance Computing Group, Iowa State University,
Ames, Iowa 50011, USA
Email: [†]kraeva@iastate.edu, [‡]jjc@iastate.edu, [*]grl@iastate.edu,
[§]iroy@iastate.edu, [‖]ekleiman@mtmercy.edu and [¶]hoekstra@iastate.edu

***Abstract*—The ability of system software to detect compile-time errors and issue messages that help programmers quickly fix these errors is an important productivity criterion for developing and maintaining application programs. To evaluate this capability for Unified Parallel C (UPC) compilers, 3141 Compile-Time Error Detection (CTED) tests and a CTED evaluation tool, called UPC-CompilerCheck, have been developed. UPC-CompilerCheck assigns a score from 0 to 5 for each compiler-generated error message based on the usefulness of the information in the message to help a programmer fix the error quickly. This tool also calculates average scores for each error category and then prints the results. Compiler vendors could use UPC-CompilerCheck to evaluate and improve the compile-time error detection capabilities of their UPC compilers. All tests, UPC-CompilerCheck and test results for the Berkeley, Cray, GNU and HP UPC compilers are freely available.**

Keywords-Languages; UPC; compile-time error detection.

I. Introduction

Unified Parallel C (UPC) is an extension of the C programming language for parallel execution on shared and distributed memory parallel machines [1], [2]. UPC uses a single shared, partitioned address space, where shared variables may be directly read and written by any thread. Shared variables are stored in the memory of the thread for which they have affinity. "UPC combines the programmability advantages of the shared memory programming paradigm and the control over data layout and performance of the message passing programming paradigm" [3]. Providing a productive programming environment for UPC will encourage new scientific applications to be written in UPC. Since debugging UPC programs can be time consuming, it is important to have UPC compilers, tools and run-time systems that can detect both compile-time and run-time errors and issue messages that help programmers quickly fix the errors. A tool to evaluate error detection capabilities of UPC run-time systems has already been developed [4]. This paper describes the UPC-CompilerCheck tool for evaluating error detection capabilities of UPC compilers.

Application programs are usually developed by (a) writing the application, (b) compiling the application and fixing all errors detected at compile time, (c) running the application and fixing all errors detected at run-time and (d) then validating the program using problems for which answers are known. Compile-time tools cannot be expected to find all errors, so run-time error detection tools such as [5], [6] will often be needed. However, when errors can be found at compile-time, programmer productivity will be increased.

To evaluate error detection capabilities of UPC compilers, 3141 UPC compile-time error detection (CTED) tests and the UPC-CompilerCheck tool have been developed by ISU's HPC Group. Each test contains exactly one UPC compile-time error. The UPC-CompilerCheck tool compiles these tests, assigns a score from 0 to 5 based on the quality of the error message, calculates the averages of these scores for each error category and reports results.

These tests and UPC-CompilerCheck provide an easy way to evaluate and compare compile-time error detection capabilities of different UPC compilers and could be used as part of a computer procurement process along with the UPC RTED tests [4]. In addition, compiler vendors could use the CTED tests, recommended error messages and UPC-CompilerCheck to evaluate and improve the compile-time error detection capabilities of their UPC compilers.

UPC compile-time tests, recommended error messages, UPC-CompilerCheck and test results are freely available [7]. As new UPC compilers/releases become available, vendors and researchers are encouraged to send results to cted.project@iastate.edu so that they can be posted on this web site.

The paper is structured as follows. Section II provides background on UPC and on UPC tools. Section III describes the design of UPC-CompilerCheck, and how it is used. Section IV shows examples of actual error messages along with their scores, and describes why each score was assigned. Section V provides scoring averages for each error category for several UPC compilers. Section VI contains our conclusions about the current state of UPC compilers for finding the various types of errors at compile time.

II. Background

The UPC Compiler Group at the University of California Berkeley/Lawrence Berkeley National Laboratory actively participated in writing of the UPC Specification and developed the first UPC compiler. This compiler is an open-source and portable implementation of UPC [3]. Cray, HP, GNU and IBM have also developed UPC compilers.

The UPC working group at the High Performance Computing Lab (HPCL) at George Washington University is involved in the UPC specification, UPC testing strategies, UPC documentation, UPC testing suites, UPC benchmarking, and UPC collective and Parallel I/O specification [8].

At Michigan Technological University work on UPC includes the recent release of the MuPC run-time system for UPC as well as collective specification development, memory model research, programmability studies, and test suite development [9].

Researchers at the University of Florida's High Performance Computing and Simulation Laboratory are currently involved in the research and development of a next-generation performance analysis tool supporting UPC. This tool helps users to identify bottlenecks in their programs and serves as a test-bed for advanced analysis techniques aimed at increasing programmer productivity [10].

The High Performance Computing Group from Iowa State University has developed a run-time error detection tool called UPC-CHECK that includes deadlock detection [5], [11]. In addition the ROSE-CIRM tool [6] has been developed by Dan Quinlan's group at Lawrence Livermore National Laboratory to complement UPC-CHECK by detecting other run-time errors. Ali Ebnenasir from the Department of Computer Science of Michigan Technological University developed UPC-SPIN [12], a software framework for the model checking of the inter-thread synchronization functionalities of Unified Parallel C (UPC) programs. A list of UPC programming tools can be found in the Programming Tools section of the UPC Wiki page [2].

III. METHODOLOGY

This section summarizes the methodology used to develop compile-time error tests and UPC-CompilerCheck. For each error, a program has been written that contains the specified error and no other errors (each program contains one and only one compile-time error). For each test a file with a recommended error message was created that contains the error name, the line number and the file name where the error occurs along with any additional information that would assist a programmer to find and correct the error.

The UPC compile-time error tests have been written to cover a wide range of errors in many different situations. The following are the UPC compile-time error categories:

- Items that the UPC specification explicitly does not allow and that should be detected at compile-time
- Out-of-bounds shared memory access using indices
- Out-of-bounds shared memory access using pointer references
- Out-of-bounds shared memory access in UPC library functions
- Argument errors in UPC library functions
- Wrong order of UPC statements and function calls
- Uninitialized variables
- Deadlocks
- Race conditions
- Memory leaks and memory related errors
- Operations specifically undefined by the UPC specification
- Warnings

The UPC CTED evaluation tool is a collection of scripts for compiling the tests, comparing actual messages with expected messages and then assigning a score of 0, 1, 2, 3, 4 or 5 to the message generated by each test. Scores for messages are assigned as follows:

- A score of 0 is given when the error is not detected.
- A score of 1 is given for error messages with the correct error name.
- A score of 2 is given for error messages with the correct error name and line number where the error occurred but not the file name where the error occurred.
- A score of 3 is given for error messages with the correct error name, line number and the name of the file where the error occurred.
- A score of 4 is given for error messages which contain at least the information required for a score of 3 but less information than needed for a score of 5.
- A score of 5 is given in all cases when the error message contains all the information needed for fixing the error quickly.

The scoring is the same as was done for the run-time tests [4] even though for compile-time tests if a compiler identifies the correct line number it is likely that it also will identify the correct file name. This means for compile-time tests that the score of 2 will likely never be given. The information needed for scores of 4 and 5 is tailored to each test. Examples in Section IV illustrate this.

Different compilers may issue different messages (with different error names) for the same compile-time error. UPC-CompilerCheck has a list of synonymous phrases for each error so that equivalent error messages will be evaluated appropriately. Additional synonymous phrases may need to be added as new compilers/releases become available. Error messages were evaluated by UPC-CompilerCheck as follows:

- For each test and score, a scoring script was created.
- Error messages were reduced to a canonical form for easy comparison with the recommended error messages by first changing all text to lower case and then replacing selected phrases with standard phrases. Blanks, hexadecimal addresses, and integers longer than three digits are removed to reduce false matches.

- Scoring scripts were applied to the canonical form of each error message for evaluation.

UPC-CompilerCheck has been designed for easy usage. To run all tests one sets up the configuration file and then issues the 'run_tests_all' command. Sample configuration files for each compiler are provided. By issuing the 'run_tests_all <error_category>' command the user can run only the tests in the selected error category. The 'run_tests_all' command also calculates average scores for each error category and then prints the results. UPC-CompilerCheck also allows one to run individual tests and to examine the output.

IV. Examples

This section contains four examples to illustrate how the tests have been written and how messages were scored.

A. Example 1

Applying a binary operator with incorrect operands.

Example 1: **c_A_3_1_a_A.upc**

```
...
26 #include "upcparam.h"
27 #include <stddef.h>
28
29 shared [4] char Arr_A[4*THREADS];
30
31 int main() {
32       shared [4] char *Ptr_S;
33       char *Ptr_L;
34       ptrdiff_t diff;
35
36       Ptr_S=&Arr_A[4*MYTHREAD+1];
37       Ptr_L=(char *)&Arr_A[4*MYTHREAD];
38
39       diff=Ptr_S-Ptr_L;
40
41       if(MYTHREAD==0) {
42             printf("diff = %d\n",diff);
43       }
44
45       return 0;
46 }
```

The following is the recommended error message:

```
ERROR: incorrect operands
An attempt to apply subtraction binary
operator to pointer-to-shared 'Ptr_S'
and pointer-to-local 'Ptr_L' is made at
line 39 in file 'c_A_3_1_a_A.upc'.
The pointer 'Ptr_S' is declared at line
32 in file 'c_A_3_1_a_A.upc'.
The pointer 'Ptr_L' is declared at line
33 in file 'c_A_3_1_a_A.upc'.
```

A score of 3 was given to the Berkeley UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
c_A_3_1_a_A.upc: In function 'main':
c_A_3_1_a_A.upc:39: warning: Attempt to
take the difference of pointer-to-shared
and pointer-to-private
```

A score of 3 was given to the GNU UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
c_A_3_1_a_A.upc: In function â:
c_A_3_1_a_A.upc:39: error: Attempt
to take the difference of shared and
nonshared pointers
```

A score of 0 was given to the Cray and HP UPC compilers since they did not detect the error.

B. Example 2

Using an uninitialized pointer.

Example 2: **c_H_2_l.upc**

```
. . .
25 #include "upcparam.h"
26 #define N 10
27
28 int main() {
29       shared double *ptr_x;
30       double* ptr_x1;
31
32       if(MYTHREAD==THREADS/2) {
33             ptr_x1=(double*)ptr_x;
34             ptr_x=(shared double*) upc_alloc(N*sizeof(double));
35             ptr_x1–;
36             printf("ptr_x=%p; ptr_x1=%p \n", (double*) ptr_x,
                        (double*) ptr_x1);
37
38             upc_free(ptr_x);
39       }
40
41       return 0;
42 }
```

The following is the recommended error message:

```
ERROR: uninitialized pointer
An attempt to assign pointer 'ptr_x'
that is not explicitly initialized to
another pointer is made at line 33 in
file 'c_H_2_l.upc'.
The pointer 'ptr_x' is declared at line
29 in file 'c_H_2_l.upc'.
```

A score of 4 was given to the Cray UPC compiler for issuing the following message since it correctly identified the error, file name, line number and gave the variable name. It was not given a score of 5 since the message did not give the line number where ptr_x was declared.

```
CC-7212 cc: WARNING File = c_H_2_l.upc,
Line = 33
```

```
Variable ``ptr_x'' is used before it is
defined.
```

A score of 4 was given to the HP UPC compiler for issuing the following message since it correctly identified the error, file name, line number and gave the variable name. It was not given a score of 5 since the message did not give the line number where ptr_x was declared.

```
``c_H_2_l.upc'', line 33: warning:
variable ``ptr_x'' is used before its
value is set.
ptr_x1=(double*)ptr_x;
                ^
```

A score of 0 was given to the Berkeley and GNU UPC compilers since they did not detect the error.

C. Example 3

An out-of-bounds array access error.

Example 3: **c_D_1_d_E.upc**

```
. . .
25 #define N 40
26 #define M 45
27
28 shared [] long double arrA[N]; /*DECLARE1*/
29 int main() {
30      long double var_res;
31      int i;
32
33      upc_forall(i=0;i<N;i++;&arrA[i])
34      arrA[i] = (long double)(i+1);
35      var_res = 10;
36      upc_barrier;
37
38      if(MYTHREAD == (THREADS-1)){
39      /*ERROR*/
40           arrA[N-M] = var_res;
41           for(i=0;i<N;i++)
42                printf("arrA[%d]=%d\n", i, (int)arrA[i]);
43      }
44
45      return 0;
46 }
```

The following is the recommended error message:

```
ERROR: out of bounds
Index value -5 is out of bounds for
array 'arrA' at line 40 in file
'c_D_1_d_E.upc'.
The array 'arrA' is declared with bounds
0:39 at line 28 in file 'c_D_1_d_E.upc'.
```

A score of 3 was given to the HP UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
``c_D_1_d_E.upc'', line 40: warning:
subscript out of range
arrA[N-M] = var_res;
     ^
```

A score of 3 was given to the Cray UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
CC-175 cc: WARNING File = c_D_1_d_E.upc,
Line = 40
The indicated subscript is out of range.
arrA[N-M] = var_res;
     ^
```

A score of 0 was given to the Berkeley and GNU UPC compilers since they did not detect the error.

D. Example 4

An array declarator error when compiled with the dynamic threads environment option.

Example 4: **c_A_4_3_b.upc**

```
. . .
24 #include "upcparam.h"
25
26 #define SIZE 10
27
28 shared [2] int Arr_A[SIZE];
29
30 int main() {
31      int i;
32
33      if(MYTHREAD==0) {
34           for(i=0;i<SIZE;i++)
35                Arr_A[i]=SIZE+i;
36           printf("Arr_A[0]=%d\n", Arr_A[0]);
37      }
38
39      return 0;
40 }
```

The following is the recommended error message:

```
ERROR: invalid array declarator
THREADS is not used in the array
'Arr_A' declaration at line 28 in file
'c_A_4_3_b.upc'.
In the dynamic translation environment,
THREADS must appear exactly once in
declarations of shared arrays with
definite block size, either alone
or multiplied by an integer constant
expression.
```

The Cray UPC compiler was given a score of 5 since it contains all the information in the recommended error message. The Cray UPC compiler issued the following message:

```
CC-1560 cc: ERROR File = c_A_4_3_b.upc,
Line = 28
One dimension of an array of a shared
```

```
type must be a multiple of THREADS when
the number of threads is nonconstant.
shared [2] int Arr_A[SIZE];
                         ^
```

The Berkeley UPC compiler was also given a score of 5 for issuing the following message:

```
upcc: error during UPC-to-C translation
(sgiupc stage):
c_A_4_3_b.upc:28: In the dynamic
translation environment, THREADS must
appear exactly once in declarations of
shared arrays with definite block size.
Offending variable: Arr_A
```

The GNU UPC compiler was given a score of 3 for issuing the following message:

```
c_A_4_3_b.upc:28: error: variable-size
type declared outside of any function
c_A_4_3_b.upc:28: error: variable-size
type declared outside of any function
```

The HP UPC compiler was given a score of 5 for issuing the following message:

```
``c_A_4_3_b.upc``, line 28: error: one
dimension of an array of a shared type
must be a multiple of THREADS when the
number of threads is nonconstant
shared [2] int Arr_A[SIZE];
                         ^
```

V. RESULTS

Table I presents the average scores for each error category when compiling the UPC CTED tests using the Berkeley, Cray, GNU and HP UPC compilers. Authors were not able to get access to the IBM UPC compiler. Current results are listed on the web site [7].

The category "explicitly disallowed statements" contains items that the UPC specification explicitly does not allow and that should be detected at compile-time. The category "undefined UPC operations" contains situations where the outcome of certain UPC statements is stated as being undefined by the UPC specification. The "warnings" category includes tests where programmers should be warned of likely errors, e.g., use of deprecated functions, shared variables not initialized by the program, etc. The "argument errors in UPC library functions" category covers those situations where inconsistent and/or incorrect information is passed as arguments to UPC library functions. At the time this project was done, the GNU and HP UPC compilers did not support UPC I/O; so, they scored zero on these tests. Notice from

Error category	**Berkeley**	**Cray**	**GNU**	**HP**
explicitly disallowed statements	2.92	2.75	3.21	2.62
out-of-bounds shared memory access using indices	0.00	1.00	0.00	1.27
out-of-bounds shared memory access using pointers	0.00	0.00	0.00	0.25
out-of-bounds shared memory access in UPC function calls	0.00	0.00	0.00	0.00
argument errors in UPC functions	0.00	0.07	0.05	0.09
wrong order of UPC statements and function calls	0.00	0.15	0.00	0.10
uninitialized variables	0.00	1.14	0.00	2.29
deadlocks	0.00	0.00	0.00	0.00
race conditions	0.00	0.00	0.00	0.00
memory related errors	0.00	0.00	0.00	0.09
undefined UPC operations	0.16	0.21	0.16	0.21
warnings	0.00	0.00	0.00	1.84
average of the above scores	**0.28**	**0.48**	**0.34**	**0.73**

Table I
AVERAGE OF TEST SCORES FOR EACH ERROR CATEGORY AND THE AVERAGE SCORE OVER ALL ERROR CATEGORIES FOR EACH UPC COMPILER.

Table I that all the compilers achieved an average score of nearly 3.0 in the "explicitly disallowed statements" category.

For an error message to be useful it should receive at least a score of 3. Table II presents the number of tests in each error category for which an useful error message was issued. Together Tables I and II show that not only the quality of the error messages issued is low but also that the UPC compilers tested are not able to detect many of the errors.

The authors consider that all tests from the "explicitly disallowed statements" category should receive at least a score of 3. Table III shows in detail how the compilers scored on these tests including the number of tests that received a score of at least 3. Notice that many of these errors were not recognized by UPC compilers. The total number of tests in this category is 164.

VI. CONCLUSION

The ability of system software to detect compile-time errors and issue messages that help programmers quickly fix these errors is an important productivity criterion for developing and maintaining application programs. To evaluate this capability for Unified Parallel C (UPC), 3141 compile-time error tests and a UPC-CompilerCheck tool have been developed. For each error message issued, UPC-CompilerCheck assigns a score from 0 to 5 based on the usefulness of the information in the message to help a programmer quickly fix the error. If no error message is issued the test gets score of 0. UPC-CompilerCheck calculates average scores over each error category and then prints the results. All tests and UPC-CompilerCheck are freely available [7].

Error Category	Number of tests	Berkeley	Cray	GNU	HP
explicitly disallowed statements	164	133	103	149	98
out-of-bounds shared memory access using indices	462	0	154	0	196
out-of-bounds shared memory access using pointers	169	0	0	0	14
out-of-bounds shared memory access in UPC function calls	324	0	0	0	0
argument errors in UPC functions	284	0	5	5	6
wrong order of UPC statements and function calls	158	0	6	0	4
uninitialized variables	35	0	10	0	20
deadlocks	18	0	0	0	0
race conditions	785	0	0	0	0
memory related errors	644	0	0	0	14
undefined UPC operations	19	1	1	1	1
warnings	79	0	0	0	29

Table II
NUMBER OF TESTS IN EACH CATEGORY WHICH RECEIVED A SCORE OF AT LEAST 3

Score	Berkeley	Cray	GNU	HP
0	31	61	15	66
1	0	0	0	0
2	0	0	0	0
3	90	7	106	5
4	6	50	6	51
5	37	46	37	42
3-5	**133 (81.1%)**	**103 (62.8%)**	**149 (90.1%)**	**98 (59.8%)**

Table III
NUMBER OF TESTS OUT OF 164 RECEIVING THE INDICATED SCORE FOR THE "EXPLICITLY DISALLOWED STATEMENTS" CATEGORY.

The Berkeley, Cray, GNU and HP UPC compilers have been evaluated and results posted on this same web site. Error detection capabilities for these compilers were generally poor including the error category "explicitly disallowed statements" where the UPC compilers should have detected all these errors.

It is hoped that these tests and recommended error messages will be used by vendors to evaluate and improve the compile-time error detection capabilities of their UPC compilers. We also hope that these tests will be used by high performance computing centers as part of their procurement process to reward vendors whose UPC implementations provide excellent compile-time (and run-time) error detection and issue high quality messages.

ACKNOWLEDGMENT

This work was supported by the United States Department of Defense and used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.

REFERENCES

[1] T. El-Ghazawi, W. Carlson, T. Sterling, and K. Yelick, *UPC: Distributed Shared Memory Programming*. Wiley-Interscience, 2003.

[2] "Unified parallel C (upc wiki)," last accessed April 27, 2012. [Online]. Available: http://upc.wikinet.org

[3] "The Berkeley Unified Parallel C," last accessed April 27, 2012. [Online]. Available: http://upc.lbl.gov/

[4] G. R. Luecke, J. Coyle, J. Hoekstra, M. Kraeva, Y. Xu, E. Kleiman, and O. Weiss, "Evaluating error detection capabilities of UPC run-time systems," in *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*, ser. PGAS '09. New York, NY, USA: ACM, 2009, pp. 7:1–7:4. [Online]. Available: http://doi.acm.org/10.1145/1809961.1809971

[5] J. Coyle, I. Roy, M. Kraeva, and G. R. Luecke, "UPC-CHECK: A scalable tool for detecting run-time errors in Unified Parallel C," in *Proceedings of International Supercomputing Conference (ICS)*, June 2012, to appear.

[6] P. Pirkelbauer, C. Liao, T. Panas, and D. Quinlan, "Runtime detection of c-style errors in upc code," in *Proceedings of Fifth Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '11, 2011. [Online]. Available: http://pgas11.rice.edu/papers/PirkelbauerEtAl-UPC-Error-Detect-PGAS11.pdf

[7] G. R. Luecke, J. Coyle, J. Hoekstra, M. Kraeva, E. Kleiman, and I. Roy, "Compile time error detection test suite and results for upc." [Online]. Available: http://hpcgroup.public.iastate.edu/CTED/UPC

[8] "The High Performance Computing Laboratory, The George Washington University," last accessed April 27, 2012. [Online]. Available: http://upc.gwu.edu

[9] "UPC projects at Michigan Technological University," last accessed April 27, 2012. [Online]. Available: http://www.upc.mtu.edu/

[10] "High Performance Computing and Simulation Laboratory, University of Florida," last accessed April 27, 2012. [Online]. Available: http://www.hcs.ufl.edu/upc/

[11] I. Roy, G. R. Luecke, J. Coyle, and M. Kraeva, "An optimal deadlock detection algorithm for Unified Parallel C," preprint (2012). [Online]. Available: http://hpcgroup.public.iastate.edu/papers/Deadlock_Dectection_for_UPC.pdf

[12] A. Ebnenasir, "UPC-SPIN: A Framework for the Model Checking of UPC Programs," in *Proceedings of Fifth Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '11, 2011. [Online]. Available: http://pgas11.rice.edu/papers/Ebnenasir-UPC-Model-Checking-PGAS11.pdf

An Integrated Scientific Experiment Framework for Numerical Analysis in e-Science Environment

Sookyoung Park*, Hyejeong Kang*, Yoonhee Kim*, Chongam Kim†, Yunjung Hyun‡
*Dept. of Computer Science, Sookmyung Women's University, Seoul, Korea
Email: {blue, hjkang, yulan}@sookmyung.ac.kr
†School of Mechanical and Aerospace Engineering, Seoul National University, Seoul, Korea
Email: chongam@snu.ac.kr
‡Korea Environment Institute, Seoul, Korea
Email: yjhyun@kei.re.kr

***Abstract*—The analytical experiments for numerical analysis lead a sequence of complex scientific computations composing of numerical equations and require enormous computing resources with appropriate management tools. Currently most studies on e-Science environments for numerical studies focus on solving specific problems to drag out the best performance of matters and have less interest in providing a uniform framework to apply for diverse numerical domains, especially for fluid dynamics. This paper presents an integrated e-Science experiment framework which could be easily applicable to solve various numerical analyses in fluid dynamics. As a proof-of-concept, an integrated e-Science framework with diverse numerical analyses has been designed and implemented over UNICORE that runs over grid computing environment.**

***Keywords*-e-Science; PSE; scientific numerical analysis; UNICORE**

I. Introduction

The experimentation in numerical analysis needs highly efficient and enormous computational resources because these experiments are composed of computations of complicated numerical equations and computation-intensive operations. Many studies and developments have been proposed to support the various scientific computational applications in e-Science environments. But, already developed frameworks support e-Science environment only for analyzing specific computational model [1][2][3][4][5]. By constructing research environments which is defined per-application in specific e-Science environment, it is difficult to conduct research efficiently and access heterogeneous resource in absence of common interfaces even though they have similar processes. Also, as proceeding e-Science developments, common interfaces, and integrated environments are required to provide various analyze techniques to research and reuse it with demands of other applications requirements. Our research has focused on the integrated scientific experimental environment for numerical analysis, especially in fluid dynamics.

UNICORE (Uniform Interface to COmputing REsources) [6] has been developed for an integrated common environment. However, adding specific applications to UNICORE is difficult to general scientists or researchers because they have to develop interfaces for each their experiments using GridBean [7].

On developing a Problem Solving Environment (PSE) for numerical applications in fluid dynamics, we are principally concerned with the following requirements:

- Support research execution in diverse computing environment: it can be executed in personal computer or grid environments.
- Support an independent experimental environment on each research domain.
- Support pre-process for generating input files of numerical analysis.
- Support post-process for visualizing analyzed results.

In this paper, we describe a design and implementation of the PSE that aims to provide a numerical study environment that requires seamless access into grid resources through a common interface for diverse domains in fluid dynamics. To support scientific computational application, we developed a common and integrated e-Science environment based on UNICORE Rich Client supporting a user interface to each application using GridBean. In a precedent study [8], we implemented just one application in a domain, i.e. 1D Euler equation in compressible flows. It is hard to say as an integrated framework. So, in this work, we expand our target range of domains as compressible flows, turbulent flows and multiphase flows in fluid dynamics.

The rest of the paper is organized as follows. In Section 2, related works are reviewed. In Section 3, an execution scenario for numerical studies and design of PSE framework architecture are described. The implementation of PSE framework is presented in Section 4. Finally, conclusion and future work are stated in Section 5.

II. Related Work

In this section we shall briefly present some approaches that are representative for the areas of grid computing environment and e-Science projects which are based on Eclipse.

A variety of scientific computational applications for numerical study have been developed, like FLOWGRID [5], and they have been executed on grid environment with functions of managing and monitoring computational jobs as well as supporting simulations. FLOWGRID is a grid application for solving Computational Fluid Dynamics (CFD) problems and FlowServe middleware [9] is provided for interactions on different interfaces. FLOWGRID is carried out targeting the specialized engineers only and limited in terms of extensibility.

One of the most well-known Eclipse-based e-Science environments is g-Eclipse [10], which is an integrated workbench framework to access the power of existing grid infrastructures. The g-Eclipse supports connectors between a user environment and many different grid middlewares (such as gLite, UNICORE, Globus toolkit) and provides tools to customize users experimental environments. However, g-Eclipse included as a plugin to existing eclipse IDE (Integrated Development Environment), is uncomfortable with complicated interfaces because there exists the irrelevance of the functions to users experimentations.

UNICORE [6] is a Grid computing technology providing grid software that combines resources of supercomputer centers and makes them available through the Internet. Within the UNICORE environment the user has a convenient way of using distributed computing resources without having to learn site or system specifics by a seamless way.

Also the UNICORE supports various clients for jobs creation, submission, and monitoring, both graphical and command-line oriented. The graphical UNICORE Rich Client referred to as URC used in our framework offers graphical editors for setting up job descriptions. Instead of editing text-based job descriptions, the user is provided high level interfaces which are tailored to the applications what he wants to execute on remote systems. And these graphical interfaces are developed using GridBean which provides pluggable interfaces.

Our framework is specifically supported in the UNICORE Rich client (URC) as adding scientific domain-specific plugins. Besides, the advantage of adopting open standards in UNICORE 6 allows for the seamless use of UNICORE components by other technologies.

III. Design of Integrated Scientific Experiment Framework

A. Framework Scenario for numerical study

A numerical analysis environment for target domains such as compressible flows, turbulent flows and multiphase flows require pre-process, simulation process, and post-process. Figure 1 shows a scenario of using our PSE framework. At first, the user installs our PSE program on PC and then selects a method of numerical studies to be performed. In pre-process step, the user can generate or upload input files that are proper for the selected numerical study. When it is finished, in simulation process step, the user configures solving conditions such as experiment method. The user selects resource to submit the job and then a JSDL (Job Submission Description Language) file which specified by GridBean model about a job definition and attributes will be submitted and executed at the previously selected resource. If the job execution is finished, the user can check whether the execution is finished, and then he can download output files. As a post-process step, graphical view is also offered for output files which require visualization.

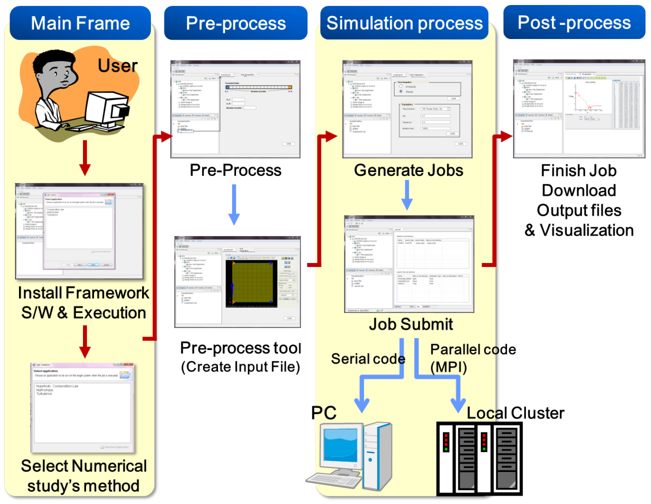

Main Frame
Pre-process
Simulation process
Post -process
User
Install Framework S/W & Execution
Select Numerical study's method
Pre-Process
Pre-process tool (Create Input File)
Generate Jobs
Job Submit
Serial code
Parallel code (MPI)
PC
Local Cluster
Finish Job Download Output files & Visualization

Figure 1. Scenario for numerical study

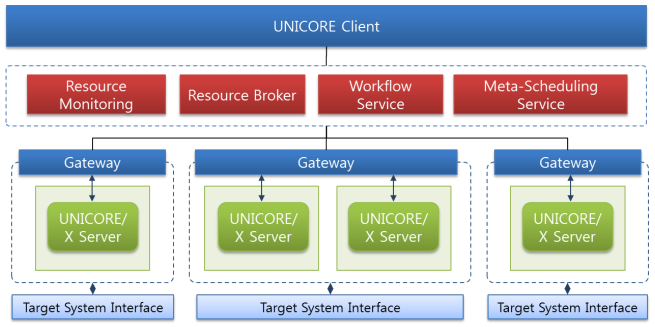


Figure 2. Framework Architecture

B. Integrated Scientific Experiment Framework Architecture

The overall architecture of the framework based on UNICORE is depicted in Figure 2. If the user submits a job or a workflow for specific application using PSE framework interfaces based on URC, Resource Broker receives the requests from UNICORE Client and collects the necessary information (such as application name, resource information; number of nodes, number of cores) through Resource Monitoring to choose the set of acceptable machines. If well suited resource (s) is selected, the job with user information is authenticated by Gateway and forwarded to web services

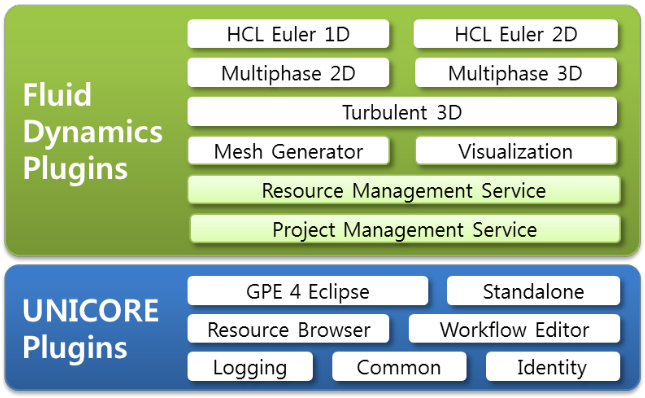


Figure 3. Client Architecture

interfaces offered by UNICORE/X server and executes it on Target System Interface. The workflow job is supported by Workflow Service module that interacts with Resource Broker and Meta-Scheduling Service. The Meta-Scheduling Service collects information across all pre-selected resources about availability, user policies, and cost parameters.

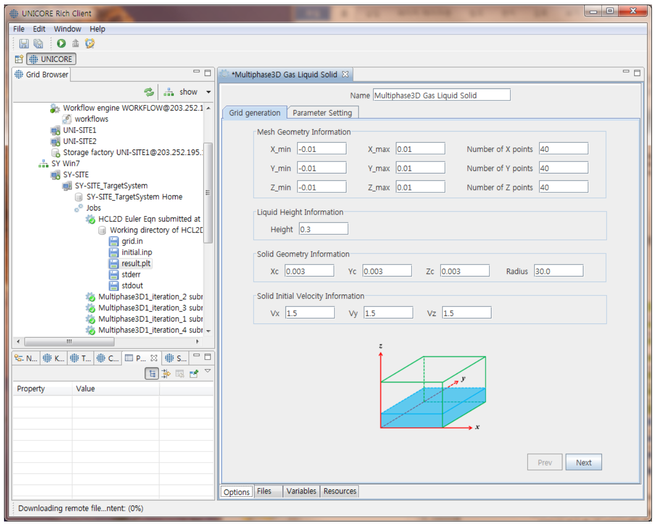
Figure 3 shows entire plugins architecture in client-side, it is made up of two layers, the one is basic UNICORE plugins that constitute the core of the URC and the other is Fluid Dynamics plugins that consist of applications for three research domains. Descriptions of basic UNICORE plugins can be found in other publication [11].

The following list contains short descriptions of Fluid Dynamics Plugins about what each plugin does:

- HCL Euler 1D / HCL Euler 2D: Plugins for the numerical study of compressible flows.
- Multiphase 2D / Multiphase 3D: Plugins for the numerical study of multiphase flows.
- Turbulent 3D: A plugin for the numerical study of turbulent flows.
- Mesh Generator: A plugin for generating input files and setting up boundary conditions of pre-processing that may be used for the applications.
- Visualization: A plugin for post-process to visualize the experiment results.
- Resource Management Service: A module to add job properties for specific application and provides filters that find resources with certain types or attributes.
- Project Management Service: A module to management history of experiments such as execution information, simulation results, errors, log files and user descriptions.

IV. Implementation

To implement user interfaces of PSE in three other domains, we developed applications specific plugins per each application using GridBean that is presented above Section 3. Within these plugins, components are organized


Name Multiphase3D Gas Liquid Solid

Mesh Geometry Information
X_min -0.01 X_max 0.01 Number of X points 40
Y_min -0.01 Y_max 0.01 Number of Y points 40
Z_min -0.01 Z_max 0.01 Number of Z points 40

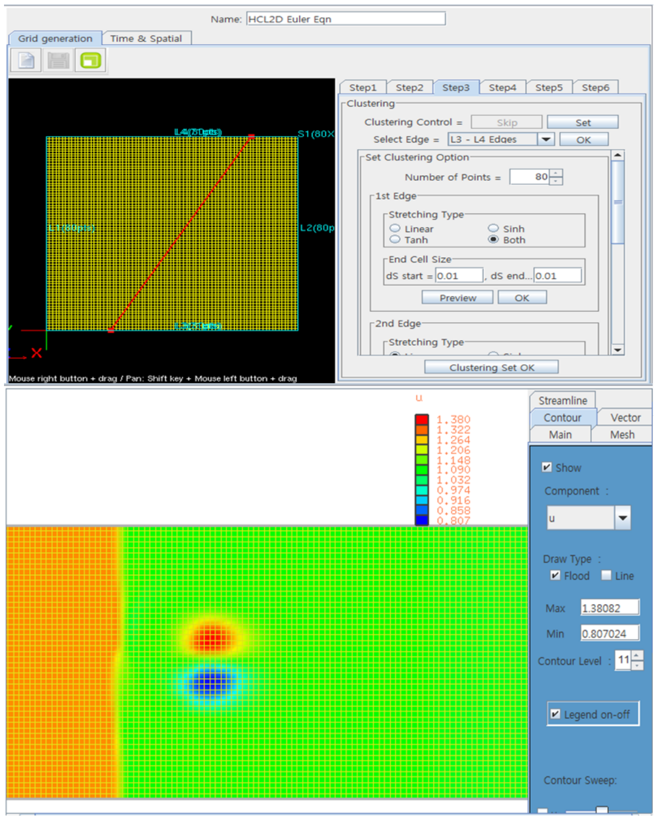
Liquid Height Information
Height 0.3

Solid Geometry Information
Xc 0.003 Yc 0.003 Zc 0.003 Radius 30.0

Solid Initial Velocity Information
Vx 1.5 Vy 1.5 Vz 1.5

Downloading remote file...ntent: (0%)

Figure 4. An Integrated Scientific Experiment Tool


Name: HCL2D Euler Eqn

Clustering
Clustering Control = Skip Set
Select Edge = L3 - L4 Edges OK
Set Clustering Option
Number of Points = 80
1st Edge
Stretching Type: Linear / Tanh / Sinh / [✓ Both]
End Cell Size
dS start = 0.01, dS end... 0.01
2nd Edge

Mouse right button + drag / Pan: Shift key + Mouse left button + drag

Show: ✓
Component: u
Draw Type: [✓ Flood] Line
Max 1.38082
Min 0.807024
Contour Level: 11
Legend on-off: ✓

Figure 5. A Mesh Generator and a Visualization Tool

dynamically by providing functions to control events of the experiment. For a specific example of user interfaces, to execute the Multiphase 3D numerical study of the multiphase flows is shown in Figure 4. Figure 5 depicts a mesh generator for pre-processing and a visualization tool for post-process in HCL 2D Euler numerical study. Moreover, with the aim of an integrated workbench framework, applications in three other domains can be accessed through a uniform manner within a PSE.

Basically user interfaces of the framework consist of one or more plugin modules and one data model depends on each of numerical studies method. But, there are a lot of methods and parameters which need to experiment.

For users convenience, we developed interface to be organized dynamically by controlling event about the experiment method selection. To dynamically configure parameters when experimental methods are selected in each application, we classified parameters as commonly applied ones and additional ones instead of defining a set of parameters in each experimental method. Defining parameter sets like this, makes management of parameters more efficient and accurate.

In case of middleware, we used most of UNICORE, and we modified IDB (Incarnation DB) which saved applications information such as name, version, location of an executable file, and parameters required for the job execution. Also, in order to provide resources among authorized users only, we generated authorization keys for each application and its information is managed through XUUDB.

V. Conclusion and Future Work

In this paper, we presented a design of the integrated scientific experiment framework supporting numerical analysis in fluid dynamics within e-Science environments. The framework adopting three numerical analysis applications is implemented as a proof-of-concept within the Eclipse-based UNICORE Rich Client. The implementation contains a mechanism to define sequences of simulations and the framework can conveniently add other numerical analysis models. A common execution environment to support various applications that are in diverse domains is important. Hence, this framework is expected to improve their experiments efficiency, convenience and reusability of technologies.

In the future, we will expand our framework to support more complex experiments and more diverse numerical study methods. Also we plan to develop a resource selection algorithm reflecting requirements of application and properties of resources.

Acknowledgment

This work was supported by the National Institute for Mathematical Sciences (NIMS) grant funded by the Korean government (No. A21101).

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (#2012-0002572).

References

[1] Jung-hyun Cho, Byung Sang Kim, Eunhye Song, Yoonhee Kim, Chongam Kim, and Min Joong JEONG, “e-AIRS : An Integrated Aerospace Portal for Collaborative Experiments and Dynamic Parametric Studies”, Proc. of the KIISE Korea Computer Congress 2006, Vol. 33, No. 2 (A), pp. 552-556, 2006 (in Korean)

[2] Jung-hyun Cho, Cinyoung Hur, Yoonhee Kim, Chongam Kim, and Kum Won Cho, “CFD Research System for e-Science based Cyber Education”, Korean Network Operations and Management (KNOM) Review, Vol. 12, No. 1, pp. 42 50, June, 2009 (in Korean)

[3] Chemomentum Project, http://www.chemomentum.org/ [retrieved: 5, 2012]

[4] Bioclipse Project, http://www.biomedcentral.com/ [retrieved: 5, 2012]

[5] FLOWGRID Project, http://www.unizar.es/flowgrid/ [retrieved: 5, 2012]

[6] UNICORE, http://www.unicore.eu/ [retrieved: 5, 2012]

[7] Sandra Bergmann. GridBean Developers Guide. UNICORE, 2009.

[8] Hyejeong Kang, Kyoung-A Yoon, Seoyoung Kim, Yoonhee Kim, and Chongam Kim, “An e-Science problem solving environment for scientific numerical study”, International Conference on Advanced Communication Technology, ICACT, Gangwon-Do, Korea (South), pp. 266-269., Feb, 2011

[9] FlowServe middleware, http://www.unizar.es/flowgrid/middleware.htm [retrieved: 5, 2012]

[10] g-Eclipse, http://www.geclipse.org/ [retrieved: 5, 2012]

[11] Bastian Demuth, Bernd Schuller, Sonja Holl, Jason Daivandy, Andre Giesler, and Valentina Huber, “The UNICORE Rich Client: Facilitating the Automated Execution of Scientific Workflows”, Proceedings of 6th IEEE International Conference on e-Science (e-Science 2010), pp. 238-245, IEEE Computer Society Press, 2010.

Minimally Invasive Interpreter Construction
– How to reuse a compiler to build an interpreter –

Christoph Schinko
Institut für ComputerGraphik
und WissensVisualisierung (CGV)
Technische Universität Graz, Austria

c.schinko@cgv.tugraz.at

Torsten Ullrich[1], Dieter W. Fellner[1,2]
[1] *Fraunhofer Austria, Graz, Austria*
[2] *Fraunhofer IGD & TU Darmstadt, Germany*

torsten.ullrich@fraunhofer.at
d.fellner@igd.fraunhofer.de

***Abstract*—Scripting languages are easy to use and very popular in various contexts. Their simplicity reduces a user's threshold of inhibitions to start programming – especially, if the user is not a computer science expert. As a consequence, our generative modeling framework *Euclides* for non-expert users is based on a JavaScript dialect. It consists of a JavaScript compiler including a front-end (lexer, parser, etc.) and back-ends for several platforms. In order to reduce our users' development times and for fast feedback, we integrated an interactive interpreter based on the already existing compiler. Instead of writing large proportions of new code, whose behavior has to be consistent with the already existing compiler, we used a minimally invasive solution, which allows us to reuse most parts of the compiler's front- and back-end.**

***Keywords*-JavaScript; generative modeling; procedural modeling; compiler; interpreter**

I. Introduction

As John Ousterhout has written in *Scripting: Higher Level Programming for the 21st Century* [1], "Scripting languages such as Perl and Tcl represent a very different style of programming than system programming languages such as C or Java. Scripting languages are designed for 'gluing' applications; they use typeless approaches to achieve a higher level of programming and more rapid application development than system programming languages. Increases in computer speed and changes in the application mix are making scripting languages more and more important for applications of the future."

Therefore, scripting languages are not only a common way to automate repeated tasks, but also a relevant tool in algorithm design – gluing existing algorithms and data structures to new solutions.

As pointed out by Ousterhout [1] conventional system programming languages are too 'rigid' for many tasks in contrast to scripting languages, whose flexibility has to be paid by performance.

In order to trade off both, we combined ahead-of-time compilation techniques with just-in-time compilation methods to an interactive interpreter. The result is in interactive environment, in which algorithms can be designed, tested, etc., and whose consistent data structures can be exported and compiled to an application at any time. In this way, we combine the advantages of both worlds.

The field of application as well as the context of this work is presented in Section "II. Related Work". A description of the used compiler has already been published [2], [3] and is summarized in Section "III. Compiler Construction". Based on this compiler, Section "IV. The Interpreter as a Retrofitted Compiler" illustrates the needed extensions to implement an interpreter.

II. Related Work

Originally, scripting languages like JavaScript were designed for a special purpose, e.g., to be used for client-side scripting in a web browser. Nowadays, the applications of scripting languages are manifold. JavaScript, for example, is used to animate 2D and 3D graphics in VRML [4] and X3D [5] files. It checks user forms in PDF files [6], controls game engines [7], configures applications, and performs many more tasks.

A. Field of Application

Scripting geometric objects – also known as generative and procedural modeling – has gained attention within the last few years [8]. The main advantage of generative modeling techniques is the included expert knowledge within an object description. For example, classification schemes used in architecture, archaeology, civil engineering, etc. can be mapped to procedures. In combination with documentation and annotation techniques established in software engineering, 3D objects are easily identifiable by digital library services (indexing, markup and retrieval) on a textual basis.

From a historical point of view, the first procedural modeling systems were Lindenmayer systems [9], or L-systems for short. These early systems, based on grammars, provided the means for modeling plants. The idea behind it is to start with simple strings and create more complex strings by using a set of string rewriting rules.

Later on, L-systems are used in combination with shape grammars to model cities [10]. Parish and Müller presented a system that generates a street map including geometry for buildings given a number of image maps as input. The resulting framework is known as *CityEngine* – a modeling environment for *CGA Shape*.

Havemann takes a different approach to generative modeling. He proposes a stack based language called *Generative Modeling Language* (GML) [11]. The postfix notation of the language is very similar to that of *Adobe Postscript*.

B. Programming Languages and Paradigms

Generative modeling inherits methodologies of 3D modeling and programming, which leads to drawbacks in usability and productivity. The need to learn and use a programming language is a significant inhibition threshold especially for archaeologists, cultural heritage experts, etc., who are seldom experts in computer science and programming. The choice of the scripting language has a huge influence on how easy it is to get along with procedural modeling.

Processing is a good example of how an interactive, easy to use, yet powerful, development environment can open up new user groups. It has been initially created to serve as a software sketchbook and to teach students fundamentals of computer programming. It quickly developed into a tool that is used for creating visual arts [12]. Processing is basically a Java-like interpreter offering new graphics and utility functions together with some usability simplifications.

Offering an easy access to programming languages that are difficult to approach directly reduces the inhibition threshold dramatically. Especially in non-computer science contexts, easy-to-use scripting languages are more preferable than complex programming paradigms that need profound knowledge of computer science. This is why we use JavaScript – a beginner friendly, structured language.

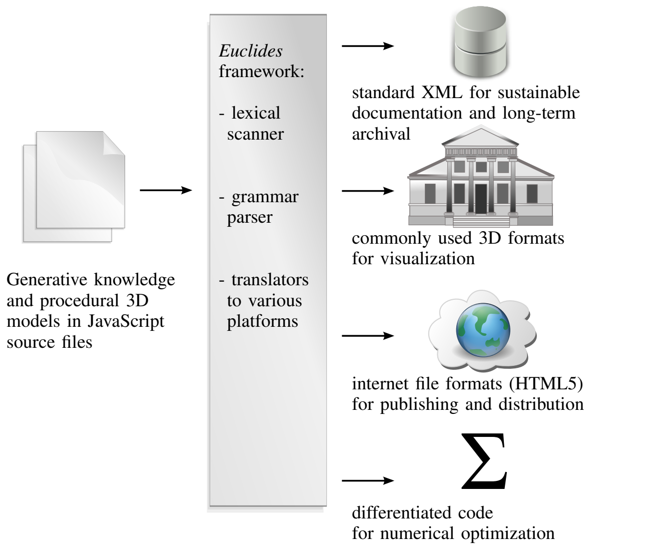
The success of Processing is based on two factors: the simplicity of the programming language on the one hand and the interactive experience on the other hand. The instant feedback of scripting environments allow the user to program via "trial and error". In order to offer our users this kind of experience, we enhanced our already existing compiler to an interactive environment for rapid application development.

C. Euclides – a JavaScript platform for Cultural Heritage

In the context of Cultural Heritage, the Generative-Modeling-Language (GML) is an established procedural modeling environment designed for expert users [13]. The aim of the *Euclides* modeling framework [14] is to offer an easy-to-use approach to facilitate these platforms. The translation mechanism for GML within *Euclides* has already been described in "Euclides – A JavaScript to PostScript Translator" and presented at the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking [2].

III. Compiler Construction

This section focuses on the existing compilation pipeline of the Euclides framework. The framework consists of several stages to translate JavaScript code to a number of target languages. Most parts are implemented in Java apart from the parser which is generated using a third-party tool (see Figure 1).



Generative knowledge and procedural 3D models in JavaScript source files
Euclides framework:
- lexical scanner
- grammar parser
- translators to various platforms
standard XML for sustainable documentation and long-term archival
commonly used 3D formats for visualization
internet file formats (HTML5) for publishing and distribution
differentiated code for numerical optimization

Figure 1. The meta-modeler approach of the *Euclides* framework has many advantages. In contrast to script-based interpreters, *Euclides* parses and analyzes the input source files, builds up an abstract syntax tree (AST), and translates it to the desired platform. Its platform and target independence as well as various exporters for different purposes are the main characteristics of *Euclides*. This innovative meta-modeler concept allows a user to export generative models to other platforms without losing its main feature – the procedural paradigm.

An editor component feeds the first stage of the framework: lexer and parser. For semantic recognition of the input source code, JavaScript syntax needs to be analyzed. All rules, which define valid JavaScript code, form its grammar. For each language construct available in JavaScript, this set of rules is validating syntactic correctness. At the same time actions within these rules create the intermediate structure that represents the input source code – a so-called abstract syntax tree (AST).

The resulting AST is the main data structure for the next stage: semantic analysis. Once all statements and expressions of the input source code are collected in the AST, a tree walker analyzes their semantic relationships, i.e., errors and warnings are generated, for instance, when they are used but not defined, or defined but not used.

Having performed all compile-time checks, a translator uses the AST to generate platform-specific files; e.g., java source code for the JVM platform. In other words, this task involves complete and accurate mapping of JavaScript code to constructs of the target language. A translation in the target language needs to be available for each statement or expression found in the AST. Usually, a direct mapping to data types or operators in the target language is not possible.

Therefore, auxiliary methods and data structures within the target language are needed to mimic JavaScript behavior.

A. Parser

The parser for JavaScript is written using ANother Tool for Language Recognition (ANTLR) [15]. ANTLR provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions. It relies on a strategy called *LL(*)* parsing, which extends the *LL(k)* parsing strategy with lookahead of arbitrary length. Using this framework, lexer and parser are generated to syntactically check the provided input for JavaScript compliance.

```
public interface ASTFactory {

   public static interface Tree {
      // tree traversal methods; e.g.
      public Tree getUp();
      public Tree[] getDown();
   }

   public static interface
      Expression extends Tree {
      // validation
      public void validate(
         ErrorHandler errorHandler);
   }

   public static interface
      Statement extends Tree {
      // validation
      public void validate(
         ErrorHandler errorHandler);
      // original source code ref.
      public int getLine();
      public String getFileName();
   }

   public static interface
      TryCatchBlock {
      // This pure markup interface
      // is used to ensure type
      // compatibility.
   }

   // the factory methods; e.g.
   public Statement statementTry(
      String filename, int line,
      Scope scope, Statement statement,
      TryCatchBlock catchBlock,
      TryFinallyBlock finallyBlock);

   // the factory utility methods
   // to create optional terms; e.g.
   public TryCatchBlock utilTryCatchBlock(
      Expression identifier,
      Statement statement);
}
```

Source code 1. This source code excerpt shows the main components of the abstract AST factory used by the Euclides parser to build up an abstract syntax tree.

A first step is to convert a sequence of characters into a sequence of tokens, which is done by special grammar rules forming the lexical analysis. For instance, only a limited number of characters is allowed for an identifier: all characters A-Z, a-z, digits and the underscore are allowed with the condition that an identifier must not begin with a digit or an underscore. These lexer rules are embedded in another set of rules – the parser rules. They are analyzing the resulting sequence of tokens to determine their grammatical structure. The complete grammar consists of a hierarchical structure of rules for analyzing all possible statements and expressions that can be formed in JavaScript, thus forming the syntactic analysis. Rules can be enriched with so-called actions. These actions create the intermediate AST structure.

Within these actions, an abstract factory, like described in [16], called `ASTFactory` is used to create necessary instances of statements and expressions for the AST. An excerpt of the abstract factory including selected inner interfaces is listed in Source Code 1.

The statements and expressions mentioned in the `ASTFactory` are defined as static, inner interfaces `Statement` and `Expression` within the definition of the factory. Both interfaces extend a common interface called `Tree`. The use of a factory has the advantage to be able to replace their implementations without touching the grammar. Additionally, markup interfaces are used to ensure type compatibility, because during AST construction, sub-parts of the AST are created bottom-up via utility methods. These parts are collected and passed to the corresponding parent rule. For example, the AST of the listing in Source Code 2 is created via the following factory calls.

```
try {
   doSomething();
} catch (exception) {
   repairSomething();
   print("caught exception " + exception);
}
```

Source code 2. The catch-block of a JavaScript try-statement automatically declares and defines a variable. In this example it is called `exception`.

The optional catch-block is parsed by a sub-rule with actions, which call the factory method `utilTryCatchBlock`. This method returns an instance of the markup interface `TryCatchBlock`, which can only be passed to a `statementTry` method. This method itself is called in the corresponding rule to match a try-statement. In this way, complex grammar rules are split up into several simpler rules while using the abstract factory pattern and maintaining type safety.

The signature of the `statementTry` call reveals some properties that are passed to the factory by all statements: the source code's file name and line together with the current scope. In case of `statementTry`, the statement to try, the optional catch-block as well as the optional finally-block are also passed to the factory. (Please note, at least one optional block must be non-null.)

B. Abstract Syntax Tree

In JavaScript, the top-level rule of an AST is always a simple list of statements – no enclosing class structures, no package declaration, no inclusion instructions, etc. Each statement contains all included substatements and expressions as well as associated comments. Furthermore, our AST stores additional formatting information (number of new lines, white spaces, tabs, etc.), which offer the possibility to regenerate the original input source code just using the AST.

During the validation step, this tree structure is extended by reference and occurrence links; e.g., each method call references the method's definition and each variable definition links to all its occurrences.

```
@Override
public void forRangeNoArray( String filename, int line) {
   warning(filename, line,
      "The range expression of this for-statement is not"
      + " an array. It will be casted automatically, "
      + "which might lead to undesired results.");
}
```

Source code 3. The Euclides compiler includes a simple and limited type inference implementation. Its main purpose is to recognize common pitfalls of JavaScript source code and to present reasonable warnings.

Having assured that all compile-time checks are carried out, symbols are stored in a so called namespace. During validation, this data structure is used to detect name collisions (e.g. redefinition of variables) and undefined references (e.g. usage of undeclared variables). In addition, a simple type inference system tries to determine the variables' types. As this system is incomplete, it cannot be used for compile time optimizations (e.g. mapping to native data types), but it can be used for warnings and recommendations. To provide meaningful error messages is an important aspect with regard to language processing. In Euclides, an error handler is responsible for collecting and preparing error and warning messages. This functionality is not only used during AST construction to deal with syntactic issues, but also for semantic validation as well. A total of 52 different errors and warnings can be issued. For example, if the type inference system checks the range expression of a for-in loop, it expects an array. If it finds a different type, the warning routine listed in Source Code 3 is issued.

C. Translator to Java – the Compiler Backend

The translation backend for the target language Java is not as straightforward as the similarity in names between Java and JavaScript would suggest. Although they have some similarities, the concepts of both languages show major differences. Java is a statically typed, class-based, general-purpose programming language.

Because of the conceptual differences in the typing system, it is not only unpractical, but impossible to project all JavaScript data types onto built-in Java data types. In JavaScript, there is no difference between integer numbers and floating point numbers. Just one data type called `Number` holds any type of number. Other differences can be found when comparing the remaining data types. Also dynamic typing is not a language feature of Java – as a consequence, each JavaScript data type is re-built in Java to match its functionality.

A total of seven data types are implemented in classes having a common interface called `Var`. These data types are: `VarUndefined`, `VarBoolean`, `VarNumber`, `VarString`, `VarArray`, `VarObject`, and `VarFunction`. A number of access functions and conversion methods are available for all data types. All internal functions provide an additional parameter that always refers to a table entry, which references the corresponding JavaScript file and line number. In this way, warnings can be generated at runtime, if implicit conversion takes place. For example, the implementation of an array access includes the statement `Log.variableTypeChangeImplicit(ii);`. In the messages table (generated by the compiler) there is an entry #`ii` that provides reasonable information needed for a runtime warning.

The access functions reveal the implementation details and the internal Java data types used:

- **Boolean**: The mapped Java data type is `boolean`.
- **Number**: A JavaScript number is mapped to `double`.
- **String**: String is mapped to `String`.
- **Array**: A JavaScript array is realized using the collection `java.util.ArrayList<Var>`.
- **Object**: And an object in JavaScript is mapped to `java.util.HashMap<String, Var>`.
- **Function**: A JavaScript functor is realized in Java as a function pointer using abstract objects.

The instantiation of variables within the generated Java code is performed using factory methods like `Factory.initString(String text)`. Furthermore, all JavaScript operators need to be recreated in Java as well.

A total of 49 operators grouped in unary, binary and tertiary operators are available. Each operator is applied via a method call and can therefore be exchanged easily. These concepts are demonstrated in Source Code 4, which shows the implementation of the binary subtraction operator found in JavaScript. In case at least one of the operands is not of type number, a warning is generated. The operator returns a new number initialized with the result of the subtraction operation of the internal Java data types used.

```
public static Var SUB(int ii, Var v1, Var v2) {
   if (!v1.getType().equals(Type.NUMBER)
    || !v2.getType().equals(Type.NUMBER))
     Log.deviantOperatorCallNoNumber(ii) ;

   return Factory.initNumber(
      v1.toNumber() - v2.toNumber());
}
```

Source code 4. During the translation of JavaScript to Java, all JS-operators are mapped to corresponding Java-based static method calls, which implement their behaviour.

The factory pattern has been chosen for the generated Java code in order to easily replace the mapping of JS variables and operators to different implementations. In this way, we realized a compiler with included, automatic derivation; i.e. within the generated code we can evaluate both: a function $f(x_1, \dots, x_k)$ as well as its partial derivatives $\frac{df}{dx_i}$. This technique offers the possibility to use standard optimization algorithms to solve numerical optimization problems [17].

All variables defined in the JavaScript source code are collected in the namespace. The Java translator backend, however, distinguishes between variables defined in global scope and local variables. A single class called `Variable` is created holding global variables as static objects. All other variables, e.g. those defined in a function, are exported in-place. Functions itself are mapped to Java functions and are collected in a class called `Function`.

All expressions are exported in their respective embedding statement. A distinction between global and local scope is made in case of the statements. All locally defined statements, e.g., statements defined within a function, are exported in-place. Global statements are collected in a class called `Main` and are executed from the Java main method.

IV. The Interpreter as a Retrofitted Compiler

As stated before, the simplicity of a programming language is only one factor of a successful development environment. Reasonable feedback and an interactive experience are also important. In order to offer our users this kind of experience, we enhanced our already existing compiler to an interpreter. A similar approach to combine interpretation and compilation has been presented by Anton Ertl and David Gregg [18], but in contrast to our system, they start with an interpreter and end up with a compiler.

A. *Compilers and Interpreters*

Unfortunately, there is no commonly accepted definition of the terms “compiler” and “interpreter”. The problem is the smooth transition between compilation and interpretation techniques, which blur a clear distinction. On the one hand many interpreters have integrated just-in-time compilers, on the other hand, some compilers rely on an interpreter integrated into each compiled unit. In combination with virtual machines [19], which have functionality not provided by any real machine, and CPUs, which can execute source code directly [20], it is even more complicated to find a clear distinction.

In our context, we differentiate between compiler and interpreter by the number of times *our* `ASTFactory` is called per JS-application execution. If the factory is called every time, the system is called interpreter. Otherwise, it’s a compiler.

B. *Interpreter Design*

In order to design, realize, and implement an interpreter based on an abstract syntax tree [21], current software engineering approaches recommend one of two main designs: the interpreter pattern and the visitor pattern [22].

According to the interpreter pattern, each node of the AST should have a specialized version of an evaluation, respectively, interpretation method; e.g., `eval(...)`. The visitor pattern in contrast only needs some callback functionality. In this way it can separate algorithms and actions from the data structure it operates on. As the visitor pattern (in combination with an iterator pattern for tree traversal) is already used by the Euclides compiler backends, it is also used by the interpreter.

The main idea of the interpreter implementation is based on a property found in many scripting languages. In contrast to, for example, Java, in which each statement is enclosed (at minimum) by a class definition, enclosed by a file definition, the scripting language JavaScript does not have this “overhead”. As a consequence, the root node of the AST is simply a list of statements: statementA, statementB, statementC and for each statement, the list of previous statements has to be a valid program. This linguistic property allows to compile each top-level JS statement as a unit of its own – a dynamic library. While this is not sensible for regular compilations, it offers the possibility to compile instructions statement by statement. Finally, if each unit is executed directly after being compiled, the resulting backend is an interpreter. Even more, additionally included callback routines can be used for debugging purposes [23].

C. *Implementation Details*

Following the observation that even a single statement can be regarded as a unit of its own, the original JavaScript compiler is extended to reflect this property. Statements in the AST are no longer stored in a one-dimensional array, but a two-dimensional array is used instead. This way it is possible to group statements, i.e., all statements passed to the interpreter in a single evaluation call form one group and are stored in a one-dimensional array. All groups are stored in an array as well, thus as a consequence, the statements are stored in a two-dimensional array. These groups can be accessed by a new set of access functions while at the same time retain compatibility to the compiler, e.g., the command `getAllStatements()` now simply copies the two-dimensional structure in a one-dimensional one.

In addition to the changes in the AST, the namespace is also using a two-dimensional array for storing all symbols the same way the AST does. It uses the same mechanism to create units of symbols while being compatible to the old compiler version. These changes are necessary to allow tracking of interpretation history as well as to speed up all operations relying on the AST such as validation and code generation.

A small change in the runtime, not related to the interpreter redesign, was carried out in the process of implementing the changes for AST and namespace. Function pointers are now being ommited in the favor of using anonymous inner classes.

V. Conclusion

The simplicity of scripting languages reduces a user's inhibition threshold to start programming. Our generative modeling framework *Euclides* for non-expert users is based on a JavaScript dialect. It consists of a JavaScript compiler including a front-end (lexer, parser, etc.) and back-ends for several platforms.

The main contribution is an interactive interpreter based on the already existing compiler. Instead of creating large proportions of new code, whose behavior has to be consistent with the already existing compiler, we envisaged a minimally invasive solution. It allows us to reuse most parts of the compiler's front- and back-end.

Acknowledgment

We would like to thank Richard Bubel for his valuable support on ANTLR and the JS grammar. In addition, the authors gratefully acknowledge the generous support from the European Commission for the integrated project 3D-COFORM [24] under grant number FP7 ICT 231809 and from the Austrian Research Promotion Agency (FFG) for the research project METADESIGNER, grant number 820925/18236.

References

[1] J. K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century," *IEEE Computer Magazine*, vol. 31, no. 3, pp. 23–30, 1998.

[2] M. Strobl, C. Schinko, T. Ullrich, and D. W. Fellner, "Euclides – A JavaScript to PostScript Translator," *Proccedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, vol. 1, pp. 14–21, 2010.

[3] C. Schinko, M. Strobl, T. Ullrich, and D. W. Fellner, "Modeling Procedural Knowledge – a generative modeler for cultural heritage," *Selected Readings in Computer Graphics 2010*, vol. 21, pp. 107–115, 2011.

[4] D. Brutzman, "The virtual reality modeling language and Java," *Communications of the ACM*, vol. 41, no. 6, pp. 57 – 64, 1998.

[5] J. Behr, P. Dähne, Y. Jung, and S. Webel, "Beyond the Web Browser – X3D and Immersive VR," *IEEE Virtual Reality Tutorial and Workshop Proceedings*, vol. 28, pp. 5–9, 2007.

[6] F. Breuel, R. Bernd, T. Ullrich, E. Eggeling, and D. W. Fellner, "Mate in 3D – Publishing Interactive Content in PDF3D," *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, vol. 15, pp. 110–119, 2011.

[7] M. Di Benedetto, F. Ponchio, F. Ganovelli, and R. Scopigno, "SpiderGL: a JavaScript 3D graphics library for next-generation WWW," *Proceedings of the 15th International Conference on Web 3D Technology*, vol. 15, pp. 165–174, 2010.

[8] T. Ullrich, C. Schinko, and D. W. Fellner, "Procedural Modeling in Theory and Practice," *Poster Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, vol. 18, pp. 5–8, 2010.

[9] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*, P. Prusinkiewicz and A. Lindenmayer, Eds. Springer-Verlag, 1990.

[10] Y. Parish and P. Mueller, "Procedural Modeling of Cities," *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, vol. 28, pp. 301–308, 2001.

[11] S. Havemann, "Generative Mesh Modeling," *PhD-Thesis, Technische Universität Braunschweig, Germany*, vol. 1, pp. 1–303, 2005.

[12] C. Reas, B. Fry, and J. Maeda, *Processing: A Programming Handbook for Visual Designers and Artists*, C. Reas, B. Fry, and J. Maeda, Eds. The MIT Press, 2007.

[13] C. Schinko, M. Strobl, T. Ullrich, and D. W. Fellner, "Modeling Procedural Knowledge – a generative modeler for cultural heritage," *Proceedings of EUROMED 2010 - Lecture Notes on Computer Science*, vol. 6436, pp. 153–165, 2010.

[14] ——, "Scripting Technology for Generative Modeling," *International Journal On Advances in Software*, vol. 4, pp. 308–326, 2011.

[15] T. Parr, *The Definite ANTLR Reference – Building Domain-Specific Languages*, T. Parr, Ed. The Pragmatic Bookshelf, Raleigh, 2007.

[16] E. Freeman, E. Freeman, B. Bates, and K. Sierra, *Head First Design Patterns*, E. Freeman, E. Freeman, B. Bates, and K. Sierra, Eds. O'Reilly Media, Inc., 2004.

[17] T. Ullrich and D. W. Fellner, "Generative Object Definition and Semantic Recognition," *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, vol. 4, pp. 1–8, 2011.

[18] A. M. Ertl and D. Gregg, "Retargeting JIT compilers by using C-compiler generated executable code," *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, vol. 13, pp. 41–50, 2004.

[19] T. Lindholm and F. Yellin, *The Java(TM) Virtual Machine Specification*, T. Lindholm and F. Yellin, Eds. Prentice Hall, 1999.

[20] T. R. Bashkow, A. Sasson, and A. Kronfeld, "System Design of a FORTRAN Machine," *IEEE Transactions on Electronic Computers*, vol. 16, pp. 485–499, 1967.

[21] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*, T. Parr, Ed. Pragmatic Bookshelf, 2010.

[22] M. Hills, P. Klint, T. van der Strom, and J. Vinju, "A Case of Visitor versus Interpreter Pattern," *Proceedings of the International Conference on Objects, Models, Components and Patterns (TOOLS'11)*, vol. 49, pp. 1–16, 2011.

[23] J. Vraný and A. Bergel, "The Debuggable Interpreter Design Pattern," *Proceedings of the International Conference on Software and Data Technologies*, vol. 2, pp. 22–29, 2007.

[24] D. Arnold, "3D-COFORM: Tools and Expertise for 3D Collection Formation," *Proceedings of Electronic Information, the Visual Arts and Beyond*, vol. 21, pp. 94 – 99, 2009.

Implicit Nested Repetition in Dataflow for Procedural Modeling

Wolfgang Thaller, Ulrich Krispel, Sven Havemann
Institute of Computer Graphics and Knowledge Visualization
Graz University of Technology
Graz, Austria
Email: {w.thaller, u.krispel, s.havemann} @cgv.tugraz.at

Dieter W. Fellner
Fraunhofer IGD and TU Darmstadt
Darmstadt, Germany
Email: d.fellner@igd.fraunhofer.de

***Abstract*—Creating 3D content requires a lot of expert knowledge and is often a very time consuming task. Procedural modeling can simplify this process for several application domains. However, creating procedural descriptions is still a complicated task. Graph based visual programming languages can ease the creation workflow, however direct manipulation of procedural 3D content rather than of a visual program is desirable as it resembles established techniques in 3D modeling. In this paper, we present a dataflow language that features a novel approach to handling loops in the context of direct interactive manipulation of procedural 3D models and show compilation techniques to translate it to traditional languages used in procedural modeling.**

***Keywords*-procedural modeling, dataflow graphs, loops, term graphs**

I. Introduction

Conventional 3D models consist of geometric information only, whereas a procedural model is represented by the operations used to create the geometry [1]. Complex man-made shapes exhibit great regularities for a number of reasons, from functionality over manufacturability to aesthetics and style. A procedural representation is therefore commonly perceived as most appropriate, but not so many 3D artists accept a code editor as user interface for 3D modeling, and only few of them are good programmers. Recently, dataflow graph based visual programming languages for 3D modeling have emerged [2], [3]. These languages facilitate a graphical editing paradigm, thus allowing to create programs without writing code. However, such languages are not always easier to read than a textual representation [4]. Therefore, the goal is a modeler that allows direct manipulation of procedural content on the concrete 3D model, without any knowledge of the underlying representation (code), while retaining the expressiveness of dataflow graph based methods.

In this paper, we present a term graph based language for procedural modeling with features that facilitate direct manipulation. First, we give an overview of related work in Section 2. Then we give a summary of the requirements for the language in Section 3. Furthermore, in Section 4 the language is formally defined, and a compilation technique to embed such models in existing procedural modeling systems and examine error handling in the context of partial model evaluation is described. Section 5 contains examples and some benchmarks showing optimization results. The last section concludes with some points of future research.

II. Related Work

Procedural modeling is an umbrella term for procedural descriptions in computer graphics. As a procedural description is basically just a computer program, there are many possibilities to express procedural content.

One category are general purpose programming languages with geometric libraries, for example C++ with *CGAL* [5] or the Generative Modeling Language (*GML*) [1] which utilizes a language similar to Adobe's PostScript [6]. *Processing* [7] is an open source programming language based on Java with a focus on computer programming within a visual context.

As many professional 3D modeling packages contain embedded scripting languages, these can be used to express procedural content. Some representatives are for example MEL script for Autodesk Maya [8] or RhinoScript for Rhinoceros [9].

Some domain specific languages have successfully been applied to express procedural content. For example, emerging from the work of Stiny et al. [10] who applied the concept of formal grammars (string replacements) to the domain of 2D shapes, Wonka et al. [11] introduced *split grammars* for automatic generation of architecture. These concepts have further been extended by Mueller et al. [12] into CGA Shape, which is available as the commercial software package CityEngine [13] that allows procedural generation of buildings up to whole cities.

Visual Programming Languages (VPLs) allow to create and edit programs using a visual editing metaphor. Many VPLs are based on a dataflow paradigm [14]; the program is represented by a graph consisting of *nodes* (which represent operations) and *wires* along which streams of *tokens* are passed. Some examples in the context of procedural modeling are the procedural modeler Houdini [3] and the Grasshopper plugin for Rhinoceros [9], which both feature visual editors for dataflow graphs. Furthermore, the work of Patow et al. [15] has shown that shape grammars can also be represented as dataflow graphs.

Term Graphs [16] arose as a development in the field of term rewriting. While term graphs are intuitively similar to dataflow graphs, there is no concept of a stream of tokens. Term graphs are a generalization of terms and expressions which makes explicit sharing of common subexpressions possible. Formally, we base our work on the definitions given in [17] rather than on any dataflow formalism.

III. Language Requirements

Dataflow languages have a number of properties that make them very desirable for interactive procedural modeling. They allow efficient partial reevaluation in order to interactively respond to "localized" changes, they are expressive enough to cover traditional domains of procedural modeling such as compass-and-ruler constructions and split-grammars, and they can be extended in various ways to support repeated structures/repeated operations.

We are currently researching direct-manipulation based user interfaces for dataflow-based procedural modeling. This means that the dataflow graph itself is not visible to the user; instead, the user interacts with a concrete instance of the procedural model, i.e., a 3D model generated from a concrete set of parameter values. The basic usage paradigm is that the user selects objects in this 3D view and applies operations to them; these operations are added to the graph.

The goal of keeping the graph hidden during normal user interaction leads to additional requirements for the language that differ from traditional approaches.

A. Repetition

Loops should not be represented explicitly, i.e., loops should not be represented by an object that needs to be visualized so the user can interact with it directly. Operations should be implicitly repeated when they are applied to collections of objects.

It must be possible to deal with **nested repetitions** as part of this implicit repetition behaviour. Existing dataflow-based procedural modeling systems use a "stream-of-tokens" concept, i.e., a wire in the dataflow graph transports a linear stream of tokens that all get treated the same by subsequent operations. Nested structures are not preserved in this model.

When directly interacting with a 3D model, we expect the user to frequently zoom to details of the model. For example, consider a model of a building facade that consists of several stories, each of which contains several identical windows, which in turn contain several separate window panes. A user will zoom in to see a single window on their screen and then proceed to edit that archetypal window, for example by applying some operation to two neighbouring window panes of that same window. All operations in the modeling user interface should always behave consistently, independent of whether the user is editing a model consisting of just a single window, or one of many windows. In both cases, the system needs to remember that a collection of window panes belongs to a single window. Thus, flat token streams are not suited to direct-manipulation procedural modeling.

B. Failures

There are many modeling operations that do not always succeed, e.g., intersection operations between geometric objects. When applying volumetric split operations, a volume might become empty, rendering (almost) all further operations on that volume meaningless.

Often, these failures have only local effects on the model, so aborting the evaluation of the entire model is excessive; rather, we propagate errors only along the dependencies in the code graph — if its sources could not be calculated, an edge is not executed. In many cases, this is exactly the desired behaviour and allows to easily express simple conditional behaviours such as "if there is an intersection, construct this object at the intersection point" or "if there is enough space available, construct an object".

C. Side Effects

Neither dataflow graphs nor term graphs are particularly well-suited for dealing with side-effecting operations; also, to simplify analysing the code for purposes of the GUI, we have a strong motivation to forbid side effects.

However, it is a fundamental user expectation to be able to have operations that *create* objects, and to be able to *replace* or refine objects. Both Grasshopper and Houdini use side-effect free operations and rely on the user to pick one or more dataflow graph nodes whose results are to be used for the final model; this solution is not applicable to a direct manipulation procedural modeler because it would require interacting with the graph rather than with a 3D model.

IV. The Language

Below, we will first define the term graphs that form the basis of our language; we will then proceed to discuss our treatment of side effects, repetition and failing operations.

A. Code Graphs

The underlying data structure is a *hypergraph* consisting of nodes, which correspond to (intermediate) values and graphical objects, and *hyperedges*, which represent the operations applied to those values as shown in Figure 1.

Note that we are following term graph terminology here, which differs from the terminology traditionally used for dataflow graphs. In a dataflow graph, *nodes* are labelled with operations, and they are connected with edges or *wires*, which transport values or tokens. In a term graph, *hyperedges* (i.e., edges that may connect more or fewer than two nodes) are labelled with operations or literal constants, and values are stored in nodes, which are labelled with a type.

We reuse the following definition from [17]:

Definition 1: A *code graph* over an edge label set ELab and a set of types NType is defined as a tuple $G = (\mathcal{N}, \mathcal{E}, \mathsf{In}, \mathsf{Out}, \mathsf{src}, \mathsf{trg}, \mathsf{nType}, \mathsf{eLab})$ that consists of:

- a set $\mathcal{N}$ of *nodes* and a set $\mathcal{E}$ of *hyperedges* (or *edges*),

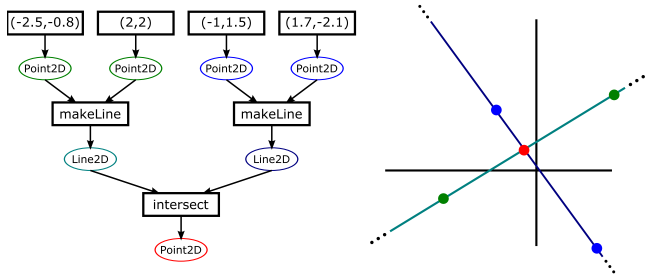


Figure 1. A **code graph** (as presented by [17]) is a hypergraph that consists of nodes that correspond to results and hyperedges that represent operations (left). In this illustration the nodes are represented as ellipses. Hyperedges are visualized as boxes; they can have any number of source and target nodes. Hyperedges with no source nodes correspond to constants. This example shows a code graph that carries out a simple construction: Two points define a straight line; two lines yield an intersection point (right).

- two node sequences $\mathsf{In}, \mathsf{Out} : \mathcal{N}^*$ containing the *input nodes* and *output nodes* of the code graph,
- two functions $\mathsf{src}, \mathsf{trg} : \mathcal{E} \to \mathcal{N}^*$ assigning each edge the sequence of its *source nodes* and *target nodes* respectively,
- a function $\mathtt{nType} : \mathcal{N} \to \mathsf{NType}$ assigning each node its *type*, and
- a function $\mathsf{eLab} : \mathcal{E} \to \mathsf{ELab}$ assigning each edge its *edge label*. □

Furthermore, we require all code graphs in our system to be acyclic and that every node occurs exactly once in either the input list of the graph, or in exactly one target list of an edge.

Definition 2: Edge labels are associated with an input type sequence and an output type sequence by the functions $\mathsf{edgeInType}$ and $\mathsf{edgeOutType} : \mathsf{ELab} \to \mathsf{NType}^*$. □

Definition 3: An edge e is considered *type-correct* if $\mathsf{edgeInType}(\mathsf{eLab}(e))$ matches the type of the edge's source nodes, and $\mathsf{edgeOutType}(\mathsf{eLab}(e))$ matches the type of its target nodes. A codegraph is type-correct if all edges are type-correct. □

B. Limited Side Effects

In Section III-C, we have noted the need to be able to model *creation* and *replacement* operations. The *scene* is the set of visible objects; we define it as a global mutable set of object references. We only allow two kinds of side-effecting operations: (a) adding a newly-created object to the scene, thus making it visible; and (b) removing a given object reference from the scene.

Replacement and refinement can be modeled by removing an existing object and adding a new one. Object removal is idempotent and only affects object visibility, not the actual object. Object visibility cannot be observed by operations. Therefore, no additional constraints on the order of execution are introduced.

C. Implicit Repetition

When an operation is applied to a list rather than a single value, it is implicitly repeated for all values in the list; if two or more lists are given, the operation is automatically applied to corresponding elements of the lists (cf. Figure 2). It is assumed that the lists have been arranged properly.

We define our method of implicitly handling repetition by defining a translation from codegraphs with implicitly-repeated operations to codegraphs with explicit loops.

1) Explicit Loops:

Definition 4: A *codegraph with explicit loops* is a codegraph where the set of possible edge labels ELab has been been extended to include *loop-boxes*. A loop-box edge label is a tuple $(\mathtt{LOOP}, G', f)$ where G' is a code graph (the loop body) with n inputs and $f \in \{0,1\}^n$ is a sequence of boolean flags, such that at least one element of f is 1. The intention behind the flags f is to indicate which inputs are lists that are iterated over ($f_i = 1$), and which inputs are non-varying values that are used by the loop ($f_i = 0$). The number of iterations corresponds to the length of the shortest input list. The edge input and output types of a loop are defined by wrapping the input and output types of the loop body (referred to as ti_i and to_i below) with $\mathtt{List}[\cdots]$ as appropriate:

$$\mathsf{edgeOutType}((G,f))_i := \mathtt{List}[to_i]$$

$$\mathsf{edgeInType}\,((G,f))_i := \begin{cases} \mathtt{List}[ti_i] & \text{if } f_i = 1 \\ ti_i & \text{otherwise} \end{cases} \qquad \square$$

2) Codegraphs with Implicit Repetition:

To allow implicit repetition, we relax the type-correctness requirement that edge input/output types match the corresponding node types.

A codegraph with implicit repetition is translated to a codegraph with explicit loops by repeatedly applying the following translation; the original codegraph is considered type-correct iff this algorithm yields a codegraph with explicit loops that fulfills the type-correctness requirement.

Consider an edge e where the type-correctness condition is violated. If any of the output nodes is not a list, or if any of the mis-matching input nodes is not a list, abort; in this case, the input codegraph is considered to be invalid. Replace the edge e by a loop edge e'. The repetition flags f_i for the new loop edge are set to 1 for every input with a type mismatch, and to 0 otherwise. The loop body G' is a codegraph containing just the edge e; the types of its input and output nodes are chosen such that the edge e' becomes type-correct within the outer codegraph. The translation is then applied to the loop body G'.

3) Fusing Loops:

The result of the above translation is a codegraph that contains separate (and possibly nested) loops for each edge. This is undesirable for two reasons, namely performance and code readability. Performance is relevant whenever the operations used in the codegraph edges are relatively cheap, such as, for example, compass and ruler constructions, as opposed to boolean operations

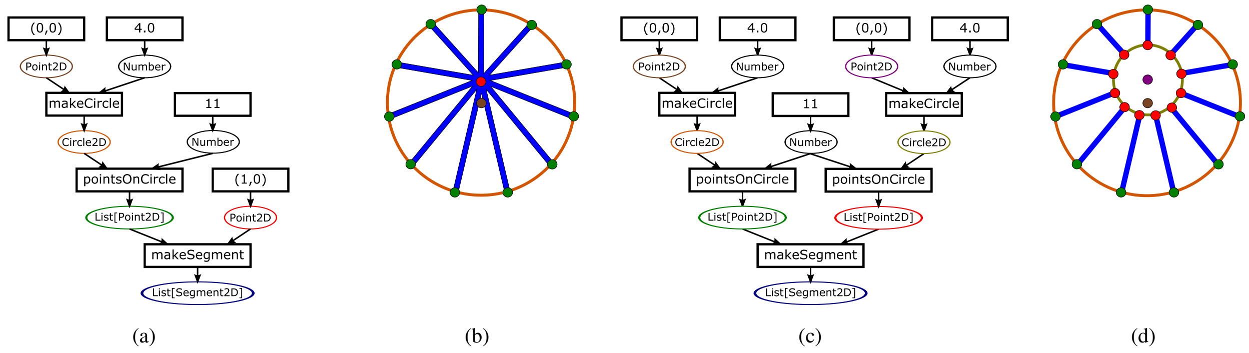


Figure 2. Handling repetitions: The images show examples of simple procedural models ((b) and (d)) that create a list of line segments (blue) and their respective code graphs ((a) and (c)). Points, lines and circles correspond to intermediate results (nodes) of the same color. makeCircle creates a circle out of a point and a radius, pointsOnCircle creates a list of evenly distributed points on a circle and makeSegment creates a straight line segment between two points. This operation can be implicitly repeated to create segments from a list of points (on a circle) to a single point ((b)), or between two lists of points on circles ((d)) using makeSegment. Multiple graphical elements are represented by single nodes in the corresponding code graphs ((a) and (c)).

on 3D volumes (constructive solid geometry, CSG). Code readability is important because a procedural model might still need to be modified after it has been exported from our system to a traditional script-based system.

Consecutive loops, i.e., loops where the second loop iterates over an output of the first, can be fused if both loops have the same number of iterations and if the second loop does not, either directly nor indirectly, depend on values from other iterations of the first loop.

To determine which loops have the same number of iterations, we will annotate each occurence of `List` in each node type with a symbolic item count, represented by a set of variable names. Each variable is an arbitrary name for an integer that is unknown at compile time. A set denotes the minimum of all the contained variables. $\texttt{List}_{\{a\}}[t]$ means a list of a items of type t, and $\texttt{List}_{\{a,b\}}[t]$ means a list of $\min(a, b)$ items.

All `List` types that appear as outputs of non-loop edges are annotated with a single unique variable name each. Every loop edge is annotated with a symbolic iteration count that is the minimum (represented by set union) of the symbolic item counts of all the lists it iterates over. Annotations on nested `List` types are propagated into and out of the loop bodies. The resulting `List` types of a loop box are annotated with a symbolic item count that is equal to the symbolic iteration count of the loop.

Two consecutive loop edges e_1 and e_2 can be *fused* when the symbolic iteration counts of the loops are equal, the repetition flag f_i is set to 1 for all inputs of e_2 that are outputs of e_1, and e_2 is not reachable from any edge that is reachable from e_1, other than e_1 and e_2 themselves.

If all these conditions are fulfilled for a given pair of edges, the edges can then be replaced by a single edge (cf. Figure 3); the fused loop body is the sequential concatenation of the two individual loop bodies. The inputs for the fused edge are the inputs of e_1 and all nodes that are inputs

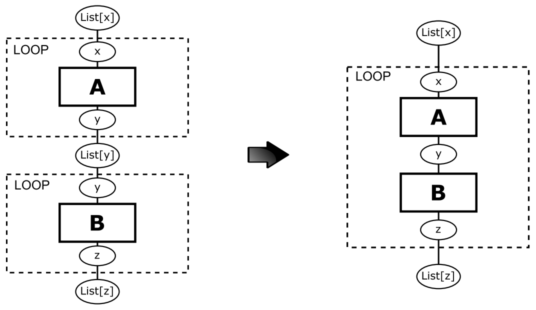


Figure 3. Two consecutive loops containing one operation each that gets applied to every item of the list. Under certain conditions (see text) the loops can be fused in order to simplify the graph.

of e_2 but not outputs of e_1. The flags f_i for the fused edge are equal to the corresponding flags for inputs of e_1 and e_2. The outputs for the fused edge are all nodes that are either outputs of e_1 or of e_2.

This fusing operation is applied until no more edges can be fused.

D. Handling Errors

The desired error-handling behaviour can be described by regarding `ERROR` as a special value which is propagated through the codegraph. If an operation fails, all its outputs are set to `ERROR`; an operation is also considered to fail whenever any of its inputs are `ERROR`.

In a naive translation, all arguments need to be explicitly checked for every single operation. To arrive at a better translation, we use a similar method as for the loops above; we first make the error checking explicit and then introduce a rule for combining consecutive error-checks.

Definition 5: $\texttt{Opt}[t] := t \cup \{\texttt{ERROR}\}$ for all types t, i.e., $\texttt{Opt}[t]$ is a type that can take any value that type t can, or a special error token. $\texttt{Opt}[t]$ is idempotent: $\texttt{Opt}[\texttt{Opt}[t]] = \texttt{Opt}[t]$. Also note that `Opt` can nest with `List` — the types

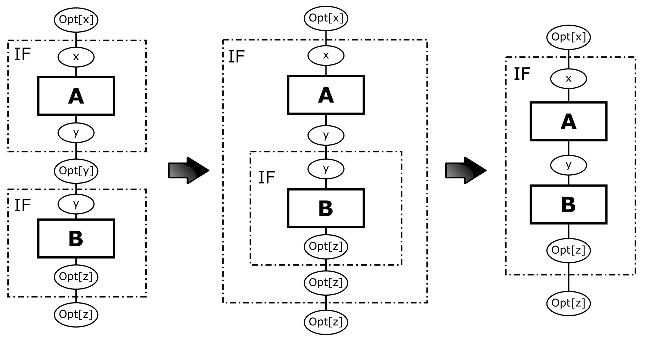


Figure 4. Left: two consecutive if-boxes used for handling potentially-failing operations. The input ($\mathtt{Opt}[x]$ at the top) is already the result of a potentially-failing operation. Note that in this example, operation **A** itself cannot fail (result type is plain y), while operation **B** can (result type is $\mathtt{Opt}[z]$). They can be combined by nesting the second box inside the first (center). This often exposes opportunities for eliminating redundant error checks (right).

$\mathtt{Opt}[\mathtt{List}[t]]$ and $\mathtt{List}[\mathtt{Opt}[t]]$ and $\mathtt{Opt}[\mathtt{List}[\mathtt{Opt}[t]]]$ are three different types. □

Definition 6: An *if-box* edge label is a tuple $(\mathtt{IF}, G', f)$ where G' is a codegraph with n inputs and $f \in \{0,1\}^n$ is a sequence of boolean flags, such that at least one element of f is 1. The edge input and output types of a loop are defined by wrapping the input and output types of the loop body with $\mathtt{Opt}[\cdots]$ as appropriate, analogously to the treatment of loop boxes (cf. Definition 4). When an if-box is executed, all input values for which $f_i = 1$ are first checked for ERRORs; if any of the input values is equal to ERROR, execution of the box immediately finishes with a result value of ERROR for each output. If none of the inputs are ERROR, the body G' is executed; its output values are the output values of the if-box. □

Predefined operations that can fail will return optional values ($\mathtt{Opt}[\cdots]$). For every edge in the code graph, if-boxes have to be inserted if necessary to make the codegraph type-consistent.

Two consecutive if-box edges e_1 and e_2 can be *fused* when the flag f_i is set to 1 for at least one input e_2 that is an output of e_1, and e_2 is not reachable from any edge that is reachable from e_1, other than e_1 and e_2 themselves.

Fusing of if-boxes happens by moving the edge e_2 into the body of the if-box e_1, yielding two nested if-boxes (cf. Figure 4). The inputs for the fused edge are the inputs of e_1 and additionally all nodes that are inputs of e_2 but not outputs of e_1; the flags f_i for the additional flags are all set to 0, which means that the outer box does not need to check these inputs against ERROR, because the inner box will do so if necessary. For the nested if-box inside the fused edge, we next check whether that box is still required; first, for every input whose node type is not of the form $\mathtt{Opt}[t]$, the corresponding flag f_i is set to 0. If all flags are set to zero for the inner if-box, the box is elminated by replacing the edge with its body codegraph.

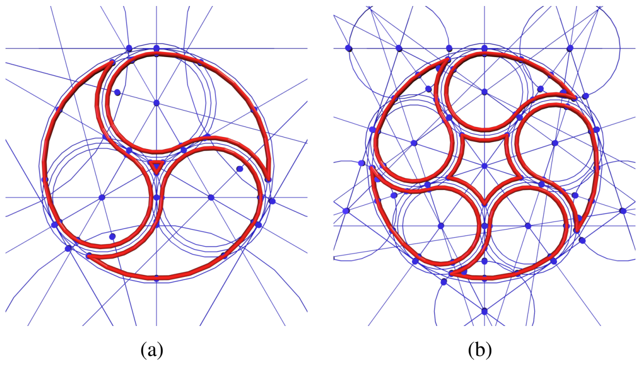


Figure 5. This gothic window construction was created in our test framework using direct manipulation without any code or graph editing. The numnber of repetitions is an input parameter of the model.

V. Examples and Results

In this section, we describe some common modeling operations and their realization within our framework. The examples in this section have been created using direct manipulation on a visible model only (without visualization of the underlying code graph), the concrete user interface is however still in a preliminary stage.

A. Compass & Ruler

Compass and ruler operations have long been used in interactive procedural modeling [18]; these operations are well suited to a side-effect free implementation, and usually return only a single result per operation. Our addition of repetition allows for new constructions (Figure 5).

B. Split Grammars

We can use a methodology similar to Patow et al. [15] to map split grammars to code graphs (see Figure 6). Just as in CGA Shape [12], volumes called *Scopes* are partitioned into smaller volumes by operations split and repeat (replacement as side-effect). split partitions the scope in a predefined number of parts, whereas with repeat the number of parts is determined by the size of the scope at the time of rule application.

C. Optimization Benchmark

We benchmarked the loop fusion and error handling optimizations on three different models. The code graphs are compiled to GML, a language syntactically similar to PostScript. The measurement is based on the number of executable statements, or tokens; this is independent of model parameters (repetition counts) and of the implementation quality of basic operations. See Table I for the results of optimizing loops (Opt A) and loops and error handling (Opt B).

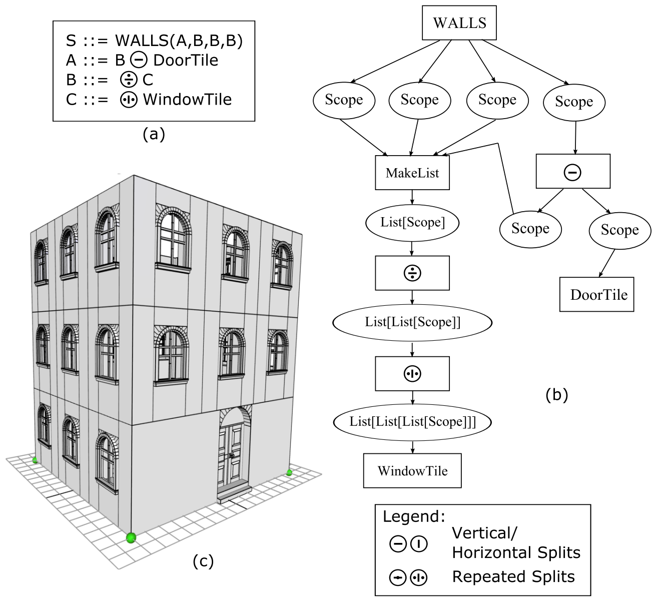


Figure 6. Split grammar example: A simple shape grammar with split and repeat operations can be expressed using a textual description (a). This structure can be mapped to a codegraph (b) and executed (c).

Model	Tokens	Opt A	Opt B
gothic ornament	1322	992	789
simple house	408	258	225
complex facade	69769	30846	24865

Table I
OPTIMIZATION BENCHMARK: EFFECTS OF FUSING LOOPS (OPT A) AND LOOPS & ERROR HANDLING (OPT B) ON MODEL SIZE.

VI. CONCLUSION AND FUTURE WORK

We have presented a formal framework for the representation of procedural models, with a focus on implicit loop representations and improved partial error handling which is particularly suited for direct manipulation of procedural 3D content. We have further described algorithms that allow translation of these models to traditional programming-language based procedural modeling systems.

Using the framework presented in this paper, we believe it will soon be possible to create procedural constructions of medium complexity without writing code or using a visual programming language.

There are many research opportunities for adapting existing techniques to our framework and to the context of direct manipulation procedural modeling. Defining modules or functions is a well-known technique, but it is unknown how well they can be adapted to the special requirements imposed by direct manipulation. Complex procedural 3D models will necessarily suffer from the same problems as complex software does in general; so at some point it will be necessary to investigate methods of 'shape refactoring'.

REFERENCES

[1] S. Havemann, "Generative mesh modeling," Ph.D. dissertation, Technical University Braunschweig, 2005.

[2] Robert McNeel & Associates, "Grasshopper for Rhino3D," [retrieved: 2012, 05]. [Online]. Available: http://www.grasshopper3d.com/

[3] Side Effects Software, "Houdini," [retrieved: 2012, 05]. [Online]. Available: http://www.sidefx.com

[4] T. Green and M. Petre, "When visual programs are harder to read than textual programs," in *Proceedings of ECCE-6*, 1992, pp. 167–180.

[5] CGAL, "Computational Geometry Algorithms Library," [retrieved: 2012, 05]. [Online]. Available: http://www.cgal.org

[6] Adobe Inc., *PostScript Language Reference Manual*, 3rd ed. Addison-Wesley, 1999.

[7] Processing, "Processing," [retrieved: 2012, 05]. [Online]. Available: http://www.processing.org

[8] D. Gould, *Complete Maya programming: an extensive guide to MEL and the C++ API*, ser. Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann Publishers, 2003.

[9] Robert McNeel & Associates, "Rhinoceros 3D," [retrieved: 2012, 05]. [Online]. Available: http://www.rhino3d.com

[10] G. Stiny and J. Gips, "Shape grammars and the generative specification of painting and sculpture," in *The Best Computer Papers of 1971*. Auerbach, 1972, pp. 125–135.

[11] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *Proc. SIGGRAPH 2003*, pp. 669 – 677, 2003.

[12] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," in *ACM SIGGRAPH*, vol. 25, 2006, pp. 614 – 623.

[13] Esri, "CityEngine," [retrieved: 2012, 05]. [Online]. Available: http://www.esri.com/software/cityengine/

[14] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM Comput. Surv.*, vol. 36, no. 1, pp. 1–34, Mar. 2004.

[15] G. Patow, "User-friendly graph editing for procedural buildings," *Computer Graphics and Applications, IEEE*, vol. PP, no. 99, p. 1, 2010.

[16] D. Plump, "Term graph rewriting," in *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*, H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds., 1999, pp. 3–61.

[17] W. Kahl, C. Anand, and J. Carette, "Control-flow semantics for assembly-level data-flow graphs," in *Relational Methods in Computer Science*, ser. Lecture Notes in Computer Science, W. MacCaull, M. Winter, and I. Düntsch, Eds. Springer Berlin / Heidelberg, vol. 3929, pp. 147–160.

[18] Y. Baulac, "Un micromonde de géométrie, cabri-géométre," Ph.D. dissertation, Joseph Fourier University of Grenoble, 1990.

Temperature Based Embedded Programming Algorithm For Conventional Machines Condition Monitoring

Michael Kanisuru Adeyeri
Department of Mechanical Engineering
The Federal University of Technology, Akure
Ondo State, Nigeria
sademike2003@yahoo.co.uk; mkadeyeri@futa.edu.ng

Buliaminu Kareem
Department of Mechanical Engineering
The Federal University of Technology, Akure
Ondo State, Nigeria
bkareem@futa.edu.ng

Adeyemi Adegbemisipo Aderoba
Department of Mechanical Engineering
The Federal University of Technology, Akure
Ondo State, Nigeria
babalojojo@yahoo.com

Sunday Olumide Adewale
Department of Computer Science
The Federal University of Technology, Akure
Ondo State, Nigeria
adewale_olumide@yahoo.co.uk

***Abstract*—** A **temperature-based embedded programming algorithm for conventional machines condition monitoring is being discussed. Machinery health deteriorates day in day out as they are being used for production purposes. If a proper check, maintenance activities and monitoring are not put in place, such machinery would not perform optimally and production efficiency would be affected. Based on this, the present work focuses on programming a temperature sensor AD595 and K-type thermocouple using a C programming language as a means of embedding them into the microcontroller with a real time clock (RTC) incorporated to keep the time of events and temperature readings of the machines' components for effective maintenance plan. The whole design is embedded in production machines to keep monitoring the machines' conditions and behavior as related to temperature induced faults and breakdown matters. The algorithm interprets and reports the fault class name to the operator, diagnosis and proffer solutions based on the embedded decision block. The hardware resulting from the design was tested using a conventional elevator, silos and hammer mill (which are parts of the production set up line for the production of vegetable oil) for a period of four months. The output performance is satisfactory as maintenance decision and machines' health monitoring are optimized.**

Keywords-Temperature; thermocouple; algorithm; conventional machines; condition monitoring; microcontroller.

I. INTRODUCTION

Machinery is required to operate within a relatively close set of limits. These limits, or operating conditions, are designed to allow for safe operation of the equipment and to ensure that equipment or system design specifications are not exceeded. These limits are usually set to optimize product quality and throughput (load) without overstressing the equipment.

Conventional machines are machines which are operated manually. These machines are controlled by cams, gears, levers, or screws. Examples of these machines are Lathe, grinding machine, flaking machine, extruder and just to mention a few. They indeed needed special attention to safe guard or vouch safe for their functionality and optimal performance as compared to the non conventional machines which are controlled automatically by integrated computer.

Manufacturing process objective focuses on efficient production of products with specific shape, acceptable dimensional accuracy and quality. Slight deviation of the machine conditions from a prescribed plan will affect the final product quality and standard. Global industrial competition and the current economic conditions have geared up many manufacturing organizations to improve product quality and cut production costs simultaneously. The requirements for increased plant productivity, safety, and reduced cost on maintenance, have resulted to a growth in popularity of methods for condition monitoring to aid the planning of plant preventive maintenance and operational policies [1].

Malfunctions in equipment and components are often sources of reduced productivity and increased maintenance costs in various industrial applications. For this reason, machine condition monitoring is being pursued to recognize incipient faults in striving towards optimizing maintenance and productivity in conventional machines.

From literature, current production systems have unsatisfactory overall availability due to excessive downtime caused by either quality related issues or machine/component failures [2]. Current single station's mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR) assessment does not reveal overall system performance and dynamic resourcing which is not addressed in today’s total productive maintenance (TPM) and manufacturing execution systems (MES) [2].

In real life applications, measuring temperature is not really a problem especially in practical applications like medical and air conditioning systems. In industrial setting on the other hand, temperature signal conditioning becomes a concept that needs to be given special attention to convention machines. That is why very high precision and accuracy is the ultimate goal of any model that targets programming ambient physical parameters like temperature. In order to achieve the above, an embedded programming model is established to ensure sensing temperature and delivering the accurate result.

II. LITERATURE REVIEW

Condition monitoring is a maintenance process where the condition of equipment with respect to overheating and vibration is monitored for early signs of impending failure. Equipment can be monitored using sophisticated instrumentation such as vibration analysis equipment or the human senses. Where instrumentation is used, actual limits can be imposed to trigger maintenance activity. Condition Monitoring (CM), Predictive Maintenance (PM) and Condition Based Maintenance (CBM) are other terms used to describe this process.

Machine condition monitoring involves the intermittent or continuous collection and interpretation of data relating to the operating condition of those critical components of a machine. Monitoring can greatly reduce maintenance costs by giving adequate notice on pending failures to permit planned repairs, as opposed to costly emergency breakdowns with their attendant lost production, overtime, and expediting costs [3].

Failure occurs when a component, structure, or system is unable to fulfill its intended purpose, resulting in its retirement from usable service. Possible failure modes include component deformation, fracture, surface changes such as cracks, material changes, displacement, leakage, and contamination [4]. Secondary effects, or symptoms, often occur prior to total machine failure, providing indicators for predicting failure onset. Frequently used indicators are vibration signals; noise, heat generation, and particle wear levels as measured in machine lubricants.

Condition-based maintenance is a maintenance strategy that recommends maintenance actions based on the information collected through condition monitoring. And the aim of this strategy is to improve the equipment’s reliability, availability, or its associated life cycle costs [5].

Neelam [6] researched on condition monitoring and fault diagnosis of induction motor using motor current signature analysis. Neelam’s research consists of experimental characterization of rotor faults in induction motors operating under different loading conditions in which the fault algorithm developed monitors the amplitudes over time. And five different faults *vis a viz*, rotor fault, short winding fault, eccentricity fault, bearing fault and load fault are practically implemented and their effects on motor’s current are considered with help of different signal conditioning techniques.

Condition-based maintenance is the diagnosis of component failure or a prognosis of a component’s time to failure [7]. Jasper *et al.* [7] aimed at formulation of empirical postulates regarding the technical system, managerial system and workforce knowledge.

Mahantesh *et al.* [8] developed and tested a condition-monitoring sub-module of an integrated plant maintenance management application based on artificial intelligence (AI) techniques, mainly knowledge-based systems, having several modules, sub modules and sections. The paper collectively deals with the analysis of the state-of-the-art expert systems for diagnosis and maintenance of general-purpose industrial machinery.

Christian [9] researched on competing through maintenance strategies in which the competitive factors were examined. The research work shows that equipment maintenance and reliability management are importantly associated with an organization’s competitiveness and be given adequate attention in the organization’s strategic planning.

Liliane *et al.* [10] evaluated the effectiveness of maintenance strategies under four frameworks that can identify and evaluate the effectiveness of a given maintenance strategy in a company. The four frameworks implored are minimization of manufacturing’s negative potential, achievement of parity (neutrality) with competitors, provision of credible support to the business strategy and aiming a manufacturing-based competitive advantage. And it is found that the framework is applicable and useful for the strategic management of the maintenance function as well as enhancing the competitive advantage of a company.

Jihong [11] modeled a prognostic algorithm for machine performance assessment and their applications explored a performance model through the advantage of logistic regression analysis with maximum likelihood technique and predict the remaining useful life, which would lead to proactive maintenance processes in minimizing downtime of machinery and production in various industries, thus increasing efficiency of operations and manufacturing. They buttress their research with two kinds of application situations with or without enough historical data.

Today's technology has given room for invention of computer numerically controlled machines for production purpose in which integrated circuits and sensors related to maintenance information are embedded to keep track of machines' health for effective and optimized performance of the machines. But there is need to bridge the gap between these computerized and conventional machines age so that the conventional machine users can effectively enjoy the machines' throughput with minimal breakdown and compete in the industrial world without abandoning the conventional machines. Hence the needs for this research work to provide a medium where temperature sensors could be embedded in conventional machines for monitoring their health status and functionality. Therefore the present work describes temperature algorithm for temperature sensor that is needed in assisting, monitoring the behavior of machines' performance or health status and the maintenance activities required for maintaining or preventing temperature related faults or breakdown of conventional machines such as elevator machine, silos and hammer mill.

III. METHODOLOGY

A. *Temperature Model Equation*

Temperature measurement (e.g., temperature-indicating paint, thermograph) helps detect potential failures related to a temperature change in equipment. Measured temperature changes can indicate problems such as excessive mechanical friction (like faulty bearings, inadequate lubrication), degraded heat transfer and poor electrical connections (for examples, loose, corroded or oxidized connections). The below equation1. is used in modeling temperature monitoring for machines.

$$T_i^{ta} = T_i^{o}[1+U_T]_b^{t_n} \tag{1}$$

where

$T_i^{t_a}$: the predicted value of temperature at next planned measuring time

T_i^{o} : the current temperature value

t_n : periodic time numbering of readings

U_T : temperature deteriorating factor and it is expressed as

$$U_T = \frac{T_i - T_o}{T_m^c} \tag{2}$$

T_i : initial temperature value

T_o = measured Temperature before $T_i^{t_a}$

T_m^c = Critical temperature limit level

b : is a function of speed, environmental condition and demand frequency.

Therefore, if $T_m^c \geq T_i^{t_a}$, then maintenance is required, otherwise do not.

B. *Factors influecing the choice of Temperature sensor*

The difference between serial and special manufacturing is only the mechanical structure of the sensors. The criterions used in choosing temperature sensor are as listed below:

- At which position is the temperature to be measured?
- The medium at which temperature is to be determined?
- Which diameter can be installed in the production process?
- Mechanical process connection to be used
- The type of electrical connection
- Which mechanical and thermal stress is the sensor subjected to?

The answers to these questions are the basis for the choice of special temperature sensors for mechanical machines and these have assisted in choosing k-type thermocouple for this research.

A block diagram of the AD594/AD595 thermo-couple signal conditioner IC is shown in Fig. 1. A Type K thermocouple is joined to amplifier differential Pins 1 and 14 so as to reference the local temperature. With the IC also at the local temperature, an ice point compensation circuit develops a voltage equal to the deficiency in the locally referenced thermocouple loop. This voltage is then applied to a second preamplifier whose output is summed with the output of the input amplifier. The resultant output is then applied to the in-put of a main output amplifier with feedback to set the gain of the combined signals. The ice point compensation voltage is scaled to equal the voltage that would be produced by an ice bath referenced thermocouple measuring the IC temperature. This voltage is then summed with the locally referenced loop voltage, the result being a loop voltage with respect to an ice point [12]. The circuit description diagram for this thermocouple is as shown in Fig. 2.

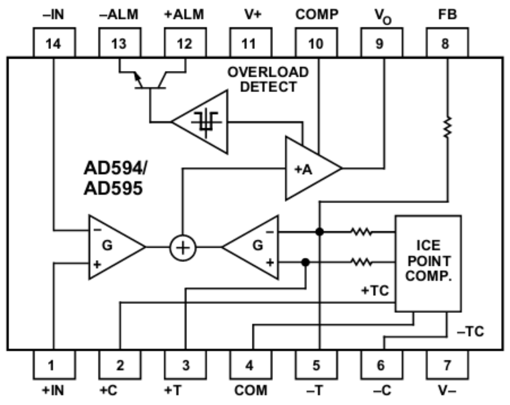


Figure 1. AD595 Block Diagram

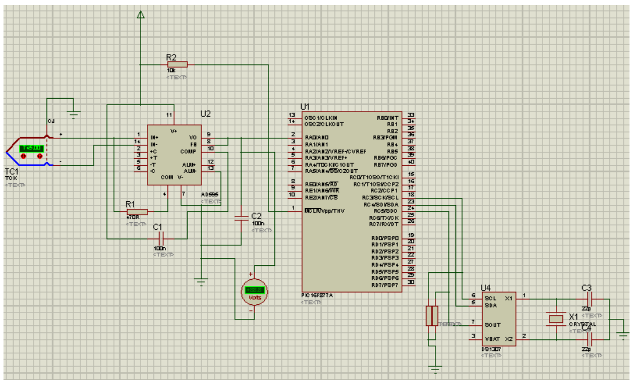


Figure 2. The AD594/AD595 Circuit Description

C. Embedded interface

The following components are used in keeping track of the system:

i. Analog Digital Converter (ADC): The output from the sensor is usually an analog signal that needs to be converted to a discrete signal for proper digitization. In order to achieve this, an analog to digital conversion module is required. For this model, a 10 bit ADC resolution is made used of.

ii. Microcontroller: The microcontroller used for developing this model is Atmel Atmega 16 MCU, this is an 8 bit MCU but with internal 10 bit resolution ADC module. This makes the MCU a choice for this model.

The block diagram shown in Fig. 3 depicts the proposed arrangement of the embedded temperature sensor interface with the machine

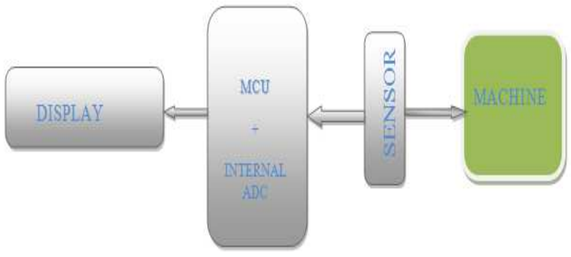


Figure 3. Block diagram of proposed arrangement of temperature sensor interface with conventional machine

D. Model Algorithm

The code for the microcontroller is developed in C language which is not included in this paper. The preliminary exercise carried out in order to get the basis of the conventional machine is to:

i. study the temperature behavioural pattern of the machine to model; and
ii. get the trend of the temperature pattern of the machine system by recording ten to twenty readings of the temperature over a wide period of time and noting the performance rating of the machine against the corresponding temperatures.

While the algorithm developed for the model is as stated below:

Step-1 Setting up: set up display, set up real time clock and set up the internal analog to digital converter
Step-2 Wait for responses
Step-3 Set timer T to zero second and the counter to zero
Step-4 Initialize timer
Step-5 Is timer equals 10seconds?
Step-6 If no, go back to timer initialization, else get the ADC value
Step-7 Counter stores value to RAM
Step-8 Has counter counted to ten values?
Step-9 If no, continue with timer initialization, else counter reset to zero
Step-10 Calculate average temperature
Step-11 Store value in EEPROM, convert temperature value to ASCII and get real time
Step-12 Display value and time
Step-13 Use inference decision block: for example, if $T_m^c \geq T_i^{t_a}$, then maintenance is required, otherwise do not; make inference on machine parts affected and give suggestion
Step-14 Is decision made? If yes, display decision
Step-15 Delay sets in to cater for decision displayed, else back to reset
Step-16 Go to start

This algorithm is therefore translated into flowchart shown in Fig. 4, so as to have a better understanding of the system.

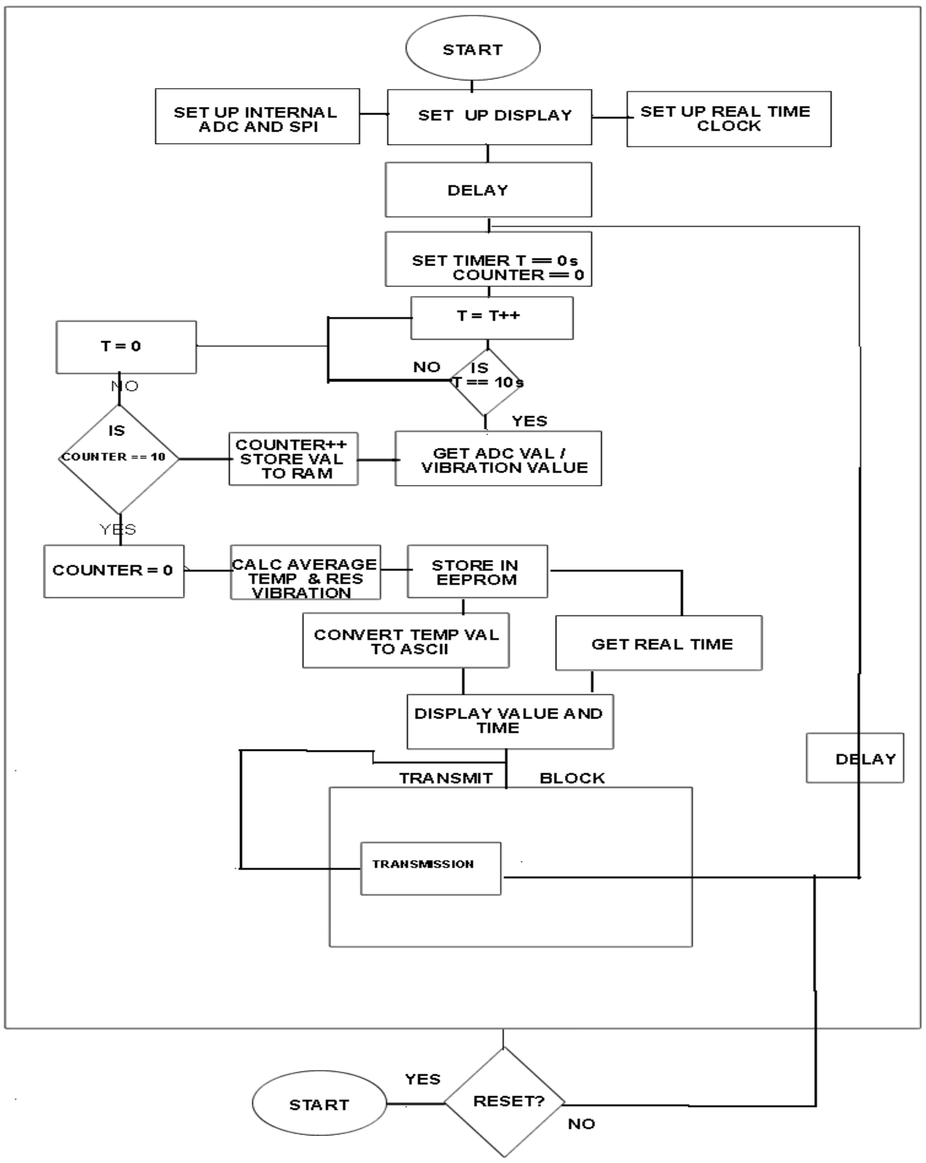


Figure 4. Flowchart of the Temperature sensor Model Algorithm [12]

IV. MODEL VALIDATION

The stated algorithm had been transformed into codes in C programming so as to ensure the practicality of the model (though this aspect will not be discussed in this submission). The whole system is tested by using a conventional elevator, silos and hammer mill which is a part of the production set up line for the production of vegetable oil for a period of four months. The readings taken are as shown in table1. The machine components being affected by temperature are listed with their corresponding readings. N denotes normal machine temperature reading on on-load condition and A connotes abnormal temperature reading.

TABLE I. GEARBOX TEMPERATURE TREND READINGS AND MAINTENANCE MODEL SUGGESTION DECISION OF ELEVATOR, SILO AND HAMMER MILL MACHINES

Sampling dates	**Elevator's Gear box temperature (^{o}C), conditions and decision**			**Silo's Gear box temperature (^{o}C), conditions and decision**			**Hammer mill's Gear box temperature (^{o}C), conditions and decision**		
	Temp.	***conditions***	***Model decision suggestion on maintenance activities***	***Temp.***	***conditions***	***Model decision suggestion on maintenance activities***	***Temp.***	***conditions***	***Model decision suggestion on maintenance activities***
01/11/2011	40.0	N		49.5	N		69.6	A	Poor lubrication
10/11/2011	40.8	N		50.0	N		42.6	N	
15/11/2011	40.8	N		50.0	N		42.7	N	
20/11/2011	46.0	N		50.0	N		43.0	N	
01/12/2011	46.0	N		51.0	N		43.0	N	
05/12/2011	46.2	N		51.3	N		43.2	N	
10/12/2011	46.1	N		52.0	N		44.1	N	
15/12/2011	46.2	N		52.0	N		44.1	N	
22/12/2011	46.5	N		52.0	N		44.5	N	
01/01/2012	46.6	N		52.0	N		46.6	N	
10/01/2012	46.7	N		52.0	N		48.9	N	
20/01/2012	47.0	N		65.0	A	Stop machine. Check for foreign materials and dirt in ball bearing	50.0	N	
01/02/2012	45.9	N		50.0	N		45.9	N	
25/02/2012	75.0	A	Stop machine. Check ball bearing	50.0	N		46.0	N	

As seen from Table 1, on the 1st of November, 2011, the temperature readings sampled from the production process revealed that the temperature values of the elevator gear box, silo gear box and hammer mill gear box read 40 ^{o}C, 49.5 ^{o}C and 69.6 ^{o}C respectively. And the corresponding health status of these machines indicated that they are normal except that of hammer mill which is abnormal. On this note, the model gave a maintenance suggestion clue that the temperature abnormality is as a result of poor lubrication. Looking through the Table 1, it has shown that the model algorithm is correct as it could distinguish between when the machine conditions are normal, abnormal and display maintenance suggestion messages which are valid and result oriented.

V. CONCLUSION AND FUTURE WORK

The efficient and optimum performances of machines lie on the prompt and on-line monitoring of the machine components and behavior. The research work under discussion has really shown that temperature which is one of the key factors affecting machine performance could be monitored to aid maintenance plan. It is to be hoped that the tool herein described will assist conventional equipment maintenance and personnel in decision making as they progress towards optimizing maintenance plans.

The present work could not give the overall true picture of the machine health status, therefore the future work would entail building an integrated sub-system hardware that incorporates other machine condition monitoring indices such as vibration and machine wear sensors which will assist in having a full diagnosis of the machines, and thus enhancing its functionality.

REFERENCES

[1] Y. Zhan, V. Makis, and A.K.S. Jardine, "Adaptive model for vibration monitoring of rotating machinery subject to random deterioration" Journal of Quality in Maintenance Engineering Vol. 9 No. 4, 2003 pp. 351-375 (http://dx.doi.org/10.1108/13552510310503222) [retrieved: July, 2012]

[2] A. Alhad, C. Xiaohui; L. Jay, N. Jun, and Y. Ziming, "Optimized Maintenance Design for Manufacturing Performance improvement using simulation." Proceedings of the 40th Conference on Winter Simulation Conference, IEEE, 2008, pp. 1811-1819.

[3] G.M. Knapp and H.P. Wang, "Machine fault classification: a neural network approach", International Journal of Production Research, Vol. 30 No. 4, 1992, pp. 811-823. (http://www.emerald-library.com) [retrieved: June, 2012]

[4] R.H. Lyon, Machinery Noise and Diagnostics, 1987, Butterworth, Boston, MA

[5] A.K.S. Jardine, T. Joseph, and D. Benjevic, "Optimizing Condition-based Maintenance decisions for equipment subject to Vibration Monitoring", Journal of Quality in Maintenance Engineering Vol. 5 No. 3, 1999, pp. 192-202. (http://www.emerald-library.com) [retrieved: June, 2012]

[6] M. Neelam, "Condition Monitoring and Fault Diagnosis of Induction Motor Using Motor Current Signature Analysis" P.hD Thesis, National Institute of Technology, Kurukshetra (Haryana) India, 2010.

[7] V. Jasper, K. Warse, and W. Hans, "Managing Condition based Maintenance Technology, a multiple case study in the process industry", Journal of Quality in

Maintenance Engineering Vol. 17 No. 1, 2011, pp. 40-62. [http://www.emerald-library.com, June, 2012]

[8] N. Mahantesh, A. Ramachandra, and A.N. Satosh Kumar "Artificial Intelligence-based Condition Monitoring for Plant Maintenance; Assembly Automation Vol. 28, No. 2, 2008, pp. 143–150.

[9] N. Christian Madu, "Competing Through Maintenance Strategies" International Journal of Quality & Reliability Management, Vol. 17, No. 9, 2000, pp. 938-948. MCB University Press.

[10] P. Liliane, K.P. Srinivas, and V. Ann, 'Evaluating the Effectiveness of Maintenance Strategies" Journal of Quality in Maintenance Engineering Vol. 12 No. 1, 2, 2006, pp. 7-20. Emerald Group Publishing Ltd. March, 2010

[11] Y. Jihong, K. Muammer, and L. Jay, "A prognostic algorithm for machine performance assessment and its application" Production Planning & Control, Taylor and Francis Group, Vol. 15, No. 8, December 2004, pp. 796–801.

[12] M. Joe, "Application note on Thermocouple Signal Conditioning Using the AD594/AD595" One Technology Way Norwood, USA [http://www.analog.com November, 2011]